報酬最大化原理にもとづく脳型AGIアーキテクチャの構想

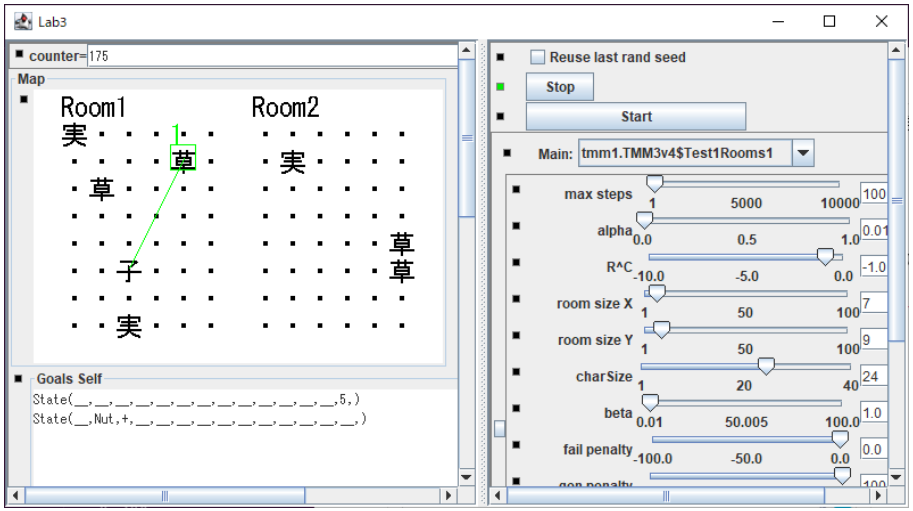
2021-08-06

産業技術総合研究所
人工知能研究センター
一杉裕志

※ 忌憚ないコメントよろしくお願いいたします。

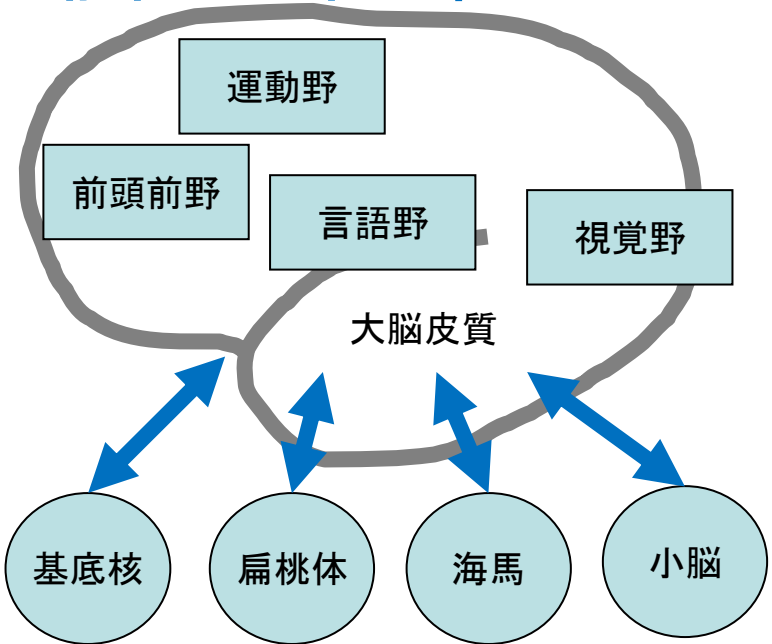
概要

- 脳型AGI実現を目指した我々のこれまでの研究
  - 設計指針の1つ: 報酬最大化原理
  - 3つの中核技術:
    再帰的強化学習、プログラム合成、生成モデル
- 脳型AGIのデモを目指した今後の研究構想
  - **実世界の本質的性質を保ったまま極力単純化した実験環境**



私の研究の目標

- 人類を豊かにするためには人間の仕事を低コストで代替できる機械が必要
- 脳のアーキテクチャをヒントにするのが近道
  → 脳型AGI
- AGI研究に携わる研究者は多くない
  → **AGIの実現可能性が一目でわかるデモが必要**



研究の前提

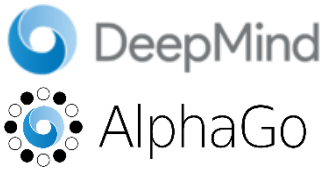
- AGIがいまだにできてない理由は汎用性が足りないのでなく、むしろ人間が得意とするタスクへの特殊化が足りない。
（Cf. ノーフリーランチ定理[Wolpert+ 1997]）

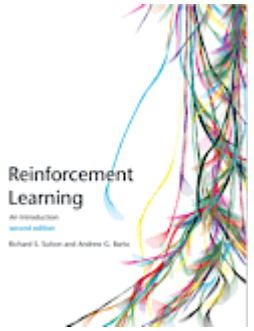
- 計算パワーだけに頼ってAGIが知識獲得することは不可能。事前知識の作り込みや他者からの知識伝達が重要。

- 人類がまず最初に必要とするAGIは、人間に解けない問題を解く機械ではなく、人間が仕事のやり方を教えればそれを人間程度の上手さで実行してくれる機械。

Reward is enough [Silver+ 2021]

David Silver, Satinder Singh, Doina Precup, and Richard S. Sutton.
Reward is enough.
Articial Intelligence, Vol. 299, p. 103535, 2021.





「知識、学習、知覚、社会的知性、言語、汎化、模倣、一般知能などの知性に関する能力はいずれも報酬最大化に役立つものとして理解できる。」

「十分に強力な強化学習エージェントから知能が生まれる。」

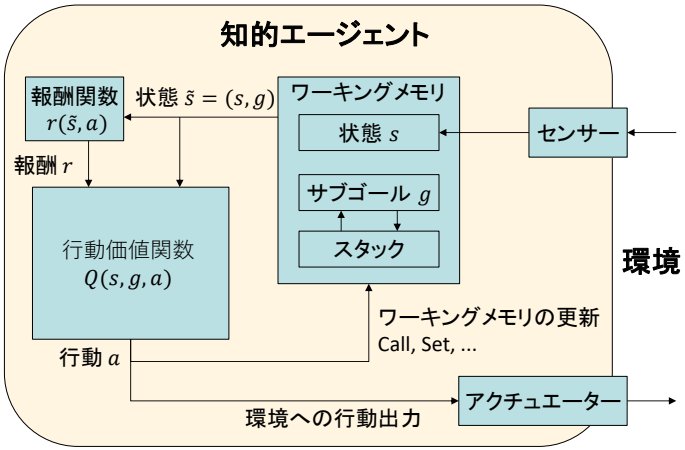
「AGIの理解・実現につながる。」

という仮説を主張。

報酬最大化原理

- **AGIの目的を報酬最大化であると仮定した設計方針**
  - 利点：アーキテクチャ全体の見通しがよくなる。
    個々のパーツの理論的妥当性の議論がしやすい。
- 実世界で動作する十分に性能の高い強化学習エージェントはAGI であると見なせる。おそらく脳も報酬最大化が目的。
- 我々のこれまでの研究：
  - 再帰的強化学習 RGoal を用いた前頭前野周辺のモデル
    [一杉+ 第9回汎用人工知能研究会 2018]
    [Ichisugi+ ICANN 2019]

再帰的強化学習＝
再帰的なサブルーチン呼び出しが
可能な階層型強化学習



階層型強化学習の利点[Dietterich 2000]

1. **サブタスク共有**:
サブルーチンをタスク間で共有することで学習を加速

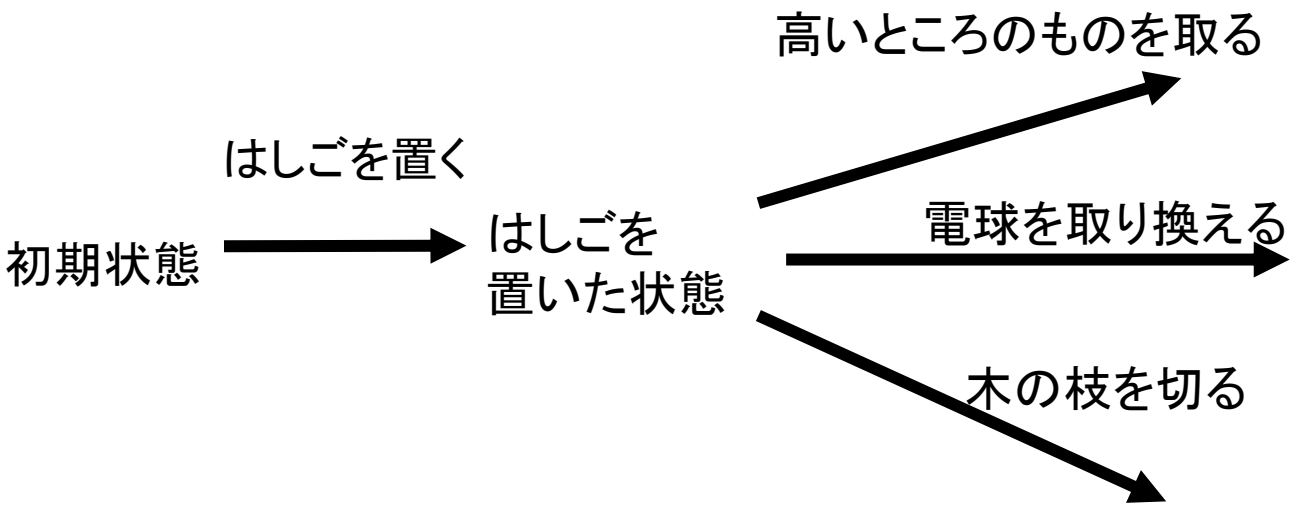
2. **時間抽象**:
行動をサブルーチンとして抽象化することで学習を加速

3. **状態抽象**:
サブルーチンごとに実行に関係ない情報を無視することで学習を加速

RGoal では再帰的なサブルーチン呼び出しを可能にすることで、これらの利点をより多く享受できる。

おそらくAGI実現に向けた最大のブレークスルーではないか？

サブルーチンの例



最終目的は違っていても、
「はしごを置く」というサブルーチンは共通

サブタスク共有

「はしごを置く」サブルーチンを実行中は
はしごの周辺以外の情報をすべて無視

状態抽象

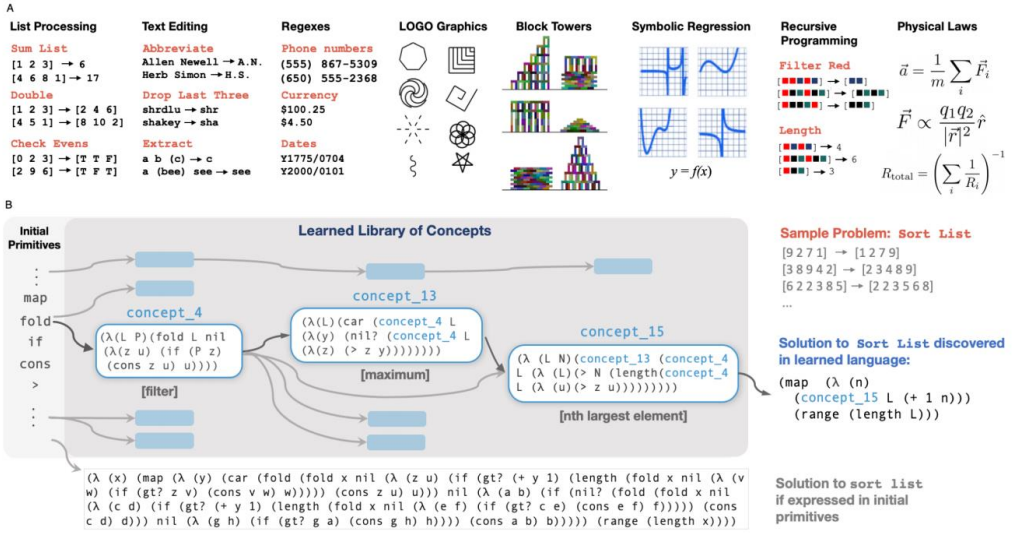
「はしごを置いた」時に得られる報酬は何？

- **RGoal では学習の目的は報酬最大化というよりコスト最小化**
  - サブルーチン実行中は毎ステップ負の報酬（コスト）
  - サブルーチン終了時（サブゴール到達時）の報酬は0
  - 報酬最大化 ＝ サブゴールに最小ステップで到達

- サブルーチンに関しては報酬設計が不要！
  → 前頭前野が行う理性的な知的活動に関しては報酬設計不要、
  生理的欲求などに関してのみ報酬設計が必要

プログラム合成

- プログラム合成は AGI 実現への有望なアプローチ
  - AIXI [Hutter 2000], MagicHaskeller [Katayama 2008, 2018], DNC(Differentiable Neural Computers) [Graves et al. 2016], DreamCoder [Ellis+ 2020]
- どういう「プログラミング言語」を使うかが、性能に大きく影響



例：DreamCoder の場合は
木構造をしたプログラムを合成

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sable-Meyer, Luc Cary, Lucas Morales, Luke Hewitt, Armando Solar-Lezama, and Joshua B Tenenbaum.
Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning.
arXiv preprint arXiv:2006.08381, 2020.

脳が学習により獲得する「プログラム」のイメージ

[一杉+ 第15回汎用人工知能研究会 2020]

行動ルール：スパースな整数値ベクトルのパターン、
状態行動対を抽象化したもの

プログラム：
100万個程度の
行動ルールの集合

(_,_,_,29,_,_,_,_,_,X,_,X,_,_,...)
(_,_,_,48,_,_,_,_,_,X,_,X,_,_,...)
(_,_,_,62,_,_,_,_,_,X,_,X,_,_,...)
...
(29,_,X,_,_,_,Y,_,_,_,12,_,_,Y,...)

参照・更新

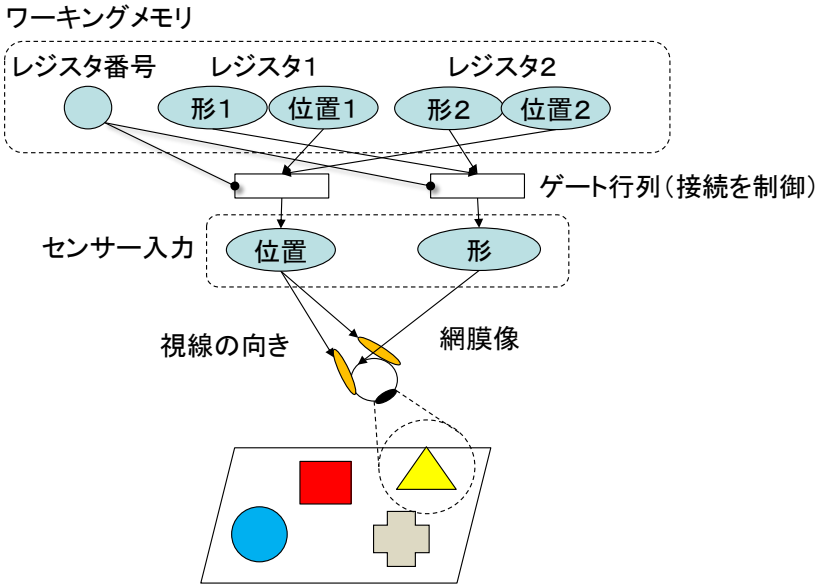
ワーキングメモリ：
100万次元程度の
スパースな
整数値ベクトル

(0,0,0,29,0,0,0,0,94,0,0,0,0,0,0,0,...)

プロダクションシステムや論理型言語に似た動作

ワーキングメモリと生成モデル

- もう1つの設計指針：**生成モデル原理**
  - ワーキングメモリの状態と環境の状態をベイジアンネットを使って生成モデルの形で結びつける
  - 大脳皮質ベイジアンネット仮説[Ichisugi 2007]が前提
- 例：脳の中にあるオブジェクトファイルをベイジアンネットで表現 [一杉+ 第16回汎用人工知能研究会 2020]



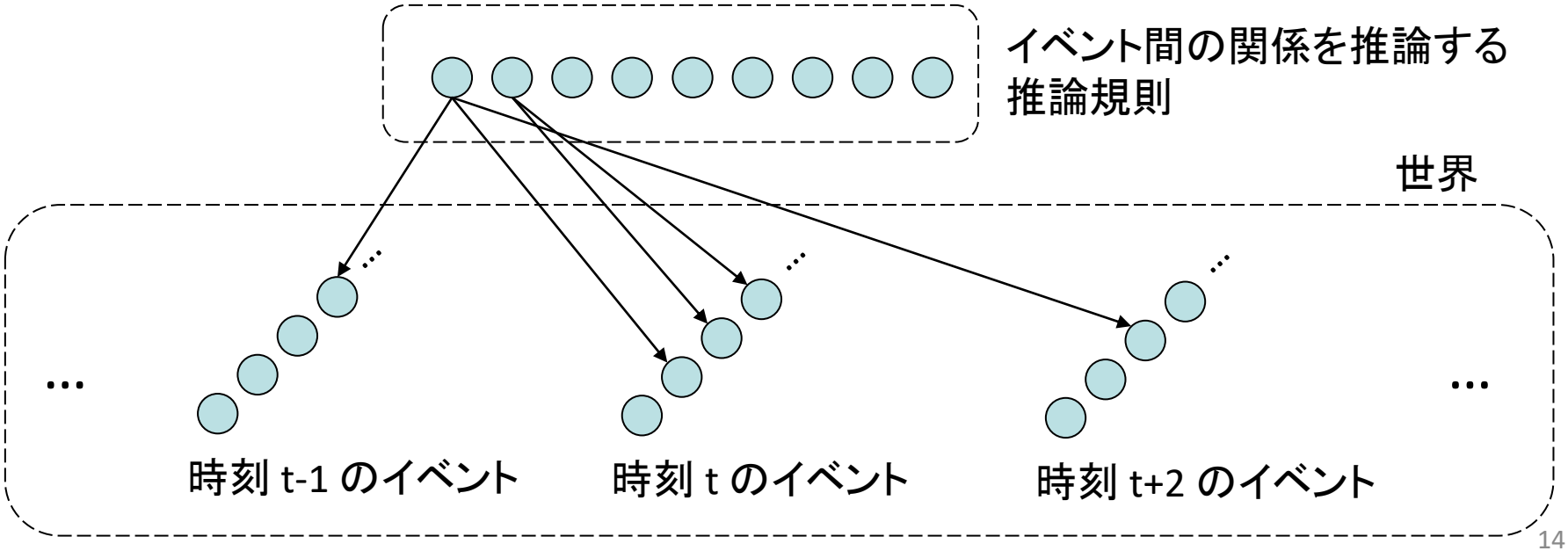
生成モデル原理の利点

- 文脈に応じたロバストな特徴抽出
- 事前知識の作り込みが容易
- モデルベース強化学習で利用可能
- 隠れた状態の推定に利用可能
- プログラム合成の解の探索空間を制限できる
- 記憶の保持と忘却を環境の状態に結びつけて設計できる
- ワーキングメモリ内で無意識のデータ表現の変換が可能
- 状態行動対テーブルの圧縮も可能

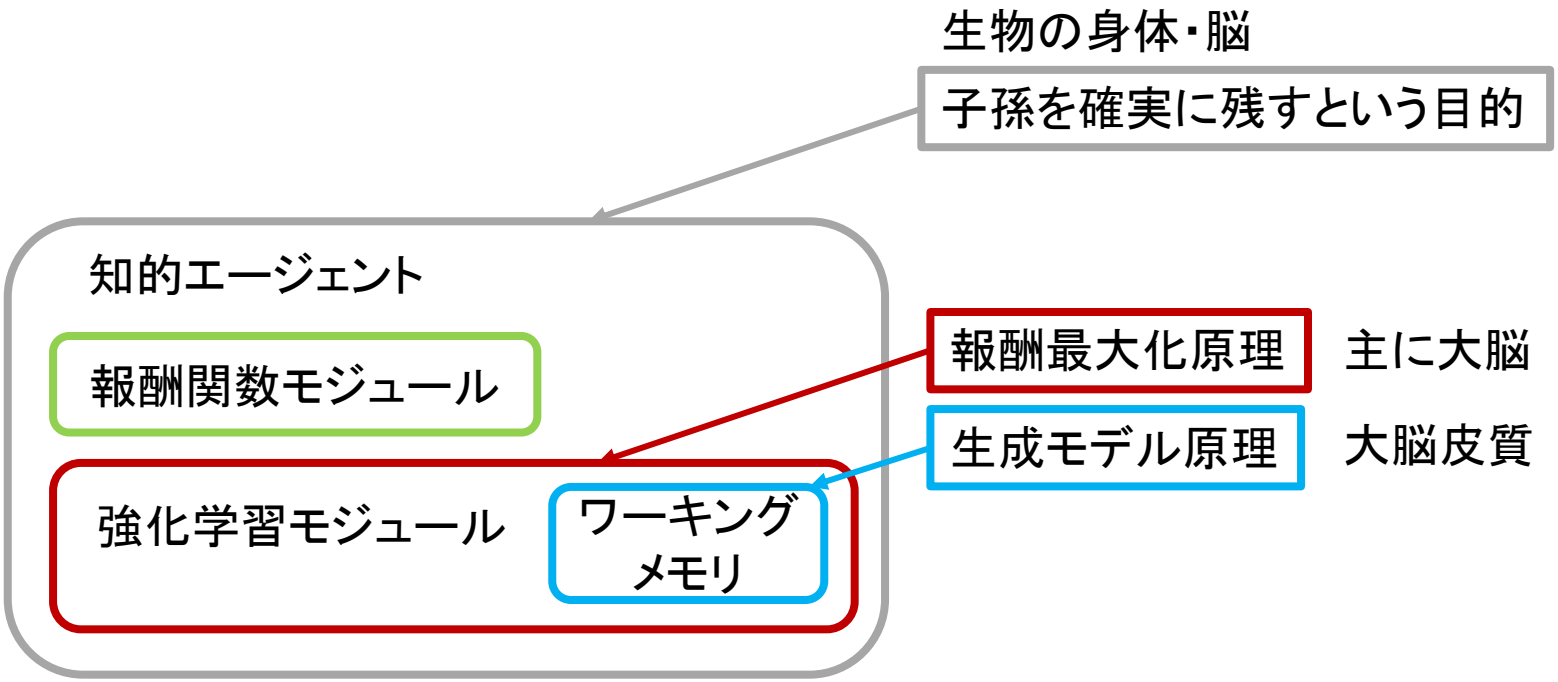
どれ1つとってもAGIの性能向上に大きく寄与するはず

エージェントが持つ「世界モデル」

- 時刻 t の世界の状態は無数のイベントに分解されて認識される。
- いくつかのイベントの間には規則的な関係がある。
- エージェントは観測可能なイベントと推論規則を組み合わせて未観測のイベントの中身を推論することができる。



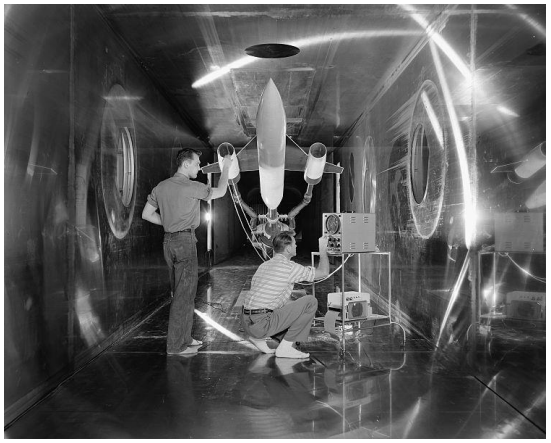
脳型AGIアーキテクチャ概念図



今後解決すべき重要な課題

エージェントの環境の設計

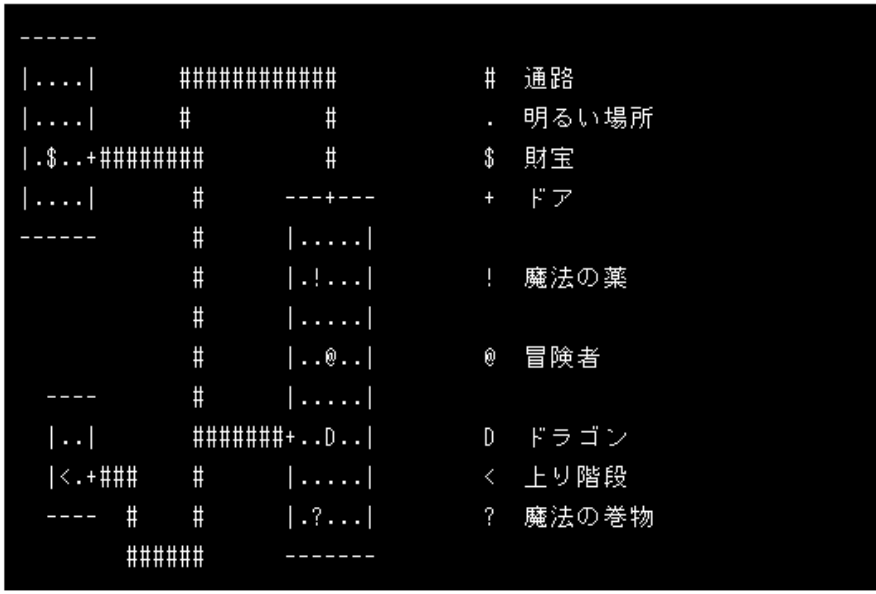
- 「豊かな環境では報酬を最大化するために様々な能力が要求されるため、豊かな知的能力が生み出される」
  [Silver+ 2021]
- AGI の目標は複雑な実世界の環境で動作すること
- 一方で、いきなり実世界でAGI開発をするのは困難
  - 計算コスト、実験コスト、ノイズや実験の再現性、・・・
- **実世界の本質的性質を保ったまま極力単純化した人工的な実験環境を作ることが重要**



https://ja.wikipedia.org/wiki/風洞

ローグライクゲーム

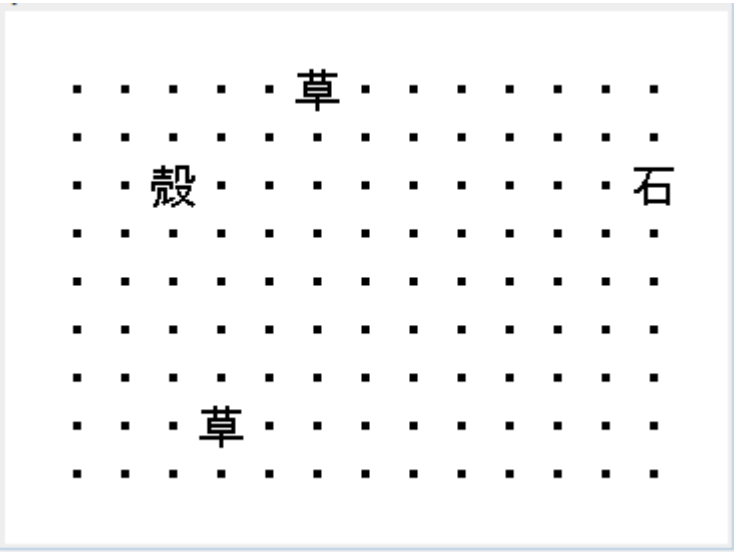
- 時間と空間が離散化
- マップ上のアイテムは記号化
- マップの大きさ、
  アイテムの数、
  物体同士の相互作用の数に制限がない
- 他者（敵）の動きが複雑
- プレイヤーは環境について経験から学習しなければならない



- NeurIPS 2021 におけるコンペティションの題材にもなっている

「ローグライクゲーム - Wikipedia」
https://ja.wikipedia.org/wiki/ローグライクゲーム

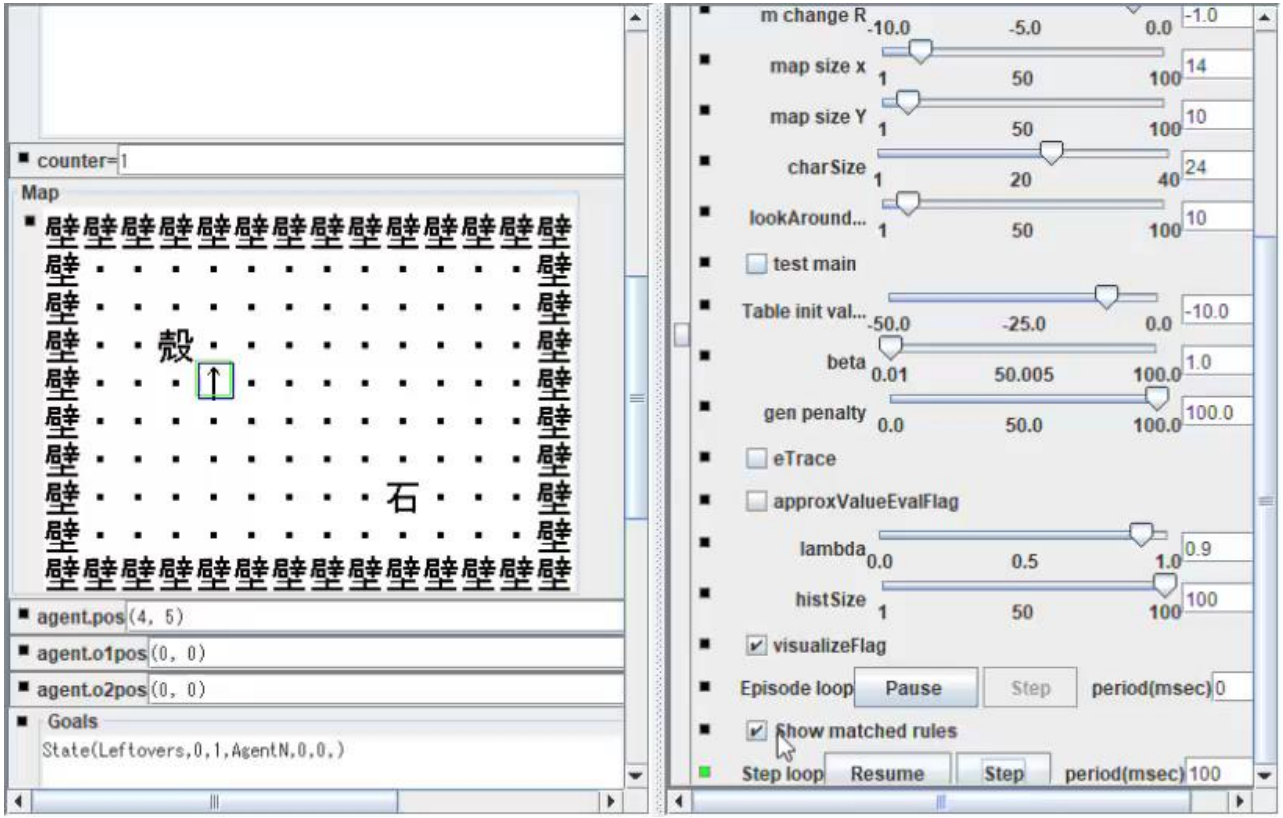
殻に石をぶつけて実を食べるタスク



「石」を「殻」にぶつけると「実」になる。
「実」は食べられる。
「実」を食べると「空」になる。

一杉裕志, 中田秀基, 高橋直人, 佐野崇,
物体操作に適したワーキングメモリを持つ汎用人工知能アーキテクチャの検討,
第16回 人工知能学会 汎用人工知能研究会(SIG-AGI), 2020.

最初に作ったプロトタイプ



・注意の機構を導入、サブルーチン実行に無関係な情報を無視
・身体中心座標と環境中心座標の分離し行動ルールの汎用性を高める

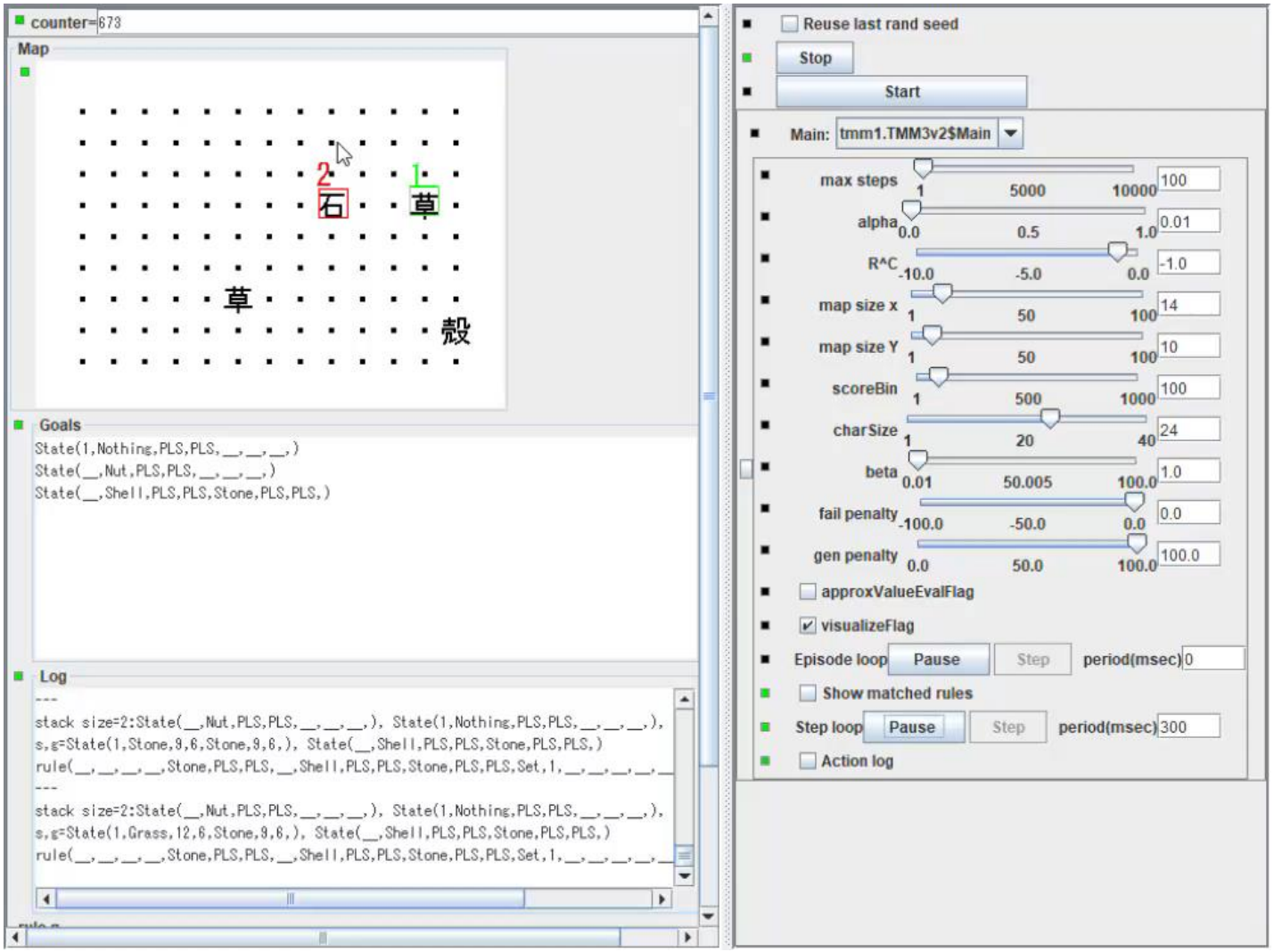
行動ルール集合（エージェントが経験から獲得しなければならないプログラムを私が手で書いてみたもの）

```
{   // o1 に近づくサブルーチン。
    // 物体２への注意をリセット。
    q(s(o1,x1,y1,o2,x2,y2), s(o1,0,1,A,0,0), Action.Reset2);
    // 前方の o1 に近づく。
    q(s(o1,x1,1,A,0,0), s(o1,0,1,A,0,0), Action.MoveForward);
    q(s(o1,x1,2,A,0,0), s(o1,0,1,A,0,0), Action.MoveForward);
    q(s(o1,x1,3,A,0,0), s(o1,0,1,A,0,0), Action.MoveForward);
    q(s(o1,x1,4,A,0,0), s(o1,0,1,A,0,0), Action.MoveForward);
    q(s(o1,x1,5,A,0,0), s(o1,0,1,A,0,0), Action.MoveForward);
    // 後方の o1 に向かって向きを変える。
    q(s(o1,x1,-1,A,0,0), s(o1,0,1,A,0,0), Action.TurnLeft);
    q(s(o1,x1,-2,A,0,0), s(o1,0,1,A,0,0), Action.TurnLeft);
    q(s(o1,x1,-3,A,0,0), s(o1,0,1,A,0,0), Action.TurnLeft);
    q(s(o1,x1,-4,A,0,0), s(o1,0,1,A,0,0), Action.TurnLeft);
    q(s(o1,x1,-5,A,0,0), s(o1,0,1,A,0,0), Action.TurnLeft);
    // 右の o1 に向きを変える。
    q(s(o1,1,0,A,0,0), s(o1,0,1,A,0,0), Action.TurnRight);
    q(s(o1,2,0,A,0,0), s(o1,0,1,A,0,0), Action.TurnRight);
    q(s(o1,3,0,A,0,0), s(o1,0,1,A,0,0), Action.TurnRight);
    q(s(o1,4,0,A,0,0), s(o1,0,1,A,0,0), Action.TurnRight);
    q(s(o1,5,0,A,0,0), s(o1,0,1,A,0,0), Action.TurnRight);
    // 左の o1 に向きを変える。
    q(s(o1,-1,0,A,0,0), s(o1,0,1,A,0,0), Action.TurnLeft);
    q(s(o1,-2,0,A,0,0), s(o1,0,1,A,0,0), Action.TurnLeft);
    q(s(o1,-3,0,A,0,0), s(o1,0,1,A,0,0), Action.TurnLeft);
    q(s(o1,-4,0,A,0,0), s(o1,0,1,A,0,0), Action.TurnLeft);
    q(s(o1,-5,0,A,0,0), s(o1,0,1,A,0,0), Action.TurnLeft);
}
{   // o1 を o2 の１つ手前の場所に移動するサブルーチン。
    StateN g = s(o1,0,1,o2,0,2);
    // 前方の o2 に近づく。
    q(s(o1,0,1,o2,x2,2), g, Action.MoveForward);
    q(s(o1,0,1,o2,x2,3), g, Action.MoveForward);
    q(s(o1,0,1,o2,x2,4), g, Action.MoveForward);
    q(s(o1,0,1,o2,x2,5), g, Action.MoveForward);
    q(s(o1,0,1,o2,x2,6), g, Action.MoveForward);
    // 後方の o2 に近づく。
    q(s(o1,0,1,o2,x2,0), g, Action.Pull);
    q(s(o1,0,1,o2,x2,-1), g, Action.Pull);
    q(s(o1,0,1,o2,x2,-2), g, Action.Pull);
    q(s(o1,0,1,o2,x2,-3), g, Action.Pull);
    q(s(o1,0,1,o2,x2,-4), g, Action.Pull);
    q(s(o1,0,1,o2,x2,-5), g, Action.Pull);
    // o1←o2 となった場合。回り込んで　→o1 o2　とする。
    q(s(o1,0,1,o2,0,-1), g, Action.TurnRight);
    q(s(o1,-1,0,o2,1,0), g, Action.MoveForward);
    q(s(o1,-1,-1,o2,1,-1), g, Action.TurnLeft);
    q(s(o1,-1,1,o2,-1,-1), g, Action.MoveForward);
    q(s(o1,-1,0,o2,-1,-2), g, Action.MoveForward);
    q(s(o1,-1,-1,o2,-1,-3), g, Action.TurnLeft);
    q(s(o1,-1,1,o2,-3,1), g, Action.MoveForward);
    q(s(o1,-1,0,o2,-3,0), g, Action.TurnLeft);
    // o1　 o2　となった場合は左側に回り込んで →o1　 o2 とする。
    // ↑
    q(s(o1,0,1,o2,x1,1), g, Action.TurnLeft);
    q(s(o1,1,0,o2,1,y2), g, Action.MoveForward);
    q(s(o1,1,-1,o2,1,y2), g, Action.TurnRight);
    q(s(o1,1,1,o2,x1,1), g, Action.MoveForward);
    q(s(o1,1,0,o2,x1,0), g, Action.TurnRight);
}
```

```
{   // Nut を食べるサブルーチン。
    StateN g = s(Leftovers,0,1,A,0,0);
    // 物体２を無視
    q(s(__,__,__,__,__,__), s(Leftovers,0,1,A,0,0), Reset2);
    // Nut を探す。
    q(s(__,__,__,A,0,0), g, c(Nut,x1,y1,A,0,0));
    {   // Nut が見つかった場合。
        // Nut のところに移動する。
        q(s(Nut,x1,y1,A,0,0), g, c(Nut,0,1,A,0,0));
    }
    {   // Nut が見つからない場合。
        // Stone と Shell を探す。
        q(s(A,0,0,A,0,0), g, c(Stone,x1,y1,A,0,0));
        q(s(Stone,x1,y1,A,0,0), g, c(Stone,x1,y1,Shell,x2,y2));
        // Shell を Stone の手前に移動。
        q(s(Stone,x1,y1,Shell,x2,y2), g, c(Stone,0,1,Shell,0,2));
        // 押す。（目の前に Nut が現れるはず。）
        q(s(Stone,0,1,Shell,0,2), g, MoveForward);
        // →石実　という状態なのでいったん右に出る。
        q(s(Stone,0,1,Nut,0,2), g, TurnRight);
        q(s(Stone,-1,0,Nut,-2,0), g, MoveForward);
        // あとは普通に　Nut を食べに行く。
        q(s(Stone,-1,-1,Nut,-2,-1), g, c(Nut,-2,-1,Nut,-2,-1));
        q(s(Nut,-2,-1,Nut,-2,-1), g, Reset2);
    }
    // 目の前に Nut がある場合は前に進む。
    q(s(Nut,0,1,__,__,__), g, MoveForward);// これは還ばれない。
    q(s(Nut,0,1,A,0,0), g, MoveForward);
}
{   // Stone を Shell の手前に移動するサブルーチン。
    StateN g = s(Stone,0,1,Shell,0,2);
    // いったん Shell を無視。
    q(s(Stone,x1,y1,Shell,x2,y2), g, Reset2);
    // Stone のところに移動
    q(s(Stone,x1,y1,A,0,0), g, c(Stone,0,1,A,0,0));
    // Stone が目の前にある時に Shell を再度探す。
    q(s(Stone,0,1,A,0,0), g, c(Stone,0,1,Shell,x2,y2));
    // o1 を o2 の１つ手前の場所に移動。
    //q(s(Stone,0,1,Shell,x2,y2), g, c(Stone,0,1,Shell,0,2));
}
{   // 物体１を探すサブルーチン。
    StateN g = s(o1,__,__,o2,__,__);
    q(s(__,__,__,o2,__,__), g, Find1);
}
{   // 物体２を探すサブルーチン。
    StateN g = s(o1,__,__,o2,__,__);
    q(s(o1,__,__,__,__,__), g, Find2);
}
```

問題点：座標の抽象化が不十分

改良版 [一杉+ 第16回 汎用人工知能研究会 2020.11]



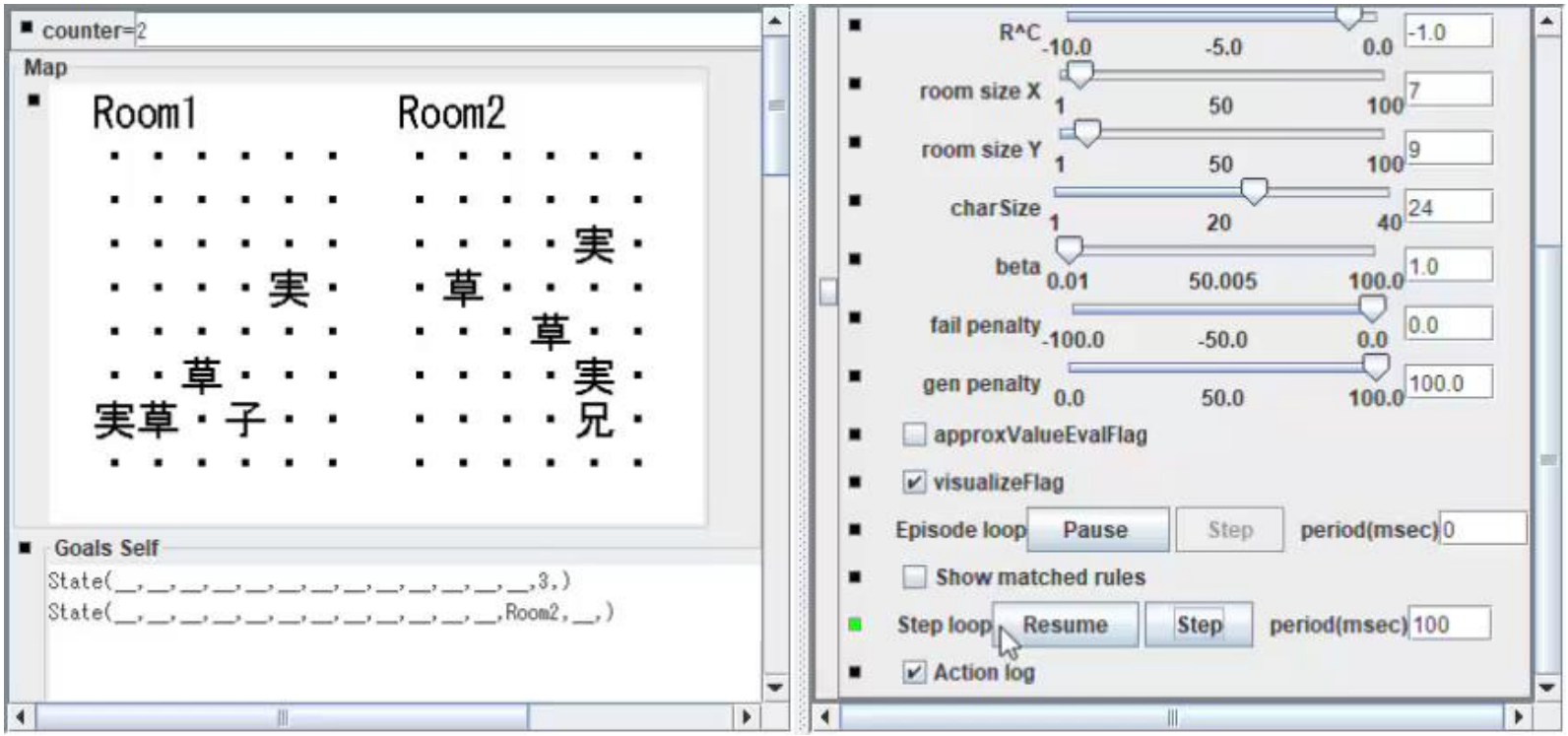
行動ルール集合

```
// Nut を食べるサブルーチン。
StateN g1 = s(__,Nothing,PLS,PLS,__,__,__);
rule(s(__,__,__,__,__,__,__), g1, call(__,Nut,PLS,PLS,__,__,__));
rule(s(__,Nut,PLS,PLS,__,__,__), g1, Eat01);

// Nut を手に入れるサブルーチン。
StateN g2 = s(__,Nut,PLS,PLS,__,__,__);
// Shell と Stone を探す。
rule(s(__,__,__,__,__,__,__), g2, call(__,Shell,PLS,PLS,Stone,PLS,PLS));
rule(s(__,Shell,PLS,PLS,Stone,PLS,PLS), g2, Move02to01);

// Shell と Stone をオブジェクトレジスタに入れるサブルーチン。
StateN g3 = s(__,Shell,PLS,PLS,Stone,PLS,PLS);
rule(s(__,__,__,__,__,__,__), g3, set(2,__,__,__,__,__,__));
rule(s(__,__,__,__,Stone,PLS,PLS), g3, set(1,__,__,__,__,__,__));
```

前のバージョンに比べて簡潔

理由:
- ユークリッド距離を排して座標表現を大胆に抽象化
- タスクに適したワーキングメモリの構造
- 眼球運動についてのハードウエア支援

開発中のバージョン



- 複数の部屋
- 他者と会話（文の意味を表現する論理式を直接やりとり）

極めて単純でありつつ実世界の本質的特徴をほぼ備えた環境

実現を目指すデモの例

- 報酬最大化（コスト最小化）する言語理解・発話・行動・推論
- 例：
  - **タスクを解くために必要な道具がなければ道具の作り方を他者に聞く。**
    （言語を通じた合目的的な知識獲得）
  - **他者から道具がある場所を聞かれたら最近見かけた場所を教えてあげる。**
    （社会性、知識共有）

- 再帰的強化学習、プログラム合成、生成モデルという中核技術と報酬最大化原理により十分実現可能

その他の課題

- データ構造の設計
- 社会性の設計
- メタ強化学習の設計
- ハードウエア支援の設計
- 実世界での動作に向けた個々の構成要素の性能向上
  - 大規模化可能なベイジアンネット
  - 記憶の複数のタイムスケールの機構
  - 効率的な思考の機構

まとめ

- 脳型AGI実現を目指したこれまでの研究
  - 設計指針：**報酬最大化原理、生成モデル原理**
  - 3つの中核技術：
    再帰的強化学習、プログラム合成、生成モデル
- 脳型AGIのデモを目指した今後の研究構想
  - **実世界の本質的性質を保ったまま極力単純化した実験環境**
  - 目指すデモ：報酬最大化（コスト最小化）する言語理解・発話・行動・推論

こういうアプローチの研究にご関心のある方、ご協力いたします。
過去の関連論文： https://staff.aist.go.jp/y-ichisugi/j-index.html

知識獲得を促進する機構の例

- 身体に作り込まれた機構が知識獲得を促進

- 目立った視覚刺激の場所に**自動的に視線を向ける**機構が**能動的に視線を向ける**行動の獲得を促進
- 熱や痛みから**自動的に体を離す**機構が**能動的に不快刺激から体を避ける**行動の獲得を促進
- 耳から入った言葉を**自動的にオウム返しする**機構が**能動的に知覚内容を発話**する行動の獲得を促進
- 現在の状況に似たエピソードを**自動的に想起する**機構が**能動的に必要なエピソードを想起**する行動の獲得を促進