Von der Aufgabe zum Code

Mario Bodemann

Agenda

- Wiederholung
- Aufgabenstellung
- Aufspalten der Aufgabe
- Vom Diagramm zum Code
- “richtiges” Programmieren
- Zusammenfassung

Agenda

- **Wiederholung**
- Aufgabenstellung
- Aufspalten der Aufgabe
- Vom Diagramm zum Code
- “richtiges” Programmieren
- Zusammenfassung

Rückblick

- Was ihr schon könnt:
  - Variablen
  - Schleifen
  - Methoden
  - Java API

Rückblick

```
// Hauptmethode
public static void main( String[] args){
  // erzeuge einen String
  String aString = “Hello World #”;

  // gib den String zufällig oft aus
  int iEnd = (int)(Math.random() * 100);
  for( int i = 0; i < iEnd; ++i){
    // ausgabe
    System.out.println( aString + i );
  }
} // ENDE
```

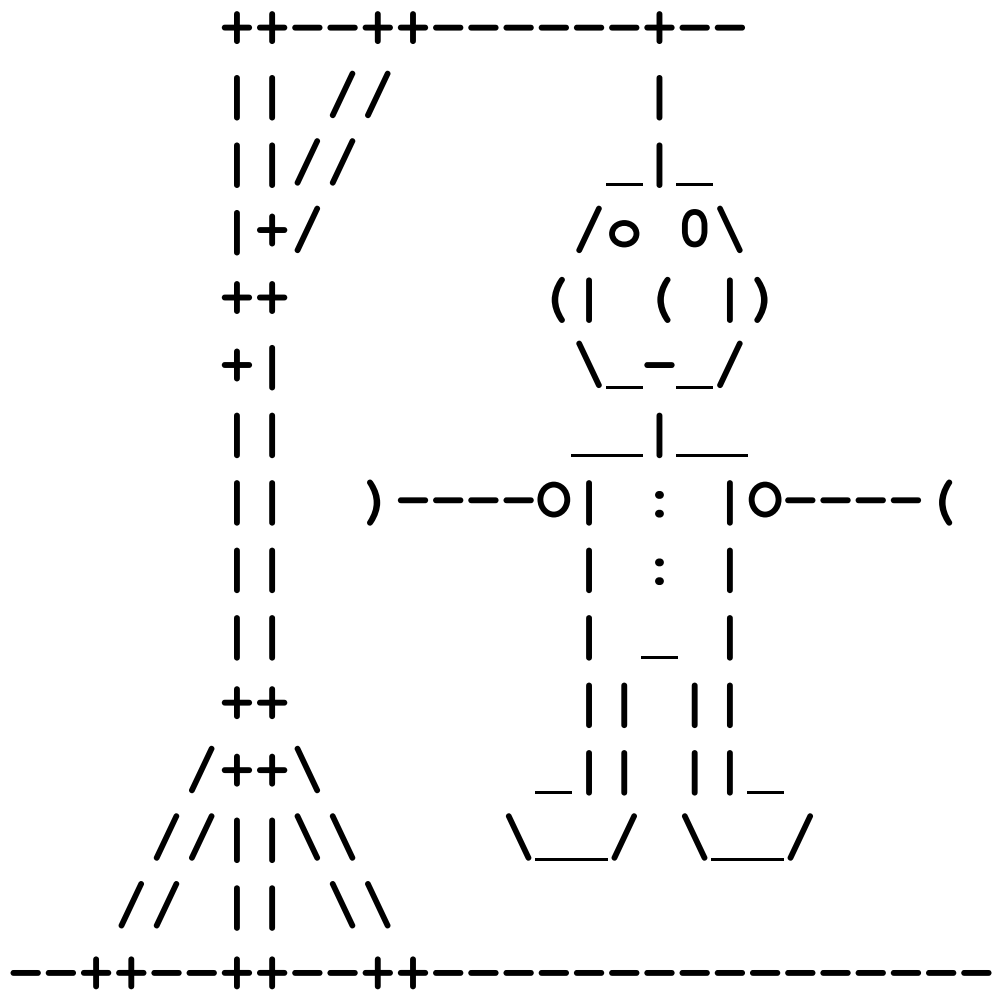
Rückblick – Was wisst Ihr

- Bisher: Faktenwissen
  - Was ist eine Variable? Was ist eine Schleife? etc.
- Heute: Wie ?
  - Wie beginnt man mit einer Aufgabe?
  - Wie löst man Teile von ihr?
  - Wie setze ich das in Java um?

Agenda

- Wiederholung
- **Aufgabenstellung**
- Aufspalten der Aufgabe
- Vom Diagramm zum Code
- “richtiges” Programmieren
- Zusammenfassung

Die Aufgabenstellung (Bild)



H A N _ _ A N

Die Aufgabenstellung (Text)

- **Schreibe das Programm “Hangman”**
  - Spieler muss Wort erraten, solange Hangman noch nicht komplett
- Das Spiel soll beliebig oft hintereinander gespielt werden dürfen
- Zu jeder Zeit soll der Spieler die bereits gewählten Buchstaben angezeigt bekommen
- Auch soll er stets den Status des Hangmans sehen können

Agenda

- Wiederholung
- Aufgabenstellung
- **Aufspalten der Aufgabe**
- Vom Diagramm zum Code
- “richtiges” Programmieren
- Zusammenfassung

Schritt I: Aufgabe verstehen

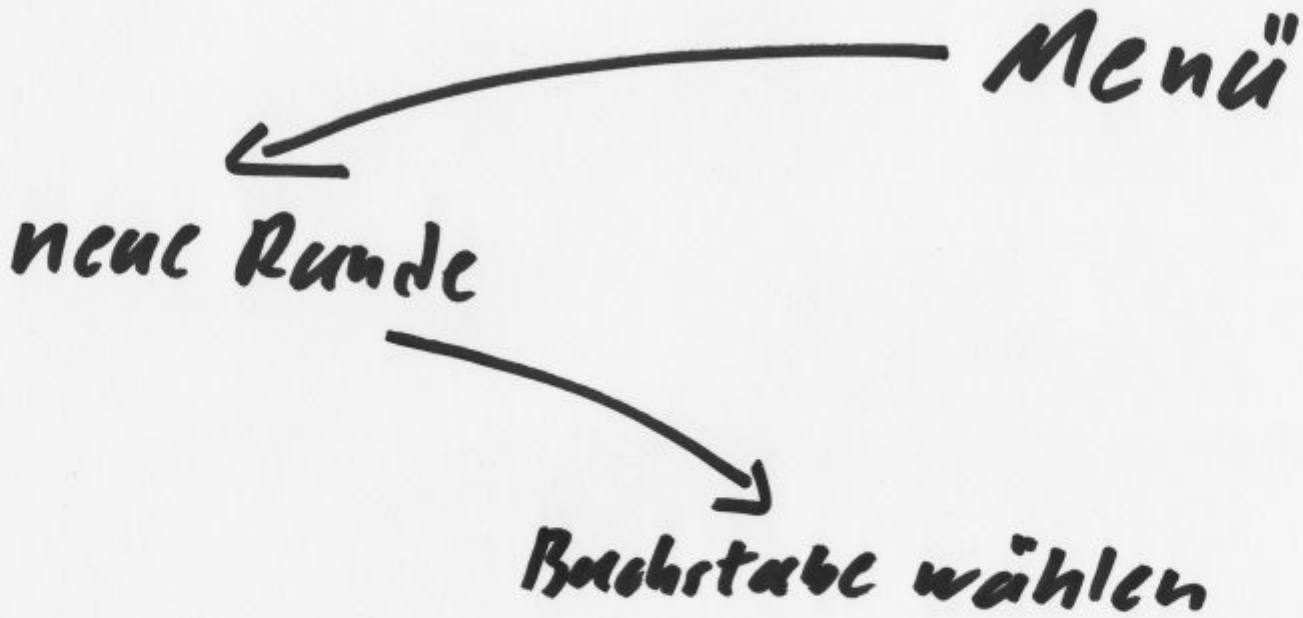
- **Was sind die Regeln von Hangman?**
  - Programm wählt zufällig ein Wort
  - Spieler gibt Buchstaben ein
    - Falscher Buchstabe führt zu einem Strich
    - Richtiger Buchstabe wird eingetragen
  - Spiel wird beendet wenn Wort erraten oder Hangman komplett

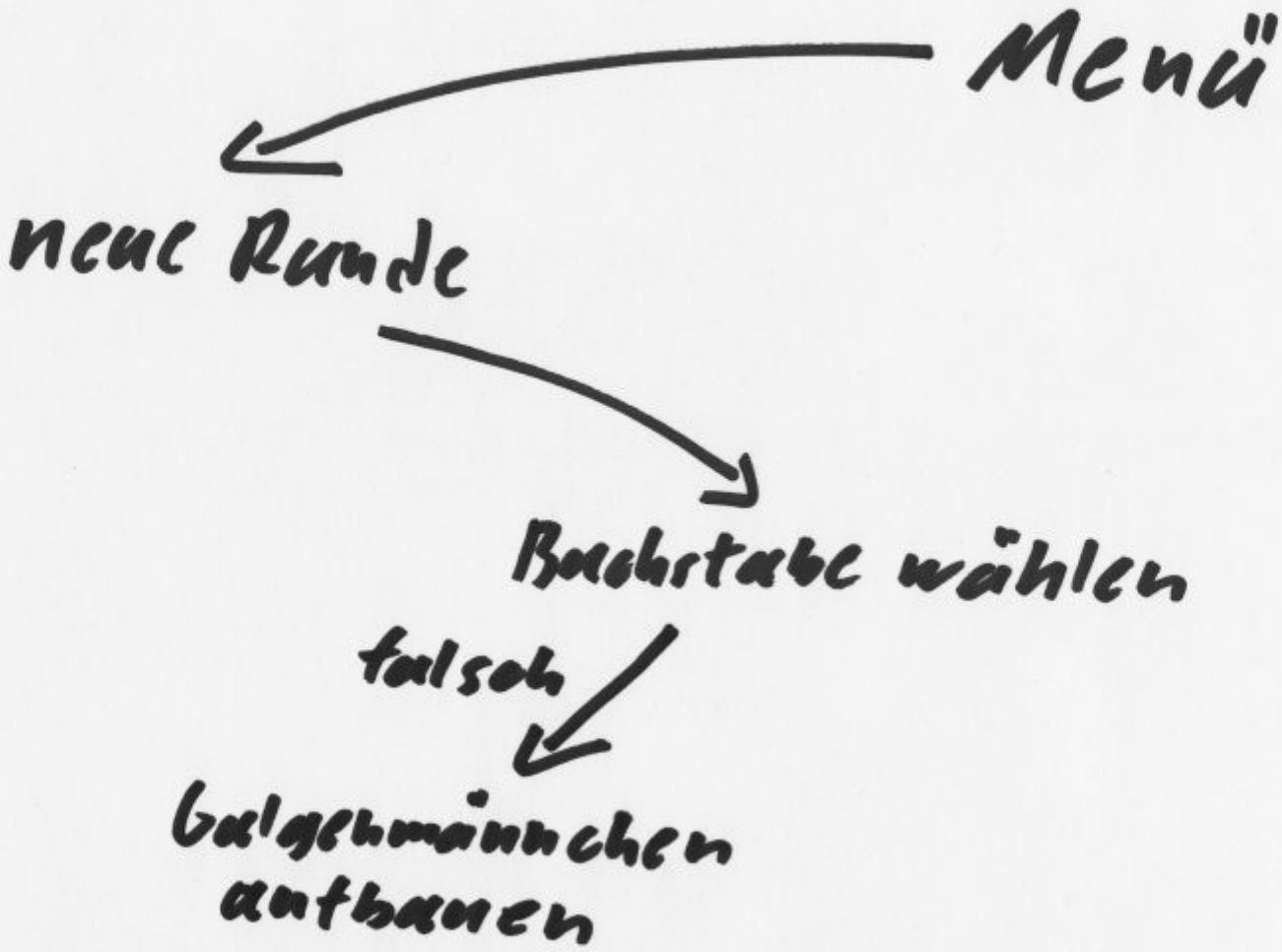
Schritt II: Ablauf nachvollziehen

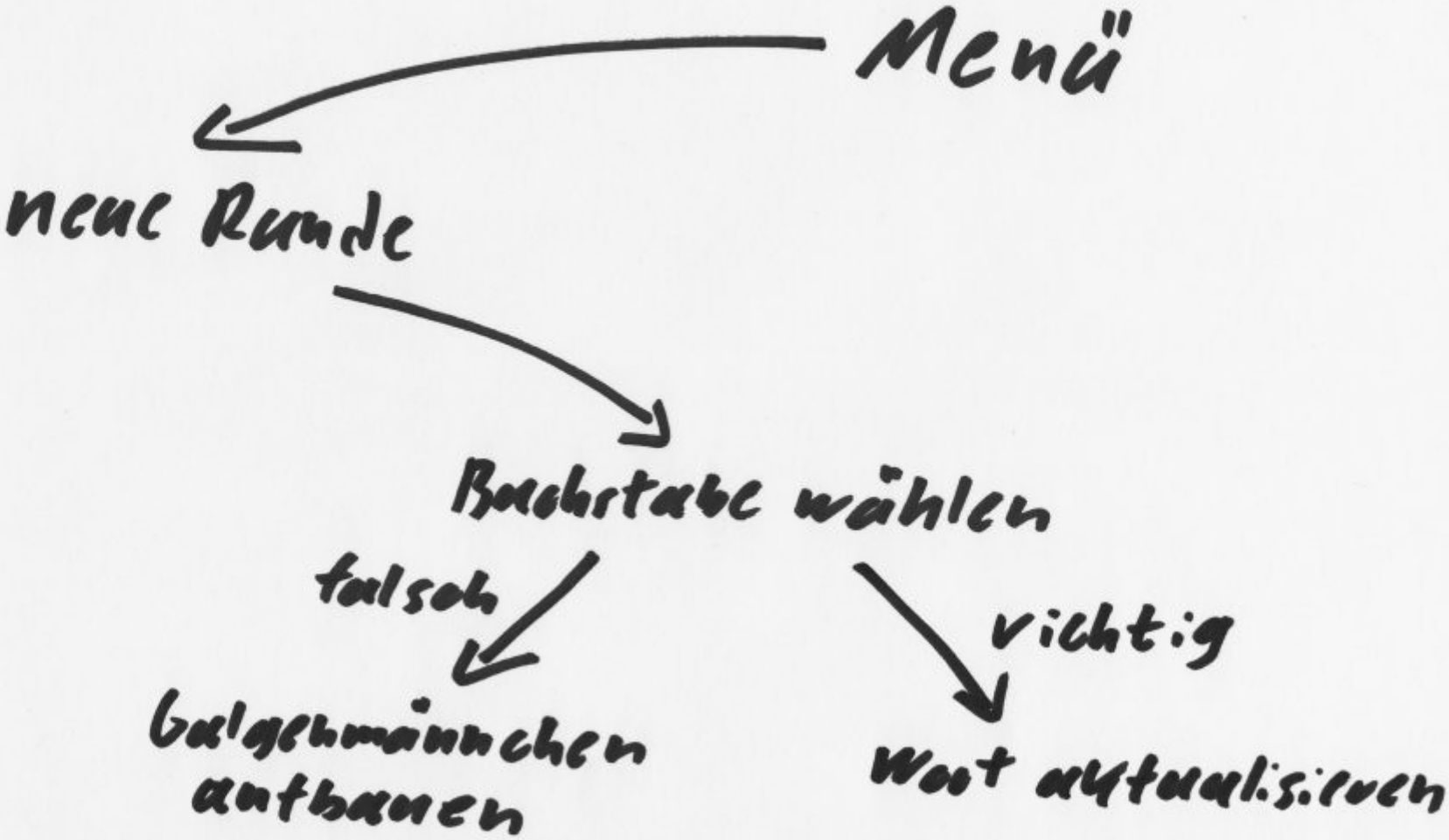
- **Am besten grafisch (Stift und Papier)**
  - Sorgt für mehr Durchblick
  - Ordnet den Ablauf des Programms
  - Definiert Teilbereiche des Spieles
    - Nützlich für Gruppenarbeiten
    - Teile und Hersche Prinzip

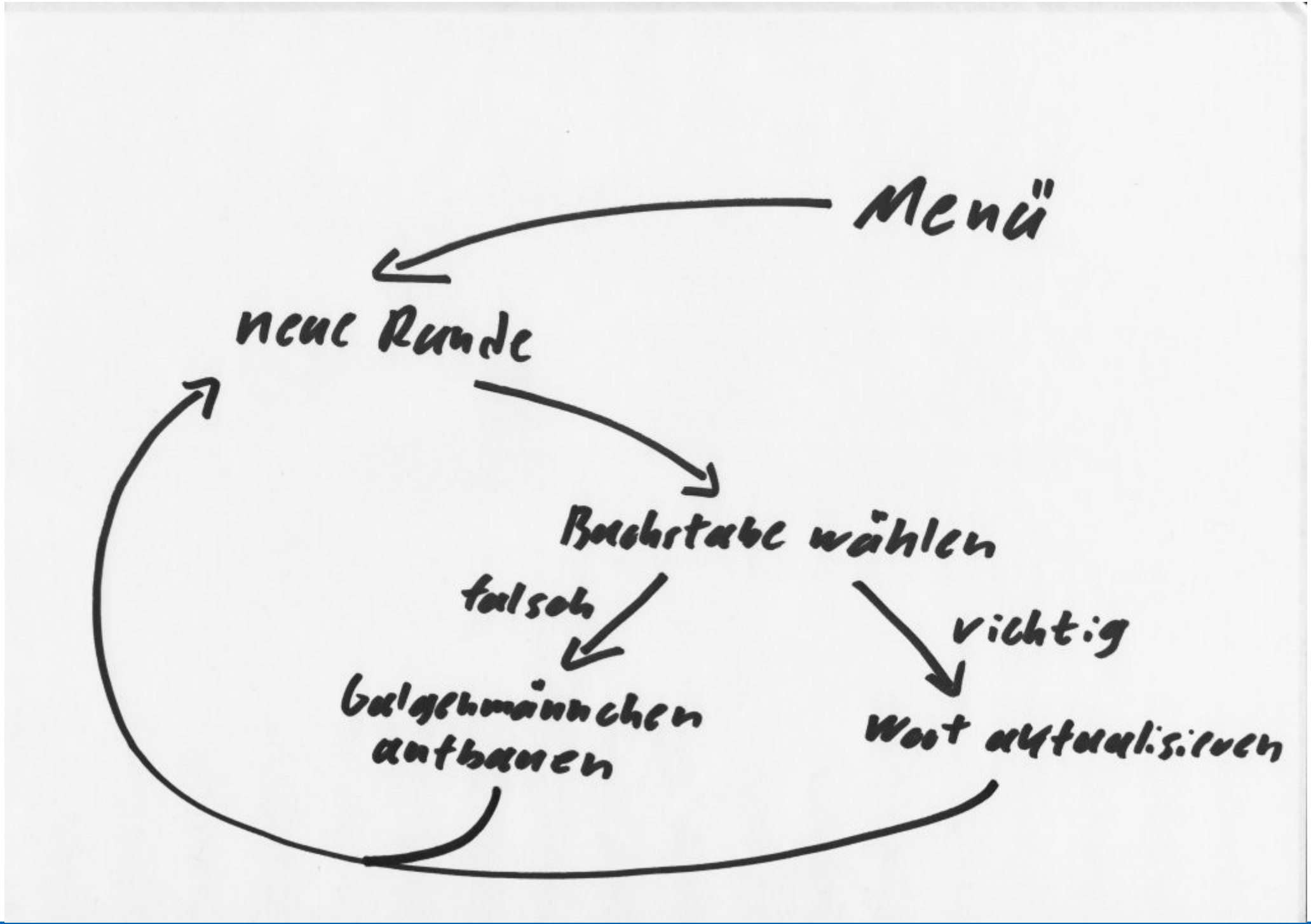
Menü

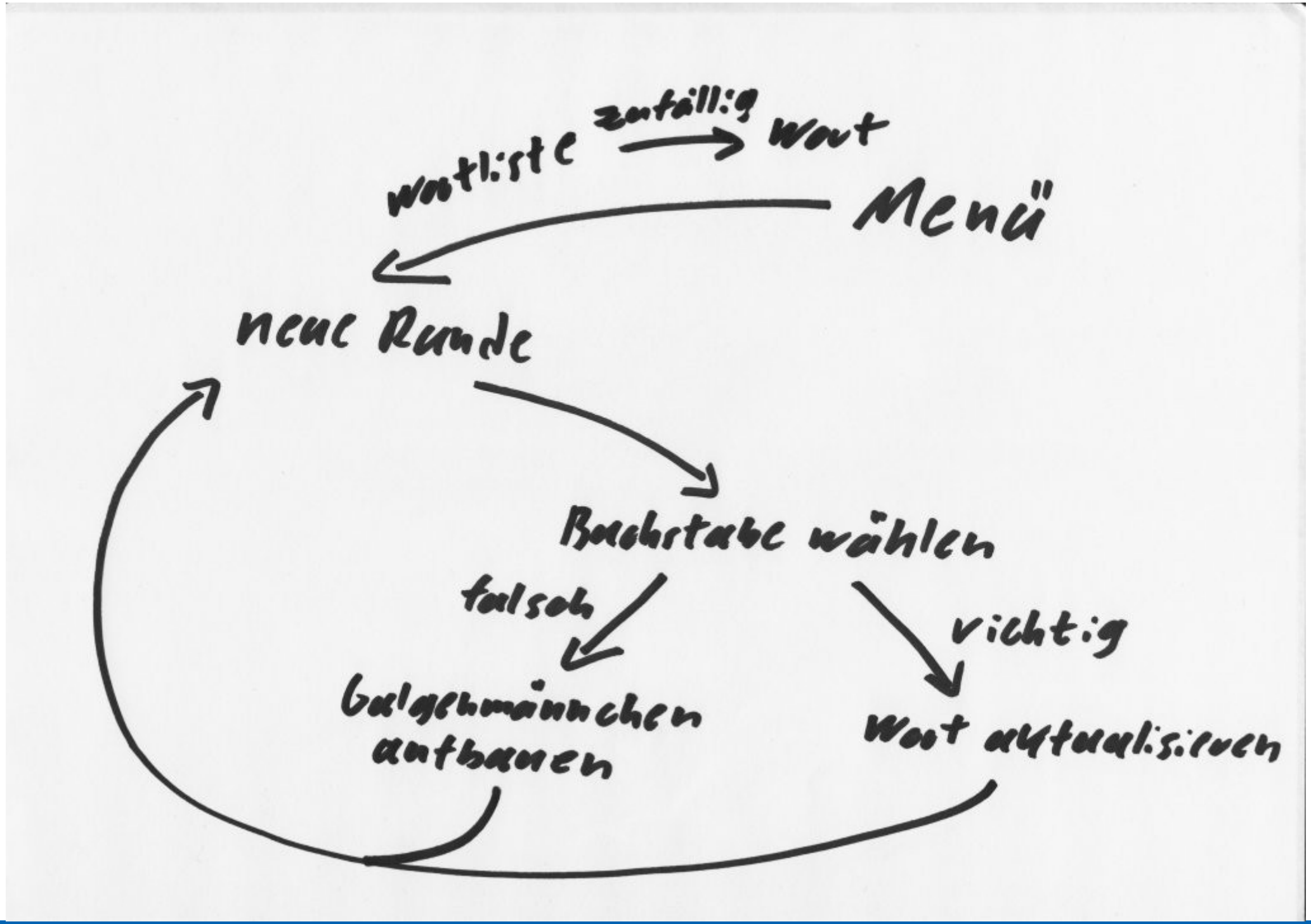

Menü
neue Runde

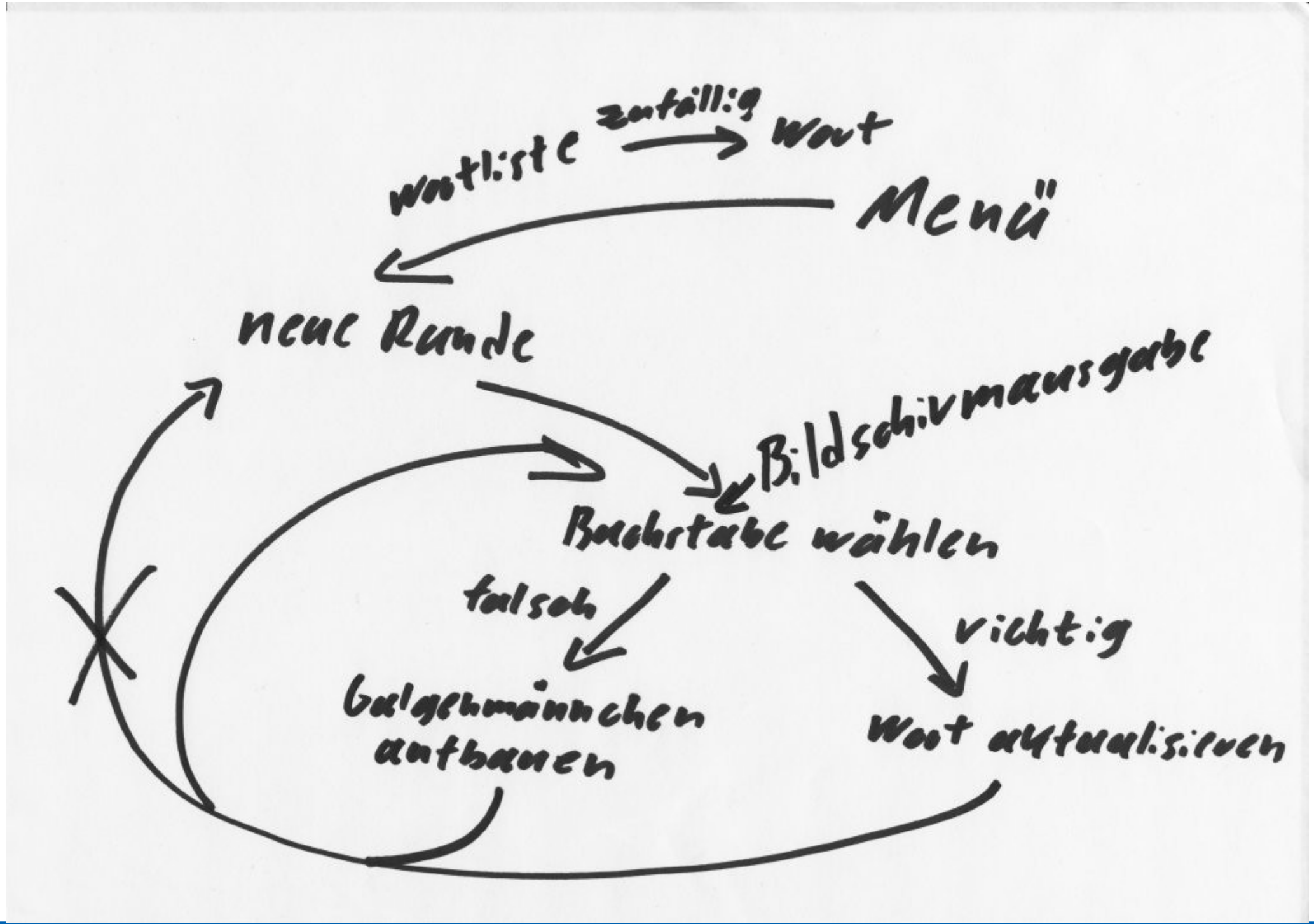

Menü
neue Runde
Buchstabe wählen

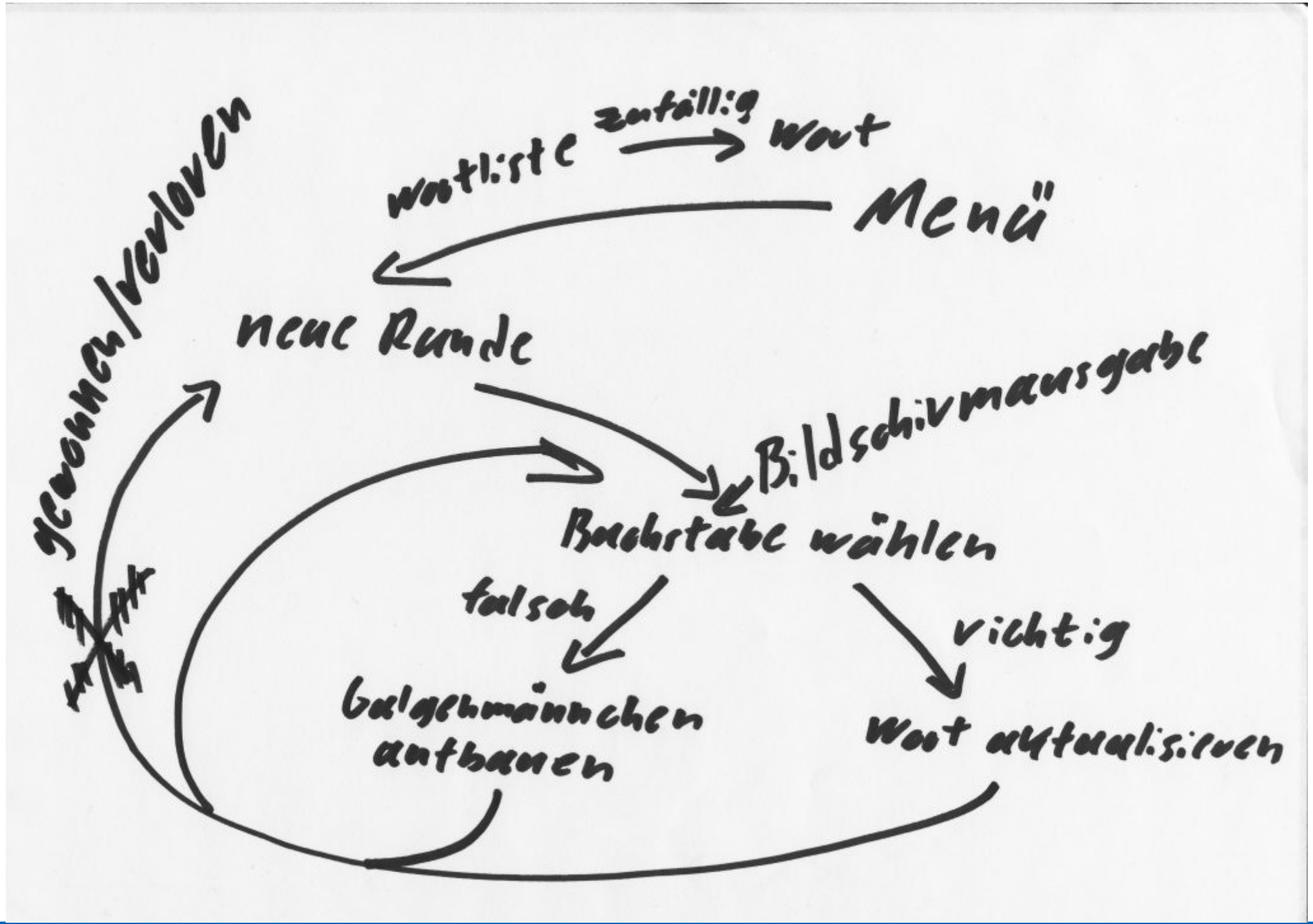

Menü
neue Runde
Buchstabe wählen
falsch
Galgenmännchen aufbauen


Menü
neue Runde
Buchstabe wählen
falsch
richtig
Galgenmännchen aufbauen
Wort aktualisieren


Menü
neue Runde
Buchstabe wählen
falsch
richtig
Galgenmännchen aufbauen
Wort aktualisieren

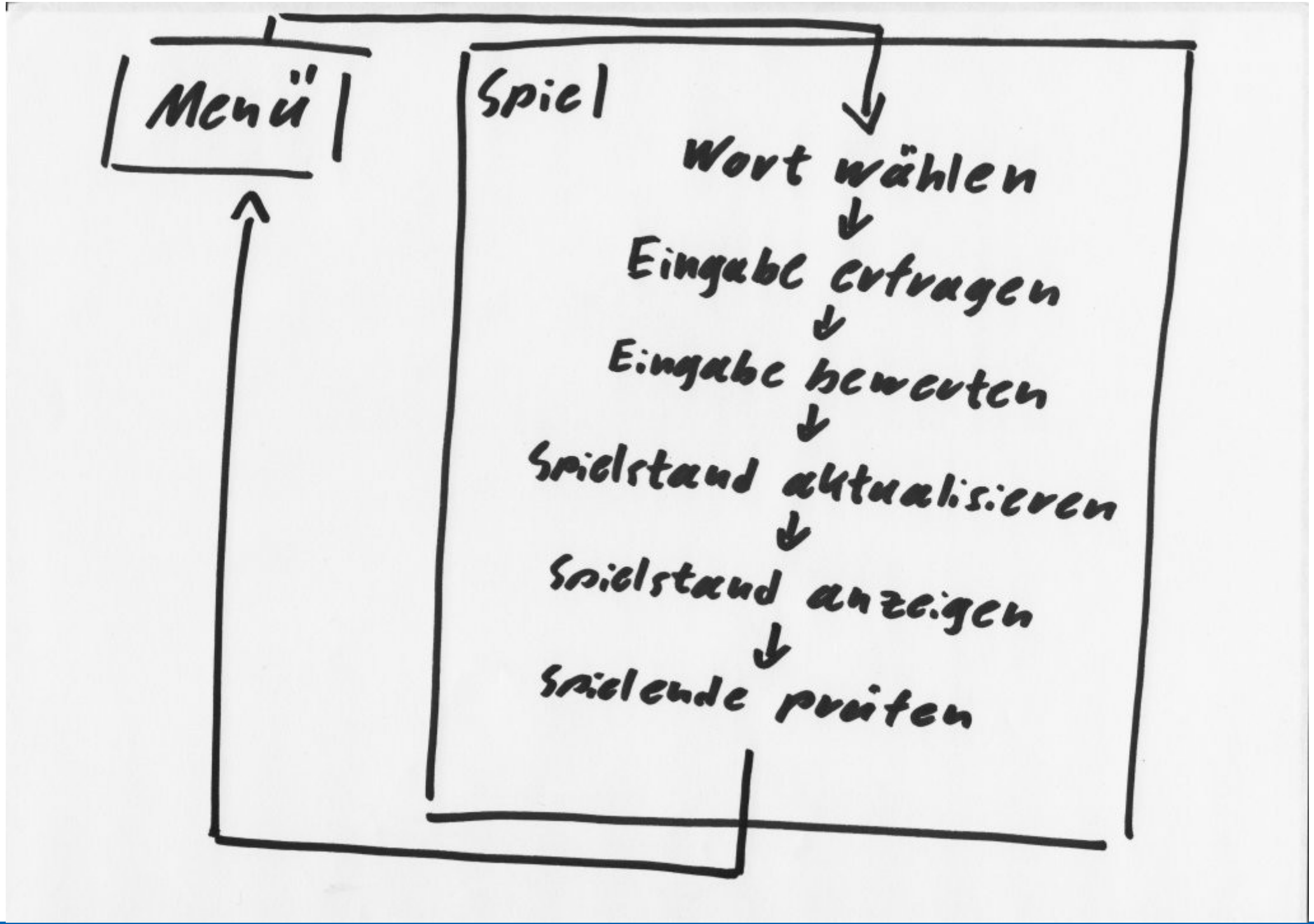

Wortliste
zufällig
Wort
Menü
neue Runde
Buchstabe wählen
falsch
richtig
Galgenmännchen aufbauen
Wort aktualisieren

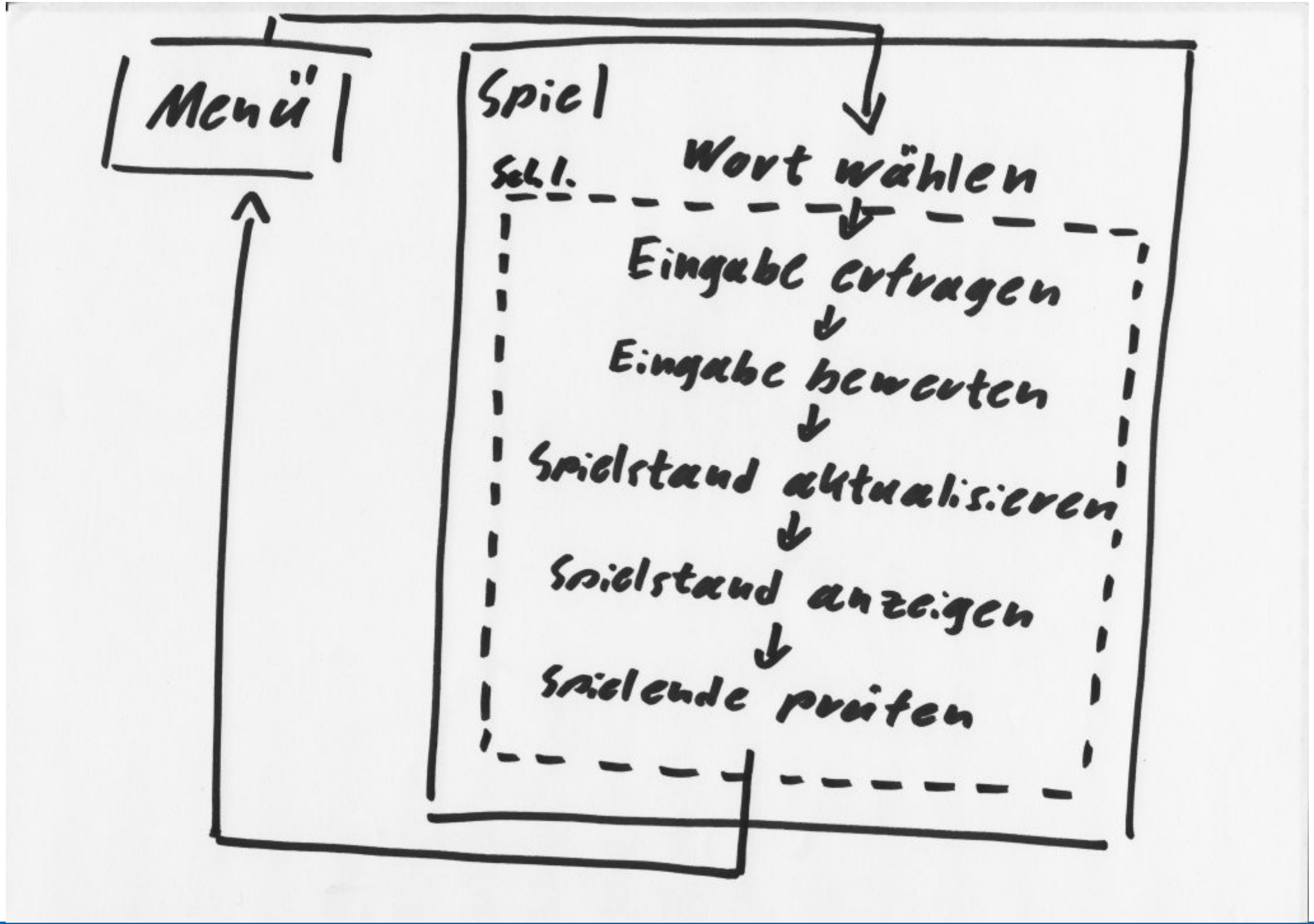

Wortliste
zufällig
Wort
Menü
neue Runde
Bildschirmausgabe
Buchstabe wählen
falsch
richtig
Galgenmännchen aufbauen
Wort aktualisieren


gewonnen/verloren
Wortliste
zufällig
Wort
Menü
neue Runde
Bildschirmausgabe
Buchstabe wählen
falsch
richtig
Galgenmännchen aufbauen
Wort aktualisieren

Schritt III: Skizze verfeinern

- **Skizzen mit Ablauf erzeugen**
  - Nützlich bei zu wenig Platz auf Diagramm
  - Erfahrungen können umgesetzt werden
  - Teilprobleme können definiert werden

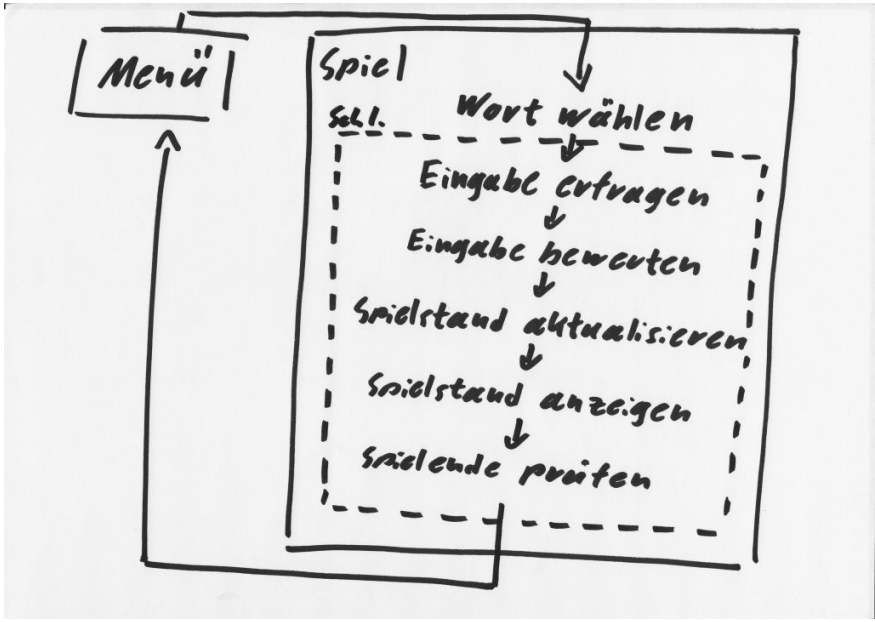

Menü
Spiel
Wort wählen
Eingabe erfragen
Eingabe bewerten
Spielstand aktualisieren
Spielstand anzeigen
Spielende prüfen


Menü
Spiel
Schl.
Wort wählen
Eingabe erfragen
Eingabe bewerten
Spielstand aktualisieren
Spielstand anzeigen
Spielende prüfen

Schritt IV: Teilen in Methoden

- **Skizze in Methode umwandeln**
  - Jedes Teilproblem ist eine Methode im Programm
  - sinnvolle Namen geben!
  - Was braucht die Methode?
    - Parameter?
    - Rückgabewert?

Schritt IV: Methoden



- *void menue ()*
- boolean isGameWanted()
- String chooseWord ()
- char askForChar ()
- boolean isCharInWord (char character, String word)
- void updateGame (char guessedCharacter)
- void displayGame ()
- boolean isGameOver()

Schritt V: Wiederholen

- Inhalt der Methoden sofort ersichtlich?
  - Ja, weiter mit nächster Methode
  - Nein, neue Skizze
- Bei offensichtlichen Methoden
  - Aufschreiben der Eingabe- und Rückgabeparameter

Agenda

- Wiederholung
- Aufgabenstellung
- Aufspalten der Aufgabe
- **Vom Diagramm zum Code**
- “richtiges” Programmieren
- Zusammenfassung

Code Einblick

- Viele Methoden, wie in Java umsetzen?
- Empfehlung: TopDown–Verfahren
- Beginn bei `void main()`
- Kommentare für noch nicht existierende Methoden einfügen

Code Beispiel

```
public static void main(String args[]) {
    // -Menueschleife-
          // neues Spiel ? wenn nein, dann Ende
          // Wort wählen
          // -Spielschleife-
                // Spielstand ausgeben
                // Buchstaben erfragen
                // Eingabe bewerten
                // Spielstand aktualisieren
                // Spielende prüfen
          // -ende-
    //-ende-
}
```

Schrittweiser Fortschritt

- Nun pro Kommentar eine neue Methode
- Variablen, Kommentare sowie Ausgaben einfügen
- Sobald neue Methode fertig: Testdurchlauf

Testen

- So früh wie möglich
- So oft wie möglich
- So gründlich wie möglich

Testen – Dumme Methoden

- Methoden mit Eingabe und Ausgabe Parametern
- Jedoch ohne Funktion
- Auch “Platzhalter” genannt

```
// TODO: Befüll mich!
public char askForChar(){
  // liefere festen Wert
  return 'a';
}
```

```
// TODO: Befüll mich!
public String
chooseWord() {
  // liefere festes Wort
  return “Javakurs”;
}
```

Testen – println debug

- Wdh
- Ausgabe von Werten, die wichtig sind
- Auskommentieren und nicht löschen

```
// starte ein Spiel
public double add
    (double a, double b){

  // Parameter okay?
  System.out.println(a +
      ", " + b);

  double result = a+b;

  // Ergebnis okay?
  System.out.println(
     Result );
  return result;
}
```

Agenda

- Wiederholung
- Aufgabenstellung
- Aufspalten der Aufgabe
- Vom Diagramm zum Code
- **“richtiges” Programmieren**
- Zusammenfassung

Fehler- und Modifikationstolleranz

Der Code

- Warum formatieren?
- Einrücken
  - pro neue Methode
  - pro Verzweigung (if/switch)
  - pro Schleife
- Sprechende Namen
- Zeilen nur so lang wie nötig, nicht wie möglich

Formatierung

- eure Handschrift
- Editoren helfen
- Wichtig für Zusammenarbeit
- Für Code der älter ist

```
import foo.bar;

// Klasse B die ...
class B extends A {

  // Konstruiere B
  public B() {
  }

  // Mache etwas
  public x( int i ) {
    // Inhalt
  }
}
```

Einrückung

- Erkennung von Teilbereichen
- Wichtig bei
  - Bedingungen
  - Schleifen
  - Klassen
  - Methoden

```
if( a == b ) {
    // dann zweig
} else {
    // sonst zweig
}

// -----

while( bAktiv ) {
   //
   // wiederhole ...
   //
}
```

Sprechende Namen

- Was steht in der Variablen?
  - Typbeschreibung `time, name, number ..`
- Was tut die Methode/das Objekt?

```
double timeLeft=0.4;

String studentName =
                    ”Hans”;

// -----

class HangmanMain{..}

boolean isGameWanted()
```

Schönheit kostet nix

- Wichtig: Code muss laufen!
- Optimierung meist vom Compiler besser als von euch
- Wenn Code viel zu langsam
  - Aufgabenstellung richtig gelesen?
  - Zuviele Nice-Too-Have-Features? (3D Galgen?)

Agenda

- Wiederholung
- Aufgabenstellung
- Aufspalten der Aufgabe
- Vom Diagramm zum Code
- “richtiges” Programmieren
- **Zusammenfassung**

Ausblick

- Ihr wisst jetzt
  - Wie zerlege ich eine Aufgabe in Teilprobleme
  - Wie Teile ich die Teilprobleme
  - Wie formuliere ich ein Teilproblem in Java
  - Was sollte ich beim Schreiben von Code beachten?
- Jetzt im Anschluss
  - Feedback Abgabe
  - Übungen im **Tel 106 / 206**

Fragen?