**Dokumentation des Prolog-Codes für Spring Solitaire**

**Hilfsprädikate**

* ersetze(Liste, Index, NeuesElement, NeueListe): Dies ist ein grundlegendes Hilfsprädikat, das dazu dient, ein Element in einer Liste an einer spezifischen **Indexposition** auszutauschen. Wenn Sie beispielsweise die erste Position (Index 0) einer Liste ändern möchten, ersetzt es das dortige Element durch NeuesElement und gibt die modifizierte Liste als NeueListe zurück. Es ist entscheidend für das Aktualisieren des Spielfeldzustands nach einem Zug.

**Spielfeld-Layout**

Die folgenden Prädikate legen die geometrische Struktur des Solitaire-Spielfelds fest. Da das Feld keine einfache Matrix ist, werden die gültigen "Linien" von Feldern explizit definiert.

* spalten(SpaltenListe): Definiert die **vertikalen Verbindungslinien** auf dem Spielfeld. SpaltenListe ist eine Liste von Listen, wobei jede innere Liste eine Reihe von Feld-Indizes darstellt, die vertikal miteinander verbunden sind. Dies ist wichtig, um vertikale Sprünge zu ermöglichen.
* zeilen(ZeilenListe): Definiert die **horizontalen Verbindungslinien** auf dem Spielfeld. Ähnlich wie bei spalten/1 ist ZeilenListe eine Liste von Listen, die die Indizes der Felder enthält, die horizontal miteinander verbunden sind. Dies ist notwendig, um horizontale Sprünge zu validieren.

**Zugdefinition**

Die zug/3-Prädikate sind das Herzstück der Spielregeln, da sie festlegen, welche Sprünge auf dem Spielfeld erlaubt sind. Ein Zug erfordert immer drei aufeinanderfolgende Felder entlang einer geraden Linie, wobei der mittlere Stein übersprungen wird.

* zug(Von, Über, Nach): Dieses Prädikat definiert einen **gültigen Zug** im Spiel. Ein Zug ist gültig, wenn die Felder Von, Über und Nach direkt aufeinanderfolgend (entweder in dieser Reihenfolge oder umgekehrt) in einer der zuvor definierten **Spalten** oder **Zeilen** liegen.
  + Von: Die Startposition des Steins, der springt.
  + Über: Die Position des Steins, der übersprungen und entfernt wird.
  + Nach: Die leere Zielposition, auf der der springende Stein landet.

**Start- und Zielzustände**

* start\_feld(Feld): Erzeugt das **Anfangsspielfeld** für eine typische Solitaire-Partie. Es initialisiert eine Liste von 33 Feldern (was der Gesamtanzahl der Felder auf dem Solitaire-Brett entspricht). Standardmäßig werden alle Felder mit einem Stein (x) belegt, außer der Mittelposition (Index 16), die als leer (o) definiert wird.
* ziel(Feld): Definiert den **Zielzustand** des Spiels. Ein Spielfeld entspricht dem Ziel, wenn es ebenfalls 33 Felder hat, aber **nur noch das Feld mit dem Index 16 einen Stein (x) enthält**, während alle anderen 32 Felder leer (o) sind.

**Züge anwenden**

* wende\_zug\_an(AktuellesFeld, Zug, NeuesFeld): Dieses Prädikat ist dafür zuständig, einen **gültigen Zug auf dem Spielfeld auszuführen** und den daraus resultierenden neuen Spielfeldzustand zu berechnen.
  + Es überprüft zunächst, ob die Voraussetzungen für den Zug erfüllt sind: Auf den Feldern Von und Über muss sich ein Stein (x) befinden, und das Feld Nach muss leer (o) sein.
  + Wenn die Voraussetzungen erfüllt sind, werden die Steine auf Von und Über entfernt (ihre Positionen werden auf o gesetzt), und auf das Feld Nach wird ein Stein gesetzt (es wird auf x gesetzt). Das Ergebnis ist das NeuesFeld.

**Spielfeld anzeigen**

Diese Prädikate sind für die Ausgabe des Spielfelds in der Konsole zuständig, um den Fortschritt des Spiels sichtbar zu machen.

* zeige\_feld(Feld): Dieses Prädikat nimmt einen Spielfeldzustand entgegen und **zeigt ihn in einer lesbaren, formatierten Weise an**. Es nutzt die zuvor definierten zeilen/1 und eine spezielle zeilen\_padding/2-Regel, um das Spielfeld korrekt als kreuzförmiges Brett darzustellen.
* nummerierte\_zeilen/3: Ein Hilfsprädikat, das Zeilen mit ihrer Indexnummer paart, was für die korrekte Padding-Berechnung nützlich ist.
* zeilen\_padding(ZeilenNummer, Padding): Legt fest, wie viele Leerzeichen (Padding) vor jeder Zeile ausgegeben werden müssen, um die optische Form des Solitaire-Bretts nachzubilden (die kurzen Zeilen an den Enden des Kreuzes benötigen mehr Einzug).
* index\_zu\_symbol(Feld, Index, Symbol): Wandelt den internen Zustand eines Feldes (z.B. x für Stein oder o für leer) in ein **anzeigbares Symbol** um (z.B. 'x ' für einen Stein und '. ' für ein leeres Feld), um die Ausgabe übersichtlicher zu gestalten.

**Löser und Zustandsinitialisierung**

Diese Prädikate ermöglichen es, Lösungen für das Solitaire-Spiel zu finden und anzuzeigen.

* zeige\_zuege(Feld, ZugListe): Dieses Prädikat wird verwendet, um eine gefundene **Sequenz von Zügen Schritt für Schritt zu visualisieren**. Es gibt jeden Zug aus, wendet ihn auf das Feld an und zeigt dann den resultierenden neuen Spielfeldzustand an, bis alle Züge in der Liste abgearbeitet sind.
* feld\_mit\_x(IndicesMitO, Feld): Ein nützliches Prädikat, um ein Spielfeld zu **initialisieren**, bei dem bestimmte Felder als leer (o) definiert werden und der Rest mit Steinen (x) belegt ist. Dies ist hilfreich, um spezifische Startkonfigurationen für Tests oder Rätsel zu erstellen.
* setze\_os(FeldAlt, IndicesMitO, FeldNeu): Dies ist ein Hilfsprädikat für feld\_mit\_x/2, das rekursiv eine Liste von Indizes durchläuft und die entsprechenden Felder im Spielfeld auf "leer" (o) setzt.
* wende\_zuege\_an(AktuellesFeld, ListeDerZuege, Zielfeld, MaxTiefe): Dies ist der **Hauptlöser** des Spiels. Er versucht, eine **Sequenz von Zügen** (ListeDerZuege) zu finden, die von einem gegebenen AktuellesFeld zu einem Zielfeld führt. Die Suche wird auf eine bestimmte MaxTiefe (maximale Anzahl von Zügen) begrenzt, um unendliche Schleifen oder zu lange Berechnungen zu vermeiden. Das Prädikat wählt einen gültigen zug/3, wendet ihn an und versucht dann rekursiv, den Rest der Züge zu finden. Es ist wichtig zu beachten, dass durch die flexible Natur dieses Prädikats auch der Start- oder Zielzustand ermittelt werden kann, wenn die Züge und der jeweils andere Endzustand (Start- oder Zielfeld) bekannt sind.

**Wichtiger Hinweis zur Performance der Zugsuche**

Aktuell kann die Suche nach Lösungen, insbesondere bei komplexeren Problemen oder wenn wende\_zuege\_an/4 für eine größere MaxTiefe aufgerufen wird, relativ lange dauern. Der Grund dafür liegt in der Reihenfolge, wie die gültigen Züge in den zug/3-Prädikaten definiert sind:

* Der Code fragt zuerst alle **vertikalen Züge** (basierend auf spalten/1) ab.
* Erst danach werden die **horizontalen Züge** (basierend auf zeilen/1) in Betracht gezogen.

Das bedeutet, dass das System bei der Suche nach möglichen Zügen immer zuerst alle vertikalen Optionen durchprobiert, selbst wenn eine horizontale Bewegung die effektivere oder sogar die einzige Lösung in einem bestimmten Zustand wäre. Je nach Startkonfiguration des Spielfelds und der Verteilung der möglichen Züge kann diese Reihenfolge zu einer ineffizienten Suchstrategie führen, da möglicherweise viele unwirksame vertikale Züge evaluiert werden, bevor die relevanten horizontalen Züge überhaupt betrachtet werden.

Beispieleingabe:  
feld\_mit\_x([4,9,15,18,22,25],Z),feld\_mit\_x([16],F), wende\_zuege\_an(F, Zuege, Z, 5), zeige\_feld(Z)