

# CHE4C, MATE!

ein Schachroboter von  
Alessio Mossudu und  
Ovidiu Tatar



# Einführung

- **C**omputer and **H**uman controlled **E**nvironment **f**or **C**hess (*CHE4C, MATE!*)
- spielt Schach
  - gegen Menschen
  - oder gegen sich selbst
- (Zeichnen)

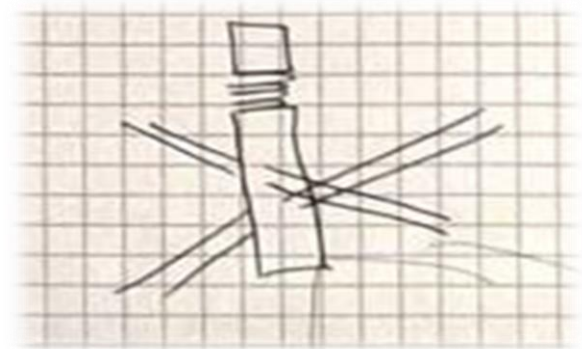
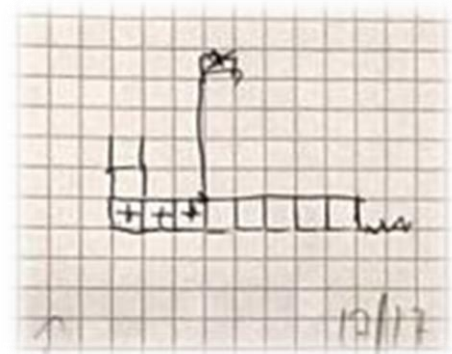


# Überblick

- Bewegung über Elektromagnet und Stepper
  - Schachfiguren
- Bilderkennung über Kamera und Farben
- Kommunikation über Knopf und LED
- Berechnung durch Schach-KI am PC

# Vertiefung: Bewegung

- Problem: Wie bewegen wir die Figuren?
  - Grundkonzept
  - Ebenen
  - Räder (Schienen)
  - Provisorisches Gestell
  - Anschluss der Motoren
- Bau des richtigen Gestells (Props an Luis)
- Testen und Verbessern (current stage)

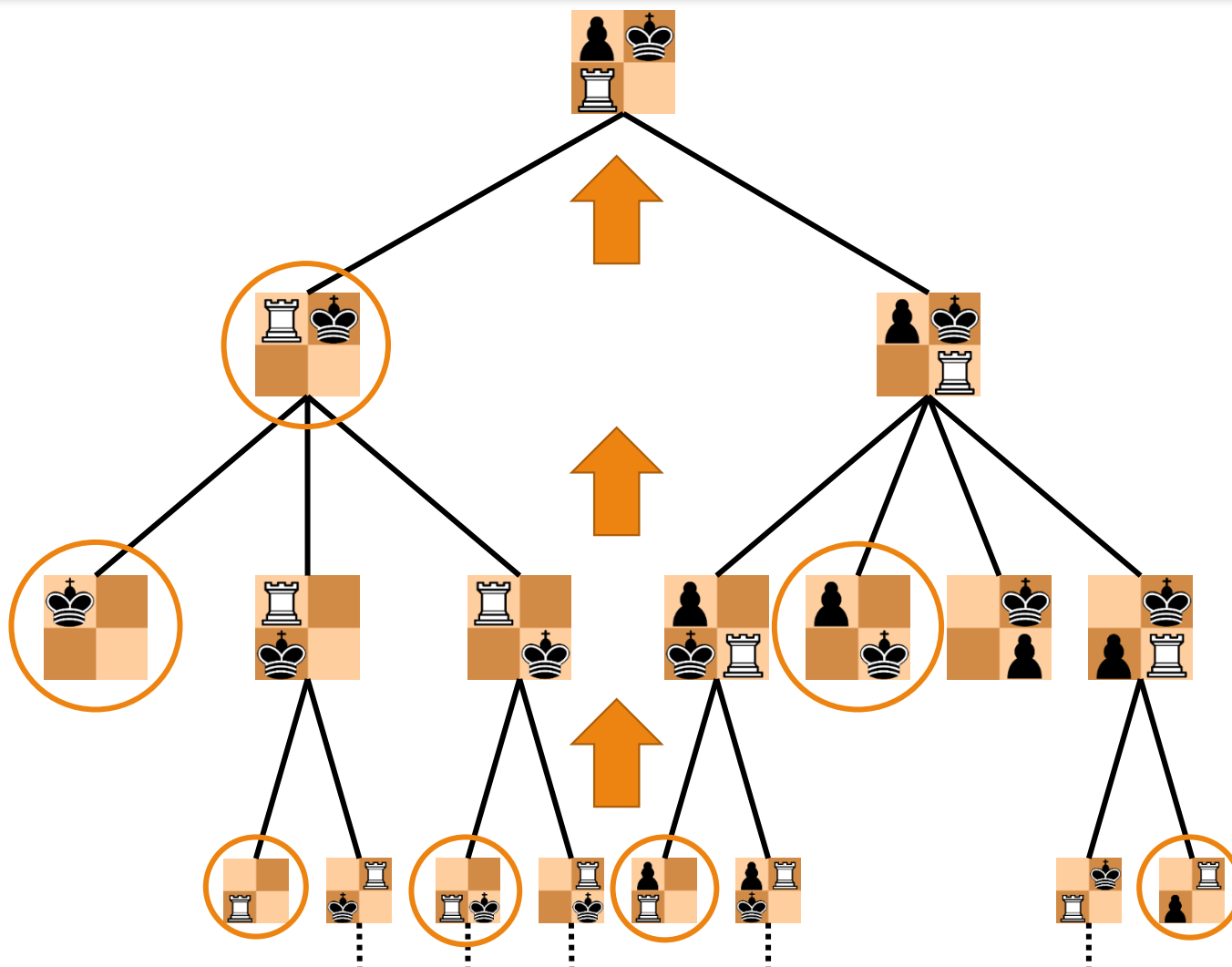




# Vertiefung: Berechnung

- Großteil des Quellcodes: mögliche Spielzüge generieren
  - Spielzug-Kommandos
  - Überprüfung des Spielbretts
- Großteil des Rechenaufwandes: Zugauswahl treffen (MiniMax)

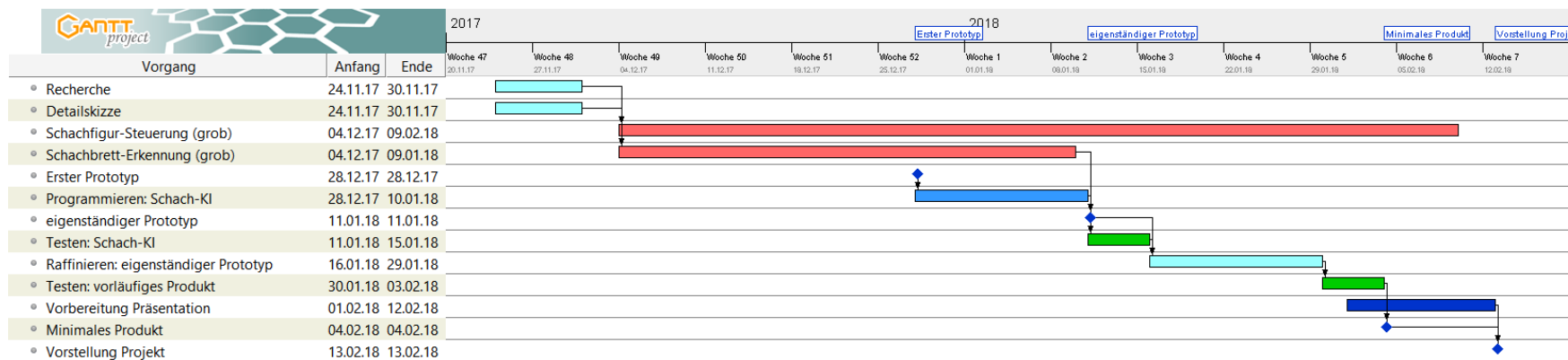
# Vertiefung: Berechnung





# Ergebnis und Fazit

- primäre Aufgaben (bald) erfüllt
  - ein paar Schrauben drehen
- Projekt-Zeitplan ist sehr optimistisch gewesen



# Ausblick: Coming Soon...

- Befestigung der Kamera
- Teilbereiche verbinden
  - Elektronik verbinden
  - Software verbinden
- (mehr) Schachfiguren
- Aufbau verschönern?!
- Steuerleiste mit Status-LED, Schalter



# Ausblick: mögliche Veränderungen

- Umstieg auf Raspberry PI
  - Vereinigung von Schach-KI, Steuerung und Bilderkennung
- Vereinigung der Stromquelle
- mehr Spielmodi
- Ansteuerung über Internet
  - reales Schachspiel über Website?



# Demonstration

---



# Fragen?

---

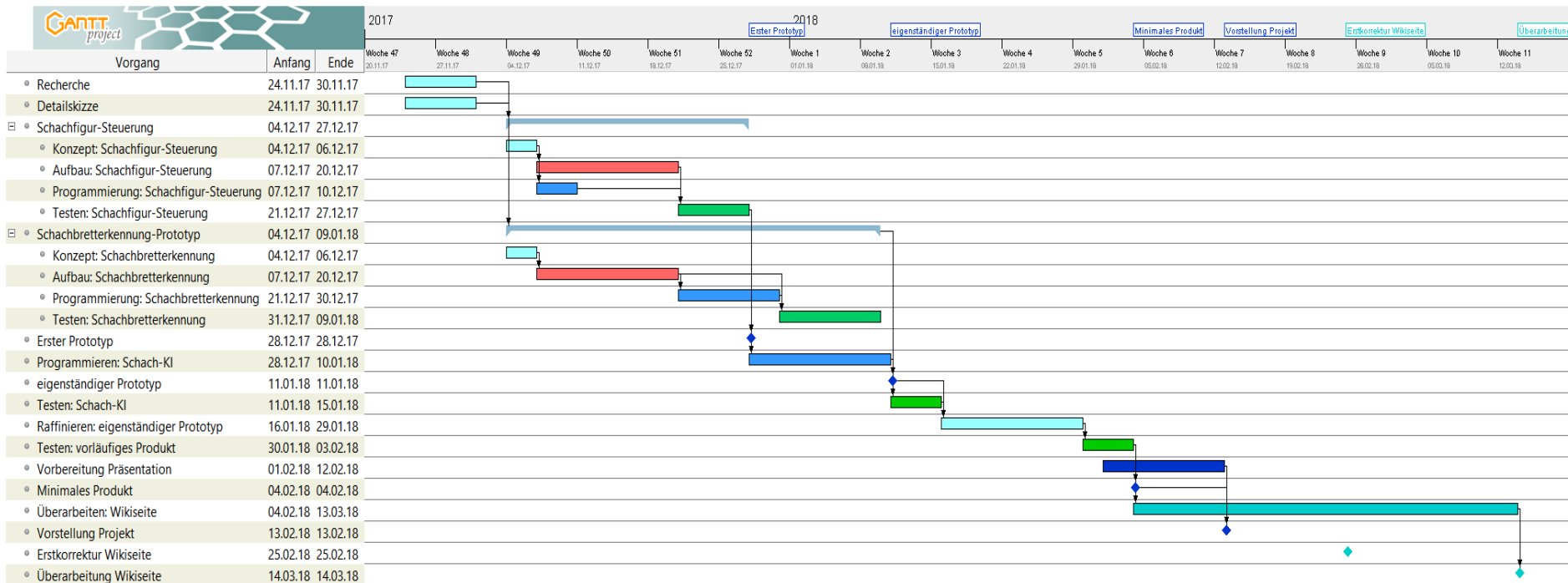
# Zusatzmaterial

---

- *Zusatzmaterial für mögliche Fragen, ggf. Zeitüberschuss*

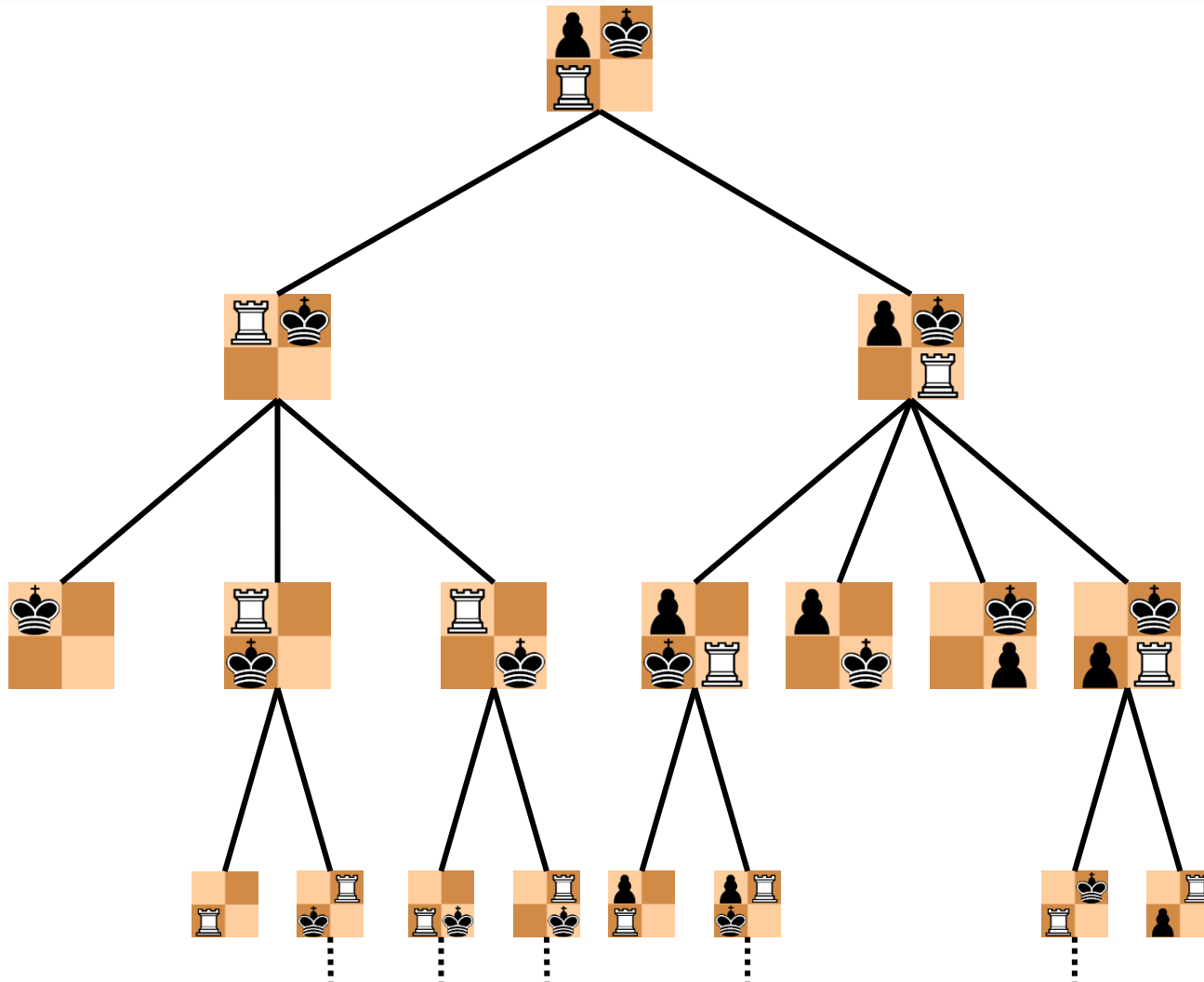


# Zusatzmaterial: kompletter Projektplan





# Zusatzmaterial: Spielzustands-Baum



# Zusatzmaterial: MiniMax-Prinzip

- Idee: Bewerte den Status des Spiels
  - positive Zahl, wenn am gewinnen/Vorteil
  - negative Zahl, wenn am verlieren/Nachteil
- maximiere eigenen Wert
- minimiere gegnerischen Wert

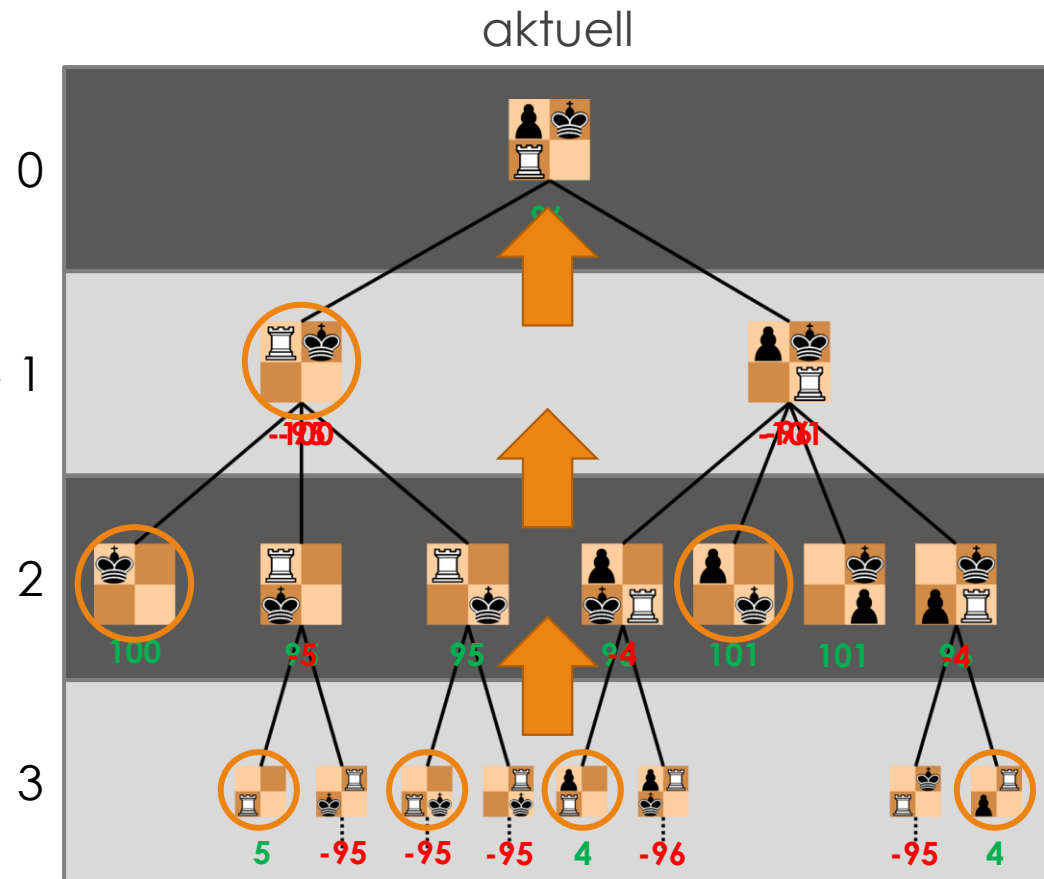
# Zusatzmaterial: NegaMax-Prinzip

- Idee: negieren des Maximums der Gegnerpunktzahl ist das gleiche wie minimieren
  - Bewertung aus Spielersicht
  - kompakterer Quellcode



# Zusatzmaterial: NegaMax-Prinzip

- jede Ebene maximiert für sich
- Bewertung:  
Figurenunterschied, etc.
- potentiell unendlich Züge 1
  - Suchtiefe
- bessere KI?
  - höhere Suchtiefe
  - bessere Bewertungsfunktion





# Zusatzmaterial:

## NegaMax-Pseudocode

```
miniMax(Spieler spieler, Spielzug spielzug, int tiefe) {  
    if (tiefe <= 0 || hatVerlohren(spieler))  
        return spielzug;  
  
    int maxWert = -Unendlich;  
    for (Spielzug zug : berechneMöglicheSpielzüge(spieler)) {  
        int wert = -miniMax(spieler.holeGegenspieler(), zug, tiefe - 1);  
        if (wert > maxWert) {  
            maxWert = wert;  
            spielzug = zug;  
        }  
    }  
    return spielzug;  
}
```