**1. Einleitung**

Im Rahmen dieses Projekts habe ich eine digitale Version des bekannten Brettspiels *Mensch ärgere dich nicht* entwickelt. Ziel war es, das Spiel als GUI-Anwendung in Java umzusetzen möglichst nah am Original. Das Projekt entstand im Rahmen einer Projektwoche, wobei sowohl die technische Umsetzung als auch Planung und Dokumentation im Fokus standen.

*Mensch ärgere dich nicht* ist ein klassisches Gesellschaftsspiel, das fast jeder kennt. Gerade deshalb war es spannend, eine eigene Version davon zu programmieren: Wie übersetzt man einfache Spielregeln in sauberen Code? Wie lässt sich ein Spielstand speichern? Und was passiert, wenn der Nutzer plötzlich eine ungültige Eingabe macht?

In dieser Dokumentation will ich nicht nur zeigen, wie das Spiel funktioniert, sondern auch, wie ich bei der Planung, Umsetzung und beim Testen vorgegangen bin und was ich dabei gelernt habe. Der Spaß am Entwickeln kam dabei nicht zu kurz!

**2. Anforderungen**

Bevor man einfach drauflos programmiert, sollte man sich erst mal überlegen, was das fertige Programm eigentlich alles können soll und was nicht. Genau darum ging es bei den Anforderungen. Ich habe sie in zwei große Bereiche aufgeteilt: **funktionale** und **nicht-funktionale Anforderungen**.

**Funktionale Anforderungen**

Das sind die Dinge, die das Spiel *wirklich* können muss. Also alles, was direkt mit dem Spielablauf zu tun hat. Dazu gehören zum Beispiel:

* **Spieleranzahl flexibel**: Das Spiel soll mit 2 bis 4 Spielern funktionieren.
* **Anpassbares Spielfeld**: Auch die Länge der Wege oder die Anzahl der Spielfiguren pro Spieler sollte man eventuell anpassen können.
* **Spielverlauf nach bekannten Regeln**: Figuren rauswerfen, nur mit einer 6 starten, usw.

Kurz gesagt: Es soll sich wie das klassische Spiel anfühlen, nur eben digital.

**Nicht-funktionale Anforderungen**

Hier geht’s mehr um die "Rahmenbedingungen", also Dinge, die das Spiel angenehmer, sicherer oder verständlicher machen:

* **Einfaches Spielprinzip**: Jeder soll’s verstehen, auch ohne dicke Anleitung.
* **Eingabesicherheit**: Das Spiel soll nicht abstürzen, wenn jemand mal etwas Falsches eingibt.

**Weitere Anforderungen**

Ein paar Extras habe ich mir auch vorgenommen.

* **JUnit-Tests**: Um zu prüfen, ob die Kernfunktionen korrekt laufen.
* **Diagramme und Planungsübersichten**: Zum Beispiel ein Gantt-Diagramm zur Zeitplanung.
* **Nutzwertanalyse**: Um zu schauen, welche Features wirklich sinnvoll sind – besonders bei mehreren Ideen.

Diese Anforderungen haben mir später als roter Faden gedient: Alles, was ich gebaut habe, musste sich an dieser Liste messen lassen.

**3. Projektziele**

Klar, ein Spiel zu programmieren klingt erstmal nach Spaß (und das war's auch). Aber damit das Ganze nicht aus dem Ruder läuft, habe ich mir vorher genau überlegt: **Was will ich eigentlich erreichen?** Und fast noch wichtiger: **Was will ich *später einbauen*?**

**Was soll am Ende rauskommen?**

Das Hauptziel war konkret:  
Ein funktionierendes, digitales Mensch *ärgere dich nicht* steuerbar über die Konsole bis GUI läuft, mit mehreren Spielern und einem Spielverlauf, der dem Brettspiel möglichst nahekommt.

Das bedeutet konkret:

* Der Spieler soll über die Konsole Figuren auswählen, würfeln und ziehen können.
* Es soll möglich sein, andere Figuren rauszuwerfen.
* Es soll einen Sieger geben, wenn alle 4 Figuren im Ziel sind.

Dazu kommt: Das Spiel soll stabil laufen, verständlich sein und die fehlerhaften Eingaben des Nutzers ordentlich abgefangen werden. Also kein Crash bei einer falschen Zahl oder einem Buchstaben, wo keiner hingehört.

**Was gehört *nicht* dazu? (Abgrenzung)**

Damit das Projekt nicht zu groß wird, habe ich bewusst ein paar Dinge **nicht** umgesetzt:

* ❌ **Keine grafische Oberfläche (GUI)**: Alles läuft über die Konsole. Schlicht, aber funktional.
* ❌ **Kein Online-Multiplayer**: Gegeneinander spielen geht nur lokal.
* ❌ **Keine Computer-Gegner (Bots)**: Jeder Spieler wird manuell gesteuert, keine KI.
* ❌ **Keine anderen Modis:** Habe mehrere Modis wie 2 vs 2 z.B. oder so geplant.

Diese Einschränkungen haben geholfen, das Projekt realistisch und im vorgegebenen Zeitrahmen durchziehbar zu halten.

**4. Projektplanung und Vorgehen**

Ein Projekt einfach drauflos zu coden ist zwar manchmal verlockend, aber meistens keine gute Idee. Damit ich nicht irgendwann den Überblick verliere oder in einer Sackgasse lande, habe ich mir vorher einen groben Plan gemacht. Und ja der wurde im Lauf des Projekts natürlich auch ein paar Mal angepasst (wie das halt so ist ).

**Phase 1: Brainstorming & Planung**

Ganz am Anfang stand die Frage: *Wie soll mein digitales „Mensch ärgere dich nicht“ eigentlich aussehen?* Ich habe mir also zuerst überlegt:

* Welche Klassen und Objekte brauche ich?
* Was passiert im Spiel nacheinander (Spielstart → Würfeln → Ziehen usw.)?
* Wie könnte der Code logisch aufgebaut sein?

Zur Übersicht habe ich auch ein einfaches Gantt-Diagramm erstellt. Damit konnte ich ungefähr planen, wie lange jede Phase dauern soll und ob ich im Zeitrahmen bleibe.

Außerdem habe ich eine **Nutzwertanalyse** gemacht: Ich hatte mehrere Ideen im Kopf (z. B. Bots, GUI, Datenbank), aber mir war klar, dass ich nicht alles umsetzen kann. Also habe ich bewertet, was den größten Mehrwert bringt – und danach entschieden.

**Phase 2: Coding & Umsetzung**

Hier ging’s dann ans Eingemachte: Klassen erstellen, Methoden schreiben, testen und das Ganze von vorne. Dabei habe ich möglichst modular gearbeitet also lieber mehrere kleine Methoden als eine einzige riesige. Das hat das Testen (und das Verstehen meines eigenen Codes ) deutlich einfacher gemacht.

**Phase 3: Testphase**

Sobald der erste grobe Ablauf stand, ging es ans Testen. Ich habe viele Dinge manuell ausprobiert, aber auch **JUnit-Tests** für zentrale Funktionen geschrieben. Zum Beispiel:

* Wird korrekt gewürfelt?
* Springt eine Figur an die richtige Position?
* Wird jemand rausgeworfen, wenn er überholt wird?

Manche Bugs haben sich erst spät gezeigt zum Beispiel, wenn zwei Figuren auf dem gleichen Feld stehen sollten oder beim Ziel, das man z.B. nicht überspringen kann. Aber auch das gehört dazu.

**5. Technische Umsetzung**

Welche Technologien kamen zum Einsatz, wie ist das Programm aufgebaut und was steckt eigentlich unter der Haube?

**Architektur: Alles dreht sich um Objekte**

Das Spiel wurde in **Java** programmiert, weil es objektorientiert ist und sich gut eignet, um Spiellogik strukturiert abzubilden.

Die wichtigsten Klassen waren:

* **Spieler**: Enthält Name, Spielfiguren, Positionen usw.
* **Figur**: Eine einzelne Spielfigur mit Position.
* **Spielfeld**: Logik zum Bewegen und Prüfen von Kollisionen.
* **Würfel**: Zufallszahl von 1–6.
* **SpeicherManager**: Ermöglicht das **Speichern und Laden** des Spielstands (über einer Textdatei).
* **Main**: Einstiegspunkt – hier wird alles zusammengeführt.

Der Ablauf war klar strukturiert: Spiel starten → Spieler reihum → Würfeln → Figur wählen → Ziehen → Kollisionen prüfen → Nächster Spieler.

**Eingesetzte Technologien und Tools**

* **Java**: Hauptsprache des Projekts.
* **Scanner**: Für die Konsoleneingaben (Spielernamen, Figurenwahl usw.).
* **JUnit**: Zum Testen der Spiellogik.
* **Textdatei**: Als einfache Speichereinheit für den Spielstand.
* **IDE (z. B. IntelliJ oder Eclipse)**: Zum Programmieren, Testen und Debuggen.

Zusätzlich kamen einfache **Textausgaben in der Konsole** zum Einsatz, um Menüführung, Spielstatus und Spiellogik sichtbar zu machen. Es gibt also keine grafische Oberfläche, aber dafür ist der Code dafür umso übersichtlicher.

**Besonderheiten & Herausforderungen**

Ein paar Dinge waren beim Coden besonders spannend:

* Die Spiellogik (z. B. wann darf eine Figur starten?) musste genau wie im Brettspiel funktionieren.
* Das **Rauswerfen anderer Figuren** erfordert ein genaues Prüfen der Positionen.

**6. Spielverlauf und Benutzerinteraktion**

Das Spiel *Mensch ärgere dich nicht* kennt fast jeder und genau das wollte ich in der Konsolenversion auch rüberbringen. Deshalb habe ich den Ablauf so programmiert, dass man sich wie beim Brettspiel fühlt nur eben mit Tastatur statt Würfelbecher.

**Spielstart**

Zu Beginn wird man durch ein kleines Menü geführt:

1. Neues Spiel starten oder gespeichertes Spiel laden
2. Anzahl der Spieler eingeben (2–8)
3. Spielernamen eingeben

Danach werden für jeden Spieler automatisch vier Spielfiguren angelegt. Die Startpositionen richten sich nach der klassischen Anordnung (z. B. „unten links“ beginnt Spieler 1 usw.).

**Rundenablauf**

Der Spielablauf ist rundenbasiert:

* Jeder Spieler kommt reihum dran.
* Man muss Enter zum Würfeln drücken. Jeder Spieler hat immer 3 Versuche, bis man eine 6 gewürfelt hat und alle raus sind.
* Wenn eine Figur raus darf oder ziehen kann, wird man zu einem weiteren Menü geführt.
* Die gewählte Figur zieht entsprechend der gewürfelten Augenzahl vorwärts.

Falls die Figur dabei auf einem Feld landet, auf dem schon eine andere gegnerische Figur steht, wird diese wie im Originalspiel rausgeworfen und muss wieder zurück in den Startbereich.

**Eingabe & Bedienung**

Gesteuert wird alles über die **Konsole**:

* Auswahl per Zahlentaste (z. B. „1“ für Figur 1)
* Menüs und Aufforderungen per Text
* Rückmeldungen und Hinweise nach jeder Aktion

**Spielende**

Sobald ein Spieler alle seine vier Figuren ins Ziel gebracht hat, ist das Spiel beendet. Es wird dann ein Glückwunsch angezeigt und das Programm endet.

**7. Tests und Qualitätssicherung**

Nur weil ein Programm *irgendwie läuft*, heißt das noch lange nicht, dass es auch richtig funktioniert. Deshalb habe ich auch beim Testen nicht einfach auf Glück vertraut, sondern mir gezielt angeschaut, ob alles so läuft, wie es soll.

**Manuelles Testen**

Am Anfang habe ich viele Funktionen einfach per Hand ausprobiert. Also klassisch:

* Spiel starten
* Würfeln
* Figuren ziehen
* Rauswürfe provozieren
* Ungültige Eingaben machen (z. B. Buchstaben statt Zahlen)

Dabei habe ich schon früh gemerkt: Ohne gute Eingabeprüfung kann man das Spiel ziemlich schnell zum Absturz bringen   
Also habe ich früh angefangen, möglichst viele Sonderfälle abzufangen.

**JUnit-Tests**

Für zentrale Spielfunktionen habe ich dann automatisierte Tests mit **JUnit** geschrieben – also kleine Testprogramme, die prüfen, ob eine Methode korrekt funktioniert.

Damit kann man z. B. sicherstellen, dass:

* Der Würfel nie eine „0“ oder „7“ ausspuckt
* Figuren auch an der richtigen Stelle landen

**Fehler, die aufgetaucht sind**

Ein paar Klassiker, die mir im Laufe des Projekts begegnet sind:

* Eigene Figuren haben sich gegenseitig gekickt
* Zwei Figuren konnten sich „überschneiden“, ohne dass eine rausgeworfen wurde
* Eine Figur im Ziel konnte von einer anderen Figur übersprungen werden.

Aber: Jeder Bug, den man findet und behebt, macht das Spiel am Ende besser und stabiler.

**9. Fazit**

Es läuft nicht immer alles glatt, aber man lernt mit jeder Zeile Code dazu.

**Was habe ich gelernt?**

* **Objektorientiertes Denken**: Anfangs war es gar nicht so einfach, zu überlegen, was jetzt eine eigene Klasse wird und was nicht. Aber je mehr ich damit gearbeitet habe, desto mehr hat das Konzept „Objekte = Dinge im Spiel“ Sinn ergeben.
* **Fehler gehören dazu**: Bugs gehören einfach zum Prozess. Viel wichtiger ist, *wie* man damit umgeht. Ich habe gelernt, systematisch zu testen, Fehlermeldungen zu verstehen und Stück für Stück zu verbessern.
* **Planung zahlt sich aus**: Auch wenn man am Anfang denkt „Ach, das geht schon“, hilft es enorm, wenn man sich vorher Gedanken macht – über Aufbau, Struktur, Zeitaufwand. Meine kleine Projektplanung hat mir dabei geholfen.

**Was lief gut?**

* Die Spiellogik funktioniert und das war mein Hauptziel.
* Die Konsole ist simpel, aber effektiv für das, was das Spiel können soll.
* Das Spiel ist stabil und kann sogar gespeichert und später fortgesetzt werden.

**Was war knifflig?**

* Das Rauswerfen anderer Figuren war tricky vor allem, wenn mehrere auf einem Feld standen.
* Und natürlich: Eingabeprüfungen denn die Nutzer sind oft kreativer als gedacht

**Hat es Spaß gemacht?**

Ja! Auch wenn’s manchmal frustrierend war, überwiegt am Ende ganz klar der Spaß. Es ist einfach cool, am Ende ein funktionierendes Spiel vor sich zu haben, das man selbst gebaut hat – inklusive eigenem Code, Testfällen und Speichersystem.

**10. Ausblick**

Auch wenn das Spiel technisch fertig ist und gut läuft, heißt das nicht, dass da nicht noch Luft nach oben wäre. Ganz im Gegenteil: Während der Entwicklung sind mir viele Ideen gekommen, wie man das Projekt noch weiterentwickeln könnte.

**Grafische Oberfläche (GUI)**

Der offensichtlichste Schritt wäre eine **richtige Benutzeroberfläche**. Statt Konsolentext könnte man Figuren auf einem digitalen Spielbrett anzeigen, Buttons einbauen, Animationen zeigen usw.  
Das würde das Spiel nicht nur schöner machen, sondern auch intuitiver für Leute, die nicht so technikaffin sind.

**Online-Multiplayer**

Ein weiteres Feature wäre ein **Netzwerkmodus**, bei dem man über das Internet gegeneinander spielen kann. Dafür müsste man Serverkommunikation einbauen aber es wäre ein cooler nächster Schritt und würde das Projekt auf ein ganz neues Level bringen.

**KI-Gegner (Bots)**

Aktuell braucht man mindestens zwei „echte“ Spieler. Eine Erweiterung wären **einfache Computergegner**, die automatisiert würfeln und ziehen. Das wäre auch eine gute Gelegenheit, sich mal mit **künstlicher Intelligenz** (oder zumindest einfacher Logik) auseinanderzusetzen.

**Mobile-Version oder Web-App**

Langfristig könnte man sich sogar vorstellen, das Spiel als **App für Smartphones** oder als **Webanwendung im Browser** umzusetzen. Dann wäre es überall spielbar – vielleicht sogar mit Touchsteuerung.

Kurz gesagt: Das Projekt war ein super Einstieg in objektorientierte Spieleentwicklung – und die Möglichkeiten, es auszubauen, sind riesig. Ob ich es tatsächlich nochmal erweitere? Wer weiß aber die Ideen sind auf jeden Fall da.