Mastermind mit dem Android SDK

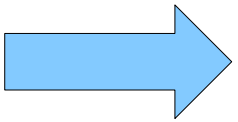
- Einführungen
  - Mastermind und Strategien (Stefan)
  - Eclipse und das ADT Plugin (Jan)
  - GUI-Programmierung (Dominik)

- Mastermind
  - Spielregeln
  - Variationen
  - Hinweise für die Programmierung
- Strategien
  - Lineare Suche
  - Min-Max
  - Genetischer Algorithmus

- Mastermind
  - Spielregeln
  - Variationen
  - Hinweise für die Programmierung
- Strategien
  - Lineare Suche
  - Min-Max
  - Genetischer Algorithmus


MASTERMIND

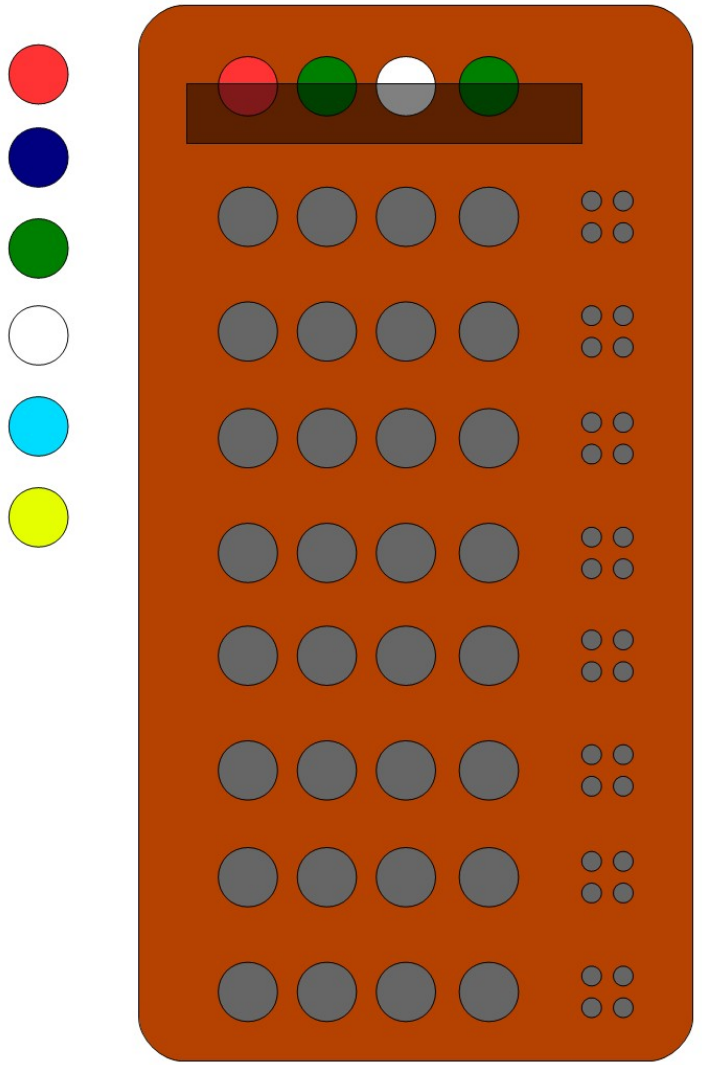
- Alice denkt sich einen verdeckten Farbcode aus.



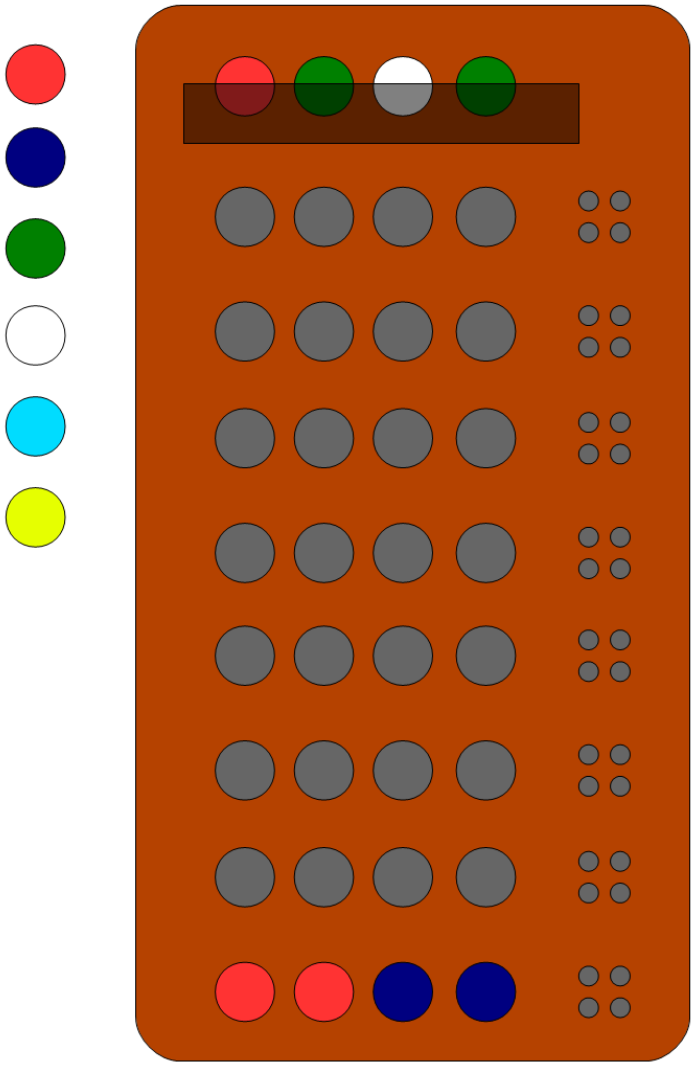
- Alice denkt sich einen verdeckten Farbcode aus.
- Bob muss den Code herausfinden.

- Alice denkt sich einen verdeckten Farbcode aus.
- Bob muss den Code herausfinden.
- Alice gibt Hinweise über die Korrektheit von Bobs Versuchen.

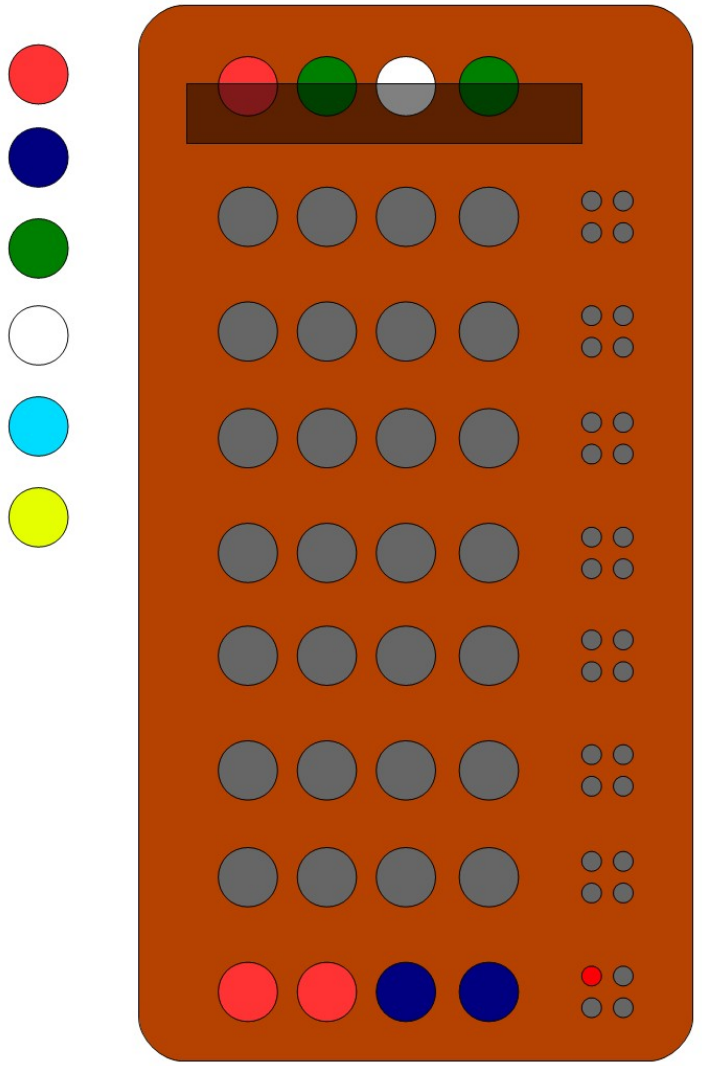
- Alice hat sich die Kombination Rot-Grün-Weiß-Grün ausgedacht.



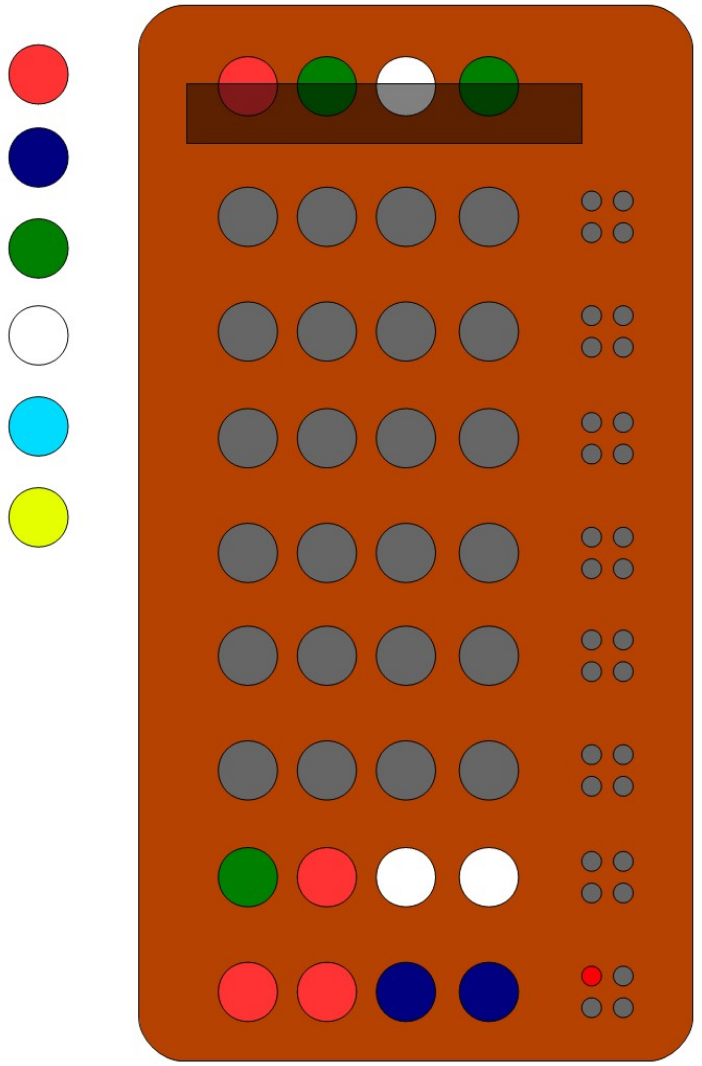
- Bob rät auf gut Glück



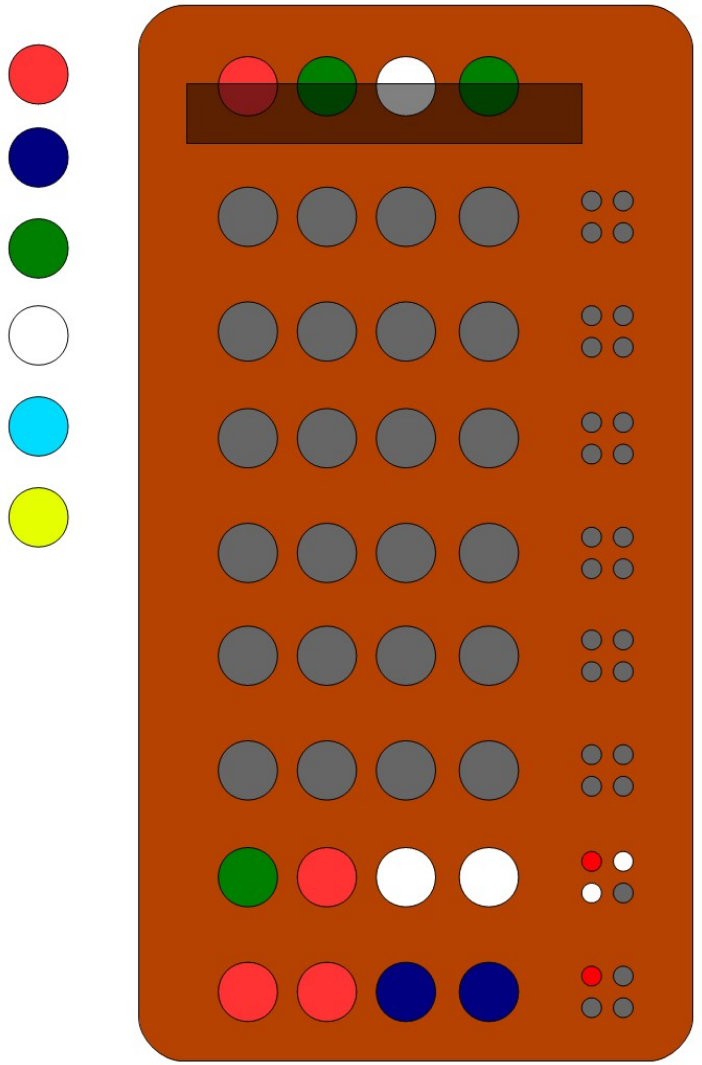
- Bobs Versuch enthält genau einen **exakten** Treffer.
- Exakt = Richtige Farbe am richtigen Platz.
- Alice markiert diese Information durch einen roten Stein.



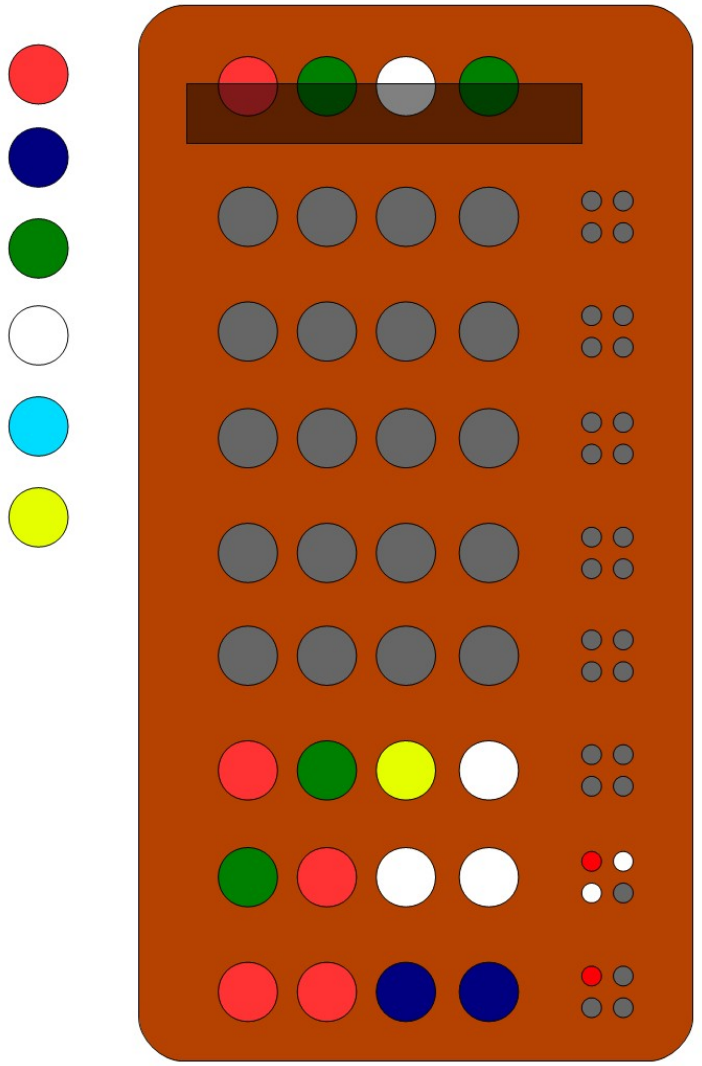
- Bobs nächster Versuch ist konsistent mit der bisherigen Information.
- Wie muss Alices Antwort lauten?



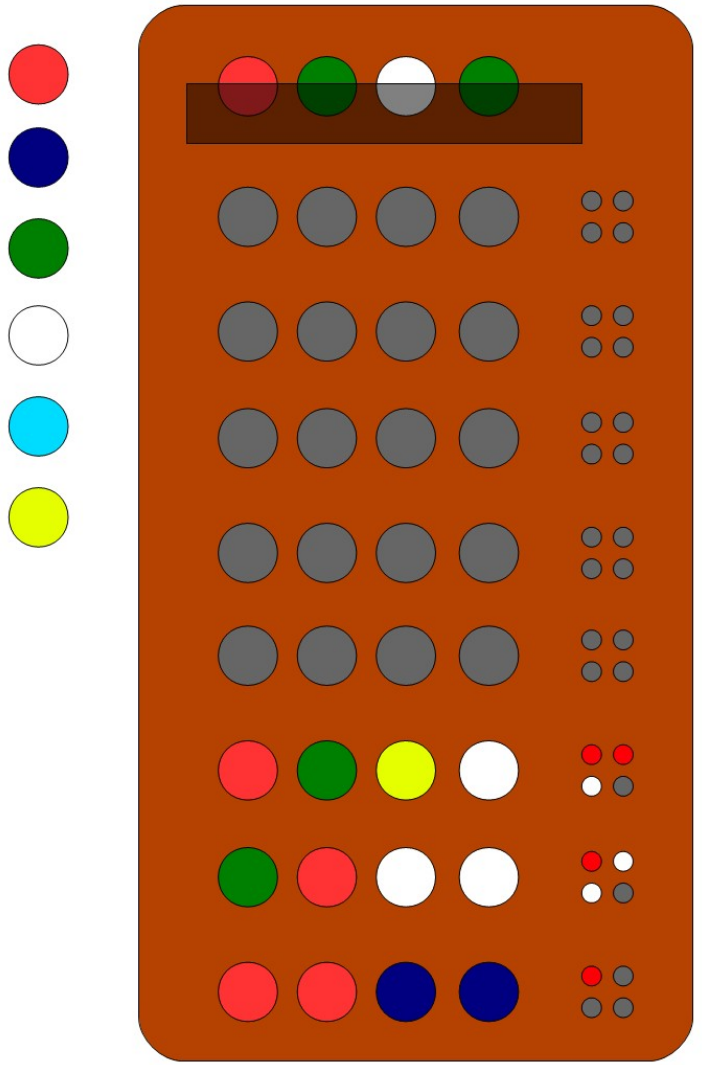
- Der Versuch enthält wieder einen exakten Treffer.
- Zwei Farben sind **korrekt**.
- Korrekt = Richtige Farbe aber am falschen Platz.
- Korrekte Treffer werden weiß markiert.



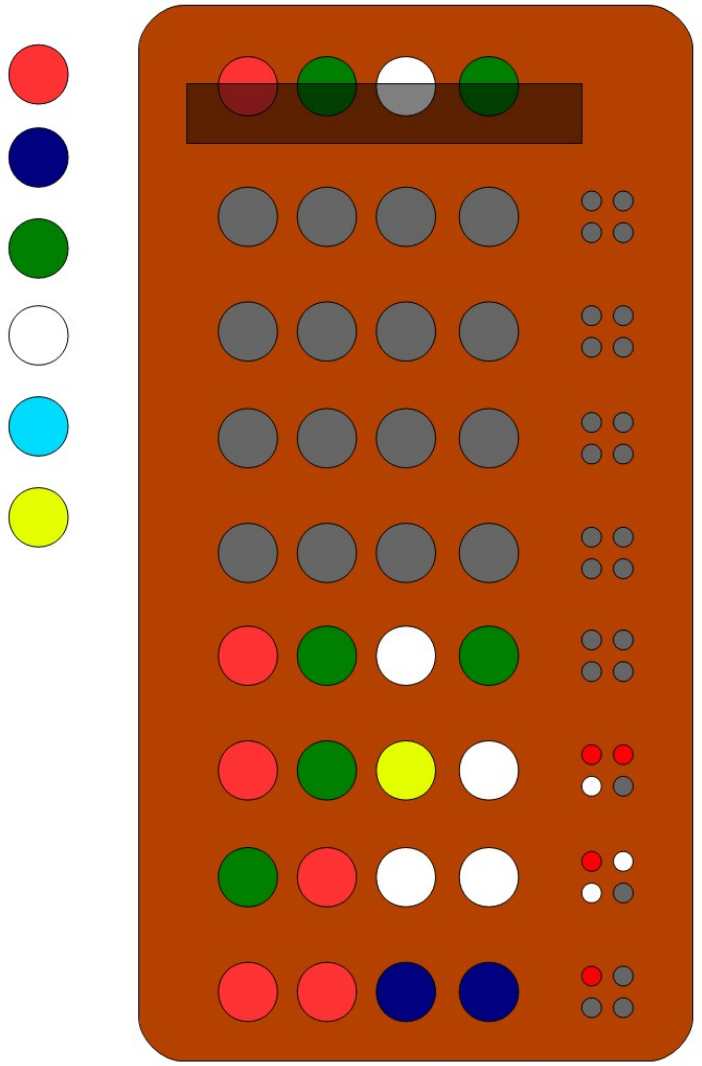
- Wie muss Alice Bobs nächsten Versuch markieren?



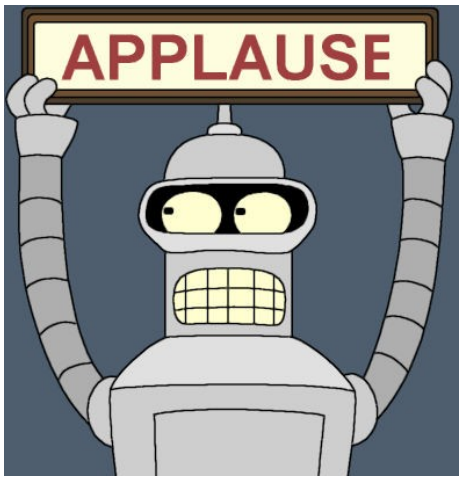
- Zwei exakte Treffer
- Ein korrekter Treffer

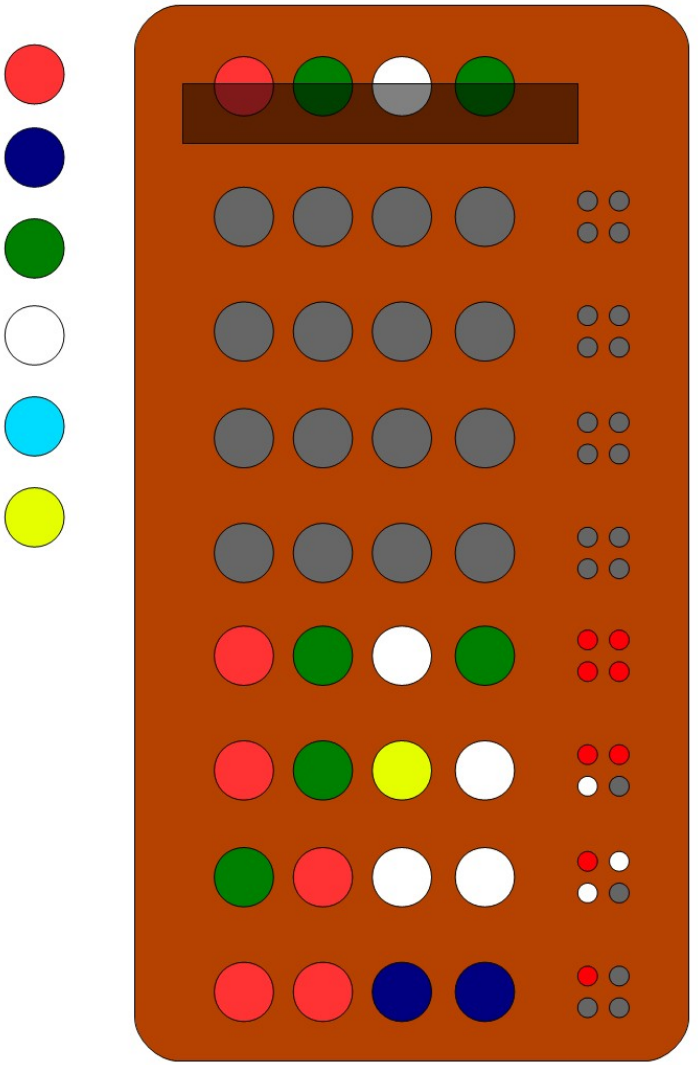


- Schließlich findet Bob den gesuchten Code.



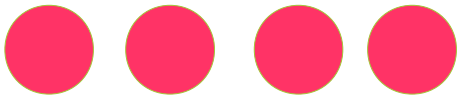
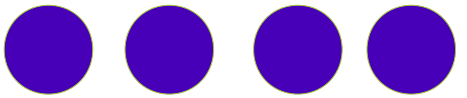
- Vier exakte Treffer





- Mastermind
  - Spielregeln
  - Variationen
  - Hinweise für die Programmierung
- Strategien
  - Lineare Suche
  - Min-Max
  - Genetischer Algorithmus

- Im obigen Beispiel (Standard-Variante):
  - 6 Farben, 4 Plätze, höchstens 8 Versuche
  - Versteckte Codes dürfen Farben mehrfach verwenden
- All diese Parameter können (fast beliebig) verändert werden.
  - Die Variation mit 8 Farben und 5 Steckplätzen ist z.B. unter dem Namen „SuperHirn professionell“ bekannt.
  - Weitere Varianten verwenden verschiedenfarbige Formen („Grand Mastermind“).

- Fragen:
  - Wie viele verschiedene Farbcodes gibt es
    - in obiger Variante (6 Farben, 4 Plätze, Farben mehrfach)?
    - wenn jede Farbe höchstens einmal vorkommen darf?
    - im allgemeinen Fall (c Farben, p Plätze, mit/ohne Wiederholung).
  - Wie viele „prinzipiell unterschiedliche“ Möglichkeiten hat Bob für seinen ersten Versuch?
    Z.B. sind  und  prinzipiell gleich.

- Mastermind
  - Spielregeln
  - Variationen
  - Hinweise für die Programmierung
- Strategien
  - Lineare Suche
  - Min-Max
  - Genetischer Algorithmus

- Lege dir zunächst den Prinzipiellen Spielablauf zurecht
  - Zufallscode wählen – Raten – Antworten – ...
  - Welche möglichen Spielausgänge gibt es?
- Überlege dir, wie sich die Anzahl exakter (roter) bzw. korrekter (weißer) Steine in Alices Antwort bestimmen lässt.
- Welche (c, p, mit/ohne WH)-Variante(n) willst du unterstützen?

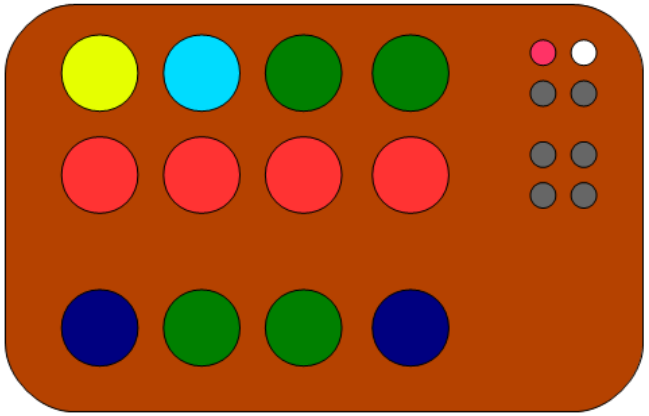
- Mastermind
  - Spielregeln
  - Variationen
  - Hinweise für die Programmierung
- Strategien
  - Lineare Suche
  - Min-Max
  - Genetischer Algorithmus

- Mögliches Feature eurer Implementierung: Tipp-Button
- Gesucht ist eine Vorschrift, die für jede Spielsituation einen sinnvollen Tipp findet
- Eine solche Vorschrift nennen wir „Strategie“

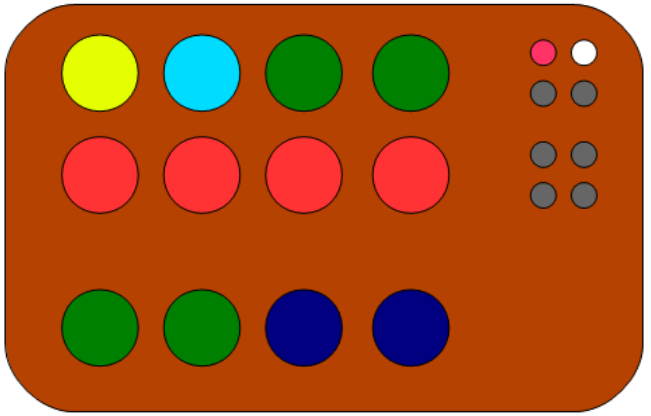
- Mögliche Fragen
  - Wann ist eine Strategie gut?
    (Worst-Case, Durchschnitt)
  - Wie gut läuft die Strategie auf dem Handy?
    (Laufzeit, Speicher)
  - Wie verhält sich die Strategie für verschiedene
    (c, p, mit/ohne WH)-Varianten?

- Terminologie (analog zu `strategies.Utils`)
  - Im folgenden: 6^4-Variante.
  - Gespielte Konfigurationen: Bisherige Versuch-Antwort-Kombinationen
  - Konsistenter Code: Farb-Code, der prinzipiell der versteckte Code sein könnte.

Gespielte Konfigurationen



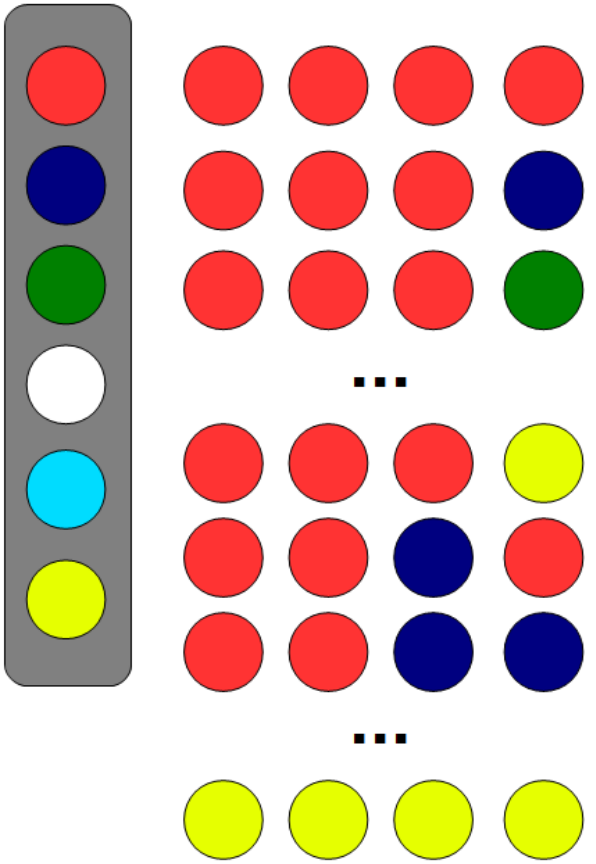
Konsistent



Nicht konsistent

- Mastermind
  - Spielregeln
  - Variationen
  - Hinweise für die Programmierung
- Strategien
  - Lineare Suche
  - Min-Max
  - Genetischer Algorithmus

- Ordne alle möglichen Codes
- Spiele den nächsten Code, der mit allen bisherigen Konfigurationen konsistent ist
- Wiederhole Schritt 2 bis die Lösung gefunden ist



- Vorteile
  - Einfache Implementierung
- Nachteile
  - Ineffizient für größere Anzahl von Farben oder Steckplätzen
  - Schlechtes Durchschnitts- und Worst-Case-Verhalten

- Mastermind
  - Spielregeln
  - Variationen
  - Hinweise für die Programmierung
- Strategien
  - Lineare Suche
  - Min-Max
  - Genetische Algorithmen

- Finde unter allen noch nicht gespielten Codes denjenigen, der
  - für alle möglichen Antworten
    - die maximale Anzahl der übrig bleibenden konsistenten Codes minimiert.

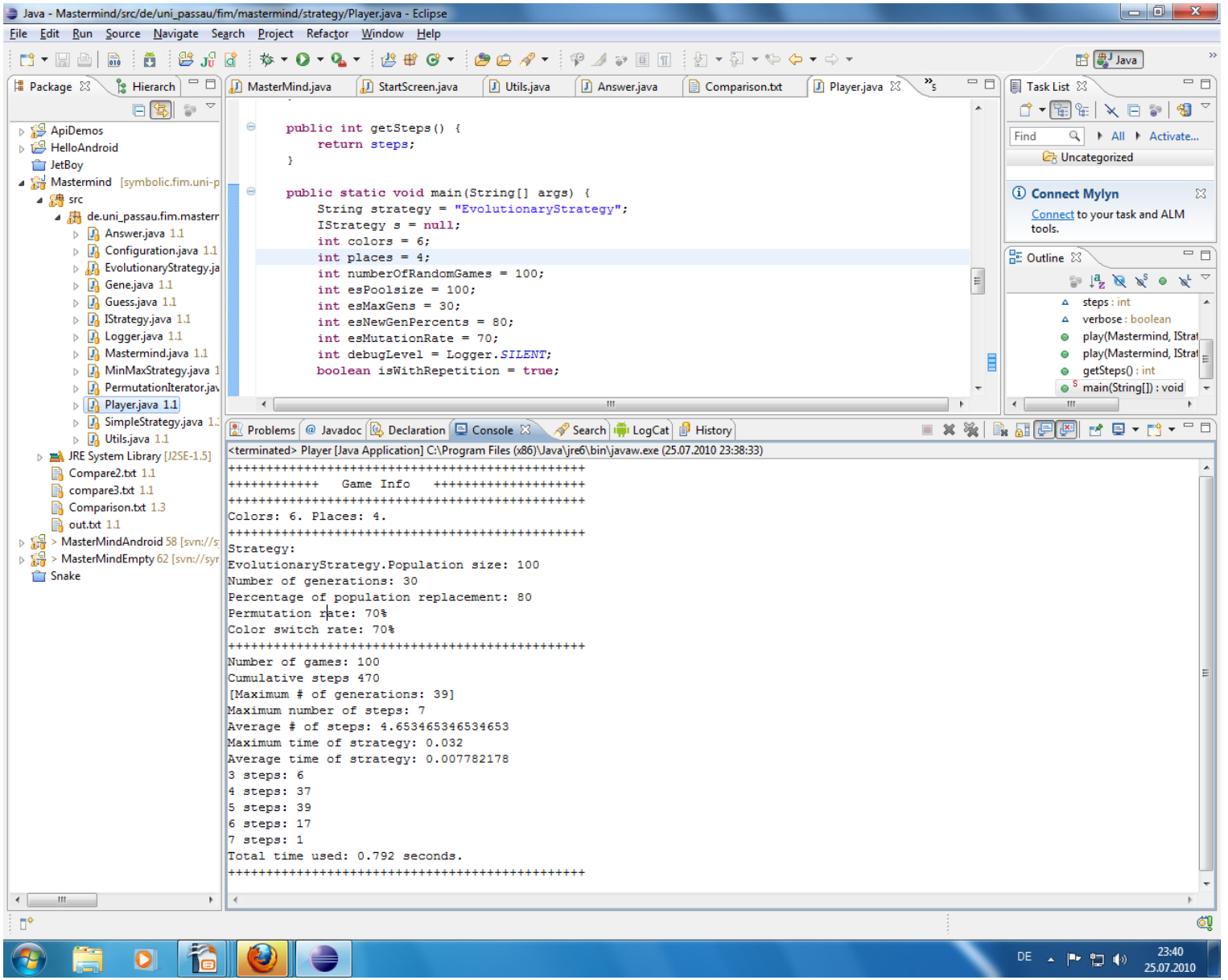
- Vorteil
  - Worst-Case-optimal: Maximal 5 Züge für 6^4-Mastermind
  - Relativ gutes durchschnittliches Verhalten: 4.478 für 6^4-Mastermind (Optimum: 4.340 bei höchstens 6 Zügen)
- Nachteile
  - Ineffizient für größere Anzahl von Farben oder Steckplätze

- Mastermind
  - Spielregeln
  - Variationen
  - Hinweise für die Programmierung
- Strategien
  - Lineare Suche
  - Min-Max
  - Genetische Algorithmen

- Generiere zufällig einen **Pool** von N Codes.
- Bestimme die **Fitness** jedes Codes mittels einer geeigneten Fitness-Funktion.
- Wende zufällige **Selektion**, **Mutation** und **Crossover** auf den Pool an, um eine neue Generation von Codes zu erhalten
- Diese ersetzen die alte Generation ganz oder teilweise.
- Wiederhole dies, bis ein konsistenter Code gefunden wurde.

- Vorteile
  - Ordentliche durchschnittliche Performance auch bei größerer Anzahl von Farben oder Steckplätzen
  - Relativ gutes durchschnittliches Verhalten
- Nachteile
  - Schlechtes Worst-Case-Verhalten

- Die vorgestellten Strategien sind lediglich eine Auswahl, eurer Kreativität sind keine Grenzen gesetzt (Hill-climbing, Imitieren menschlichen Verhaltens,...)
- Eure Strategie sollte das Interface `strategies.IStrategy` implementieren.
- Zur Auswertung Eurer Strategien wird ein entsprechendes Tool bereitgestellt.


Java - Mastermind/src/de/uni_passau/fim/mastermind/strategy/Player.java - Eclipse
File Edit Run Source Navigate Search Project Refactor Window Help
Package Explorer
Hierarchy
MasterMind.java
StartScreen.java
Utils.java
Answer.java
Comparison.txt
Player.java
public int getSteps() {
return steps;
}
public static void main(String[] args) {
String strategy = "EvolutionaryStrategy";
IStrategy s = null;
int colors = 6;
int places = 4;
int numberOfRandomGames = 100;
int esPoolsize = 100;
int esMaxGens = 30;
int esNewGenPercents = 80;
int esMutationRate = 70;
int debugLevel = Logger.SILENT;
boolean isWithRepetition = true;
Task List
Connect Mylyn
Connect to your task and ALM tools.
Outline
steps : int
verbose : boolean
play(Mastermind, IStrat
play(Mastermind, IStrat
getSteps() : int
main(String[]) : void
Problems
Javadoc
Declaration
Console
Search
LogCat
History
<terminated> Player [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (25.07.2010 23:38:33)
++
++++++++++++ Game Info ++++++++++++++++++++
++
Colors: 6. Places: 4.
++
Strategy:
EvolutionaryStrategy.Population size: 100
Number of generations: 30
Percentage of population replacement: 80
Permutation rate: 70%
Color switch rate: 70%
++
Number of games: 100
Cumulative steps 470
[Maximum # of generations: 39]
Maximum number of steps: 7
Average # of steps: 4.653465346534653
Maximum time of strategy: 0.032
Average time of strategy: 0.007782178
3 steps: 6
4 steps: 37
5 steps: 39
6 steps: 17
7 steps: 1
Total time used: 0.792 seconds.
++
DE
23:40
25.07.2010

- Alle Folien und mehr findet ihr unter

  http://sommercamp.fim.uni-passau.de/Projects/android-app/