# KI für Messe

Datenstrukturen:

* + Map kopiert, um Berechnungen durchzuführen
  + Tiles der Hand in HashMap (einmal sortiert nach Farben, einmal nach Symbols)
  + Array, in der alle möglichen Koordinaten gespeichert sind

1. freie Felder ermitteln bzw. aktualisieren
   * nach einem gemachten Spielzug den gelegten Tile ermitteln und Koordinaten x-1, x+1, y-1, y+1 prüfen ob ein Tile anliegt
   * falls nicht, dann wird die Koordinate zu dem Array der freien Felder hinzugefügt
   * sonst passiert nichts
2. Tile ermitteln, welches gelegt werden kann
   * Gehe alle Felder der möglichen zu legenden Felder durch
   * Falls ein Tile gelegt werden kann, dann lege
   * Sonst passiert nichts

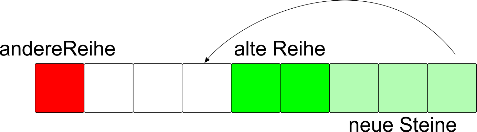
# KI für Turnier

Datenstrukturen:

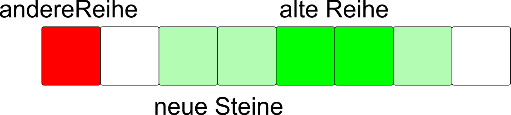
* + Map kopiert, um Berechnungen durchzuführen, dazu (in jedem Feld, wo ein Stein liegt Speichern wir Array [links, rechts, oben, unten]) (0 – hat Nachbar, 1 – ist frei, man kann in diesem Richtung etwas hinlegen, block – ist gesperrt)
  + Tiles der Hand in HashMap (einmal sortiert nach Farben, einmal nach Symbols)
  + Hilfs\_Array/ *HashMap*, in der alle möglichen Koordinaten gespeichert sind (aber nur diese, wo Array[links] == 1 oder block)
  + Ergebnis\_Array/ *HashMap*

Algo

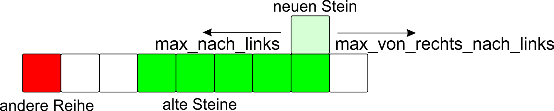
1. versucht zuerst alle Steine maximal nach rechts hinlegen und dann

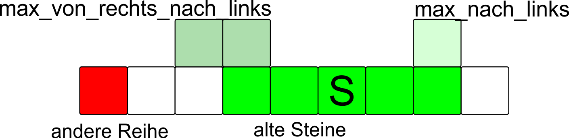
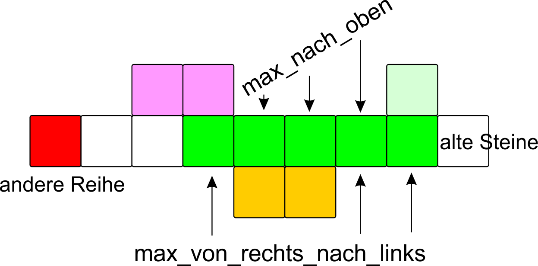
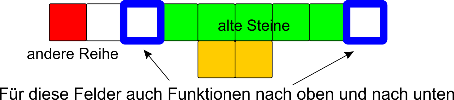


1. maximal langen Move nach links machen (aber nicht länger als die Anzahl der Farben/Symbolen im Spiel (6 – 12) ) und die Vorne von Move soll sich mit andere Reihe nicht binden (wenn bindet sich, Punkt 3 übersprungen). Für dem letzten Stein der Reihe, mache max\_von\_rechts\_nach\_links
2. Für dem ersten Stein der Reihe, mache Max\_nach\_links wenn Array[links] == 1 bis (sehe Bild)



1. Für letzten Stein, wenn Array[oben] == 1, ein Stein hinlegen



1. Für diesem Stein machen die Funktionen: max\_von\_rechts\_nach\_links und Max\_nach\_links
2. Falls sie keine gemeinsame Teil hätten, wiederhole 5. für Stein S
3. Analog unten
4. Punkt 3 – 6 wiederhole für Array[unten] ==1
5. Falls gibt es die Steine, die Array[oben] == 1 oder Array[unten]==1 haben, man soll Funktionen move\_von\_unten\_nach\_oben, und Move\_nach\_oben aufrufen.
6. Die vertikale Funktionen sollen auch für Felder wie auf dem Bild aufgerufen werden
7. Ganze Reihe wurde betrachten, es soll jetzt den nächsten Field aus Hilfs\_Array gelesen werden
8. Array Ergebnis besteht aus besten Lösungen (mit ´der größten Anzahl der Punkte), die nach Prioritäten sortiert sind.
9. Nach dem Move KI und nach dem Move der Gegner muss die Hilfs\_Array aktualisiert werden
10. Es gibt zwei Fälle:
    1. Move horizontal
11. Für alle Steine außer ersten und letzten setzt man Array[links]=Array[rechts]=0
12. Für ersten Stein: aktualisierung\_ersten\_horizontal
13. Für alle steine aus Move, die frischgelegt wurden ist die Funktion aktualisierung\_horizontal aufgerufen
    1. Move vertikal

Funktionen:

1. Laufe durch Hilfs\_Array. Nehme ein Element z.B Field(x,y):
2. Funktionen:
   1. max\_von\_rechts\_nach\_links
3. Betrachte Field(x, y), Field (x, y+1) um gemeinsame Teil zu finden (Farbe, Symbol)
4. suche den längsten Move nach rechts von Ende der Reihe (nur wenn im letzten Stein der Reihe Array[rechts] ==1), also mache Move\_horisontal\_nach\_ rechts
5. Wenn nicht alle Steine genutzt waren und in erstem Stein von der Reihe Array[links] ==1, versuche ich sie nach links hinzulegen.
   1. Max\_nach\_links
6. prüfe, ob Field(x, y-2) leer ist;
7. wenn ja, dann aus Tabelle Hand lege ich ein Stein mit passende Farbe/Symbol und Validierung
8. Wenn der Stein passt, lege ihn hin und gehe eine Stelle nach links und wiederhole alles ab 1), wenn nein (nehme nächsten Stein aus Hand / versuche ihn nach rechts zu hinlegen)

Wichtig:  
Wenn ich einen maximalen Move gefunden habe, Rechne die Punkte und gehe zum Speichern, suche nicht nach andere.

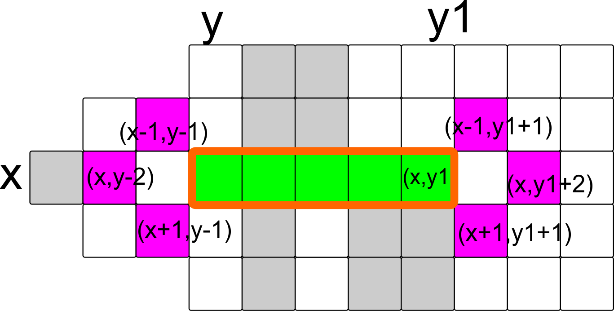
* 1. move\_von\_unten\_nach\_oben

Analog wie von rechst\_nach\_links, nur die Richtung unten – oben, also erst max nach unten, was nicht passt, nach oben.

* 1. Move\_nach\_oben

Analog wie max\_nach\_links

* 1. aktualisierung\_erstenStein\_horizontal



* Grüne Farbe – neu Move
* Graue Farbe – alte Steine
* Rosa Farbe – die Felder, die auch aktualisiert sein müssen (wenn sie nicht leer sind)

1. Erst prüfen, ob lenght.Move > die Anzahl der Farben im Spiel, wenn ja setze Blockade für betrachtetenn Felder.
2. Betrachte nur nicht leere Felder: (x-1, y-1), (x, y-2), (x+1, y-1)
3. Nehme Field(x-1, y-1).Array[unten], Field[x, y-2].Array[rechts], Field[x+1, y-1] und prüfe, ob block ist,

* wenn ja setze für Field(x,y).Array[links] = block,
* wenn nein, vergleiche Steine aus dieser Felder und suche gemeinsame Teil, wenn es solche gibt, setze Field(x,y).Array[links] auf 1, sonst setze auf block und   
  Field(x-1, y-1).Array[unten] = Field[x, y-2].Array[rechts] = Field[x+1, y-1] = block (diese, die nicht leer sind!!!!!)

1. Ähnlich für Ende der Move
2. freie Felder ermitteln bzw. aktualisieren
   * nach einem gemachten Spielzug den gelegten Tile ermitteln und Koordinaten x-1, x+1, y-1, y+1 prüfen ob ein Tile anliegt
   * falls nicht, dann wird die Koordinate zu dem Array der freien Felder hinzugefügt
   * sonst passiert nichts
3. Tile ermitteln, welches gelegt werden kann
   * Gehe alle Felder der möglichen zu legenden Felder durch
   * Falls ein Tile gelegt werden kann, dann lege
   * Sonst passiert nichts

Speichern:

1. Wenn ein Probe-Move hat:
   1. so viele Punkte, wie frühere Ergebnisse im Ergebnis\_Array, speichere ihn auch,
   2. mehr Punkte, lösche ich alle frühere und speichere neuen.