有限オートマトンを用いてボードゲームのルールを記述する言語の設計

伊藤滉貴
明治大学 総合数理学部 先端メディアサイエンス学科

概要:近年，将棋や囲碁などのボードゲームをプレイする人工知能が注目を浴びているが，ゲームのクオリティを評価することに関する研究は少なく，人間にとってゲームが面白いかという主観的な指標を評価することは難しい．もしゲームのクオリティをコンピュータが正確に評価できればゲームをデザインする効率の向上が見込める．非電源ゲームを制作する多くのゲームデザイナーはルールの検証や拡張のためにプロトタイプを制作するが，そのためには多くの時間と材料を要する．一方デジタルゲームにおいては比較的容易に検証や拡張が可能であり，エージェントを用いてテストプレイを自動化できる．そこで本稿ではボードゲームデザイナー向けにボードゲームのルールを記述するビジュアルプログラミング言語(以降，本言語)を設計した．本言語はコンポーネントの状態で構成される有限オートマトンによってゲームを表現し，状態遷移図によりルールを可視化することでプログラミングに慣れていないデザイナーにとって実装しやすい．
キーワード:ボードゲーム，ゲームデザイン，プログラミング言語，有限オートマトン

Design of a Language for Describing Rule Sets of Board Game with Finite State Machine

Koki ITO
Department of Frontier Media Science, School of Interdisciplinary Mathematical Science, Meiji University

Abstract Agents that play board games like *shogi* and *go* have recently been developed greatly. However, there are few studies which focus on evaluating quality of game, it is so difficult to evaluate an subjective index of “how human players enjoy games.” If we find a method that evaluates the quality of a game correctly, efficiency of designing games will be improved. Every board game designer makes a prototype of a game to verify and extend rules of the game, but it takes the designer time and materials. On digital games, the verification and extension are easier and it is possible to make agents test playing the games. In this paper, the author designed a visual programming language of describing rule sets of board games for such game designers. A game description in the language consists of Finite State Machine, a set of component’s states, and the state transition diagrams makes it possible to visualize rules so that game designers that are not familiar with programming can design games easier.
Keyword Board Game, Game Design, Programming Language, Finite State Machine

1. はじめに

近年，将棋や囲碁のトッププレイヤー相手にコンピュータプレイヤーが勝利し，完全情報ゲームにおける強い AI は大きな成果をあげた．AlphaZero[1]はチェス，将棋，囲碁の 3 つのゲームにおいて当時のトップレベルのプログラムを上回り，ゲームに特化した手法を使用せずに強化学習のみで学習する有用性を示した．そのため，ゲーム AI の分野では不完全情報ゲームや，相手のスキルに合わせたプレイを行う接待 AI などが注目され始めている．しかしながら，ボードゲームのルールの評価や生成に関する研究は少なく，ルールデザインのようなアイデアや創造性を要する分野では未だに人間の領域であるという見方が強い．

デジタルゲームをデザインする人工知能は既にいくつかの分野で活用されている．マップの自動生成に関してはローグライクゲームのように従来から手法が確立されている．またメタ AI というゲームシステムに介入しゲームの難易度やユーザーの緊張度を調整する人工知能がある．また ANGELINA[2]はビデオゲームのレベルデザインを行うシステムで，マップの生成，評価を繰り返すことでレベルの自動調整を実現している．このようにデジタルゲームの分野ではゲームデザイン AI の研究が盛んだが，ゲームを一から人工知能に任せて生成することは困難である．

ルールデザインにおける最大の課題はルールの評価にある．ランダムにルールを生成したとしてもそのゲームがプレイ可能であることを証明するには一般的にゲーム木を全探索する必要があり，組み合わせの大きいゲームにおいては困難である．またプレイ可能だったとしてもそのゲームの面白さには複雑な要素が絡み合い，また主観的な指標でもあるので評価が難しい．

ボードゲームは日々開発されていて，企業だけでなく個人でもゲームデザインから制作，販売まで手がける者もいる．非電源ゲームとして販売を想定している多くのゲームデザイナーは企画段階で実際にゲームを制作しテストプレイを重ねてルールを調整するため，材料の調達やプレイヤーと時間の確保が必要になる．もしゲームを自動的に評価したりいくつかルールの候補を提示したりするツールがあれば，デザイナーに大きく貢献できるが現状では難しい．一方デジタルゲームでは実装する過程で論理的にルールを定義する必要があるためルールの整合性やバグに気づきやすく比較的ルールの調整にかかる時間は少ない．また簡単なエージェントを用意すれば自動でテストプレイを行うことも可能である．ゲームを拡張するにあたっても物理的な制約が少ないデジタルゲームの方が容易である．

本稿ではそのようなデジタルゲームにおけるゲームデザインの利点に着目し，ボードゲームのデザインを支援するためにルールを記述するビジュアルプログラミング言語を設計した．本言語は有限オートマトンを用いて視覚的にルールを表現，記述できるため，テキストプログラミングに慣れていない

デザイナーも実装しやすい.

2. 関連研究

2.1 General Game Playing

ボードゲームのルールを記述する言語の 1 つに General Game Playing (GGP)[3]で用いられる Game Description Language がある. GGP とはスタンフォード大学のプロジェクトの 1 つで, 未知のゲームをプレイできる汎用的なプログラムの実現を目標としている. GGP には任意のゲームをプレイできる環境が整っていて, 予めいくつかのエージェントが用意されており人間同士のプレイだけでなくコンピュータ相手のプレイも可能である. またゲームのスタイルシートとユーザーインターフェースを記述すれば GUI でユーザーがプレイすることもできる.

2.1.1 Game Description Language

Game Description Language (GDL)は GGP においてゲームを記述する言語である. GDL の文法は Datalog と類似する点がいくつかあり論理型言語である. 通常, 前置記法で記述され 11 のキーワードを用いてゲームを表現する. GDL は幅広いゲームを表現できるが, その汎用性故にルールが複雑なゲームを表現するには多くの記述を要する. また数字と文字列は区別されず同じ定数として処理されるため, 四則演算などの基本的な機能が備わっていない. これらの理由により GDL でゲームを正しく表現するには高度な技術を要するため, ボードゲームデザイナーが GGP を使って新しいゲームをシミュレーションすることは難しい.

一方で GDL を使って多くのボードゲームが開発されている. GGP.org[4]には多くのゲームを提供するリポジトリがあり, その中には五目並べやチェス, 将棋, エイト・クイーンを含む 314 個のゲームがある. このように多くのゲームが公開されている理由の 1 つはゲームの構成にあるといえる. ゲームは主に GDL で記述するルールシート, ゲームを可視化するスタイルシート, ユーザーがゲームをプレイするためのユーザーインターフェースの 3 つで構成されていて, ルールとビジュアルが完全に独立している. ルールシート以外の 2 つは任意であり純粋にルールだけを記述すればゲームをプレイできることがメリットである.

2.2 Ludi

本稿と同じようにゲームデザイナーへの貢献を目的とした研究として, 組み合わせゲームのルールを自動的に生成する Ludi[5]というシステムが挙げられる. Ludi は 2 人用の組み合わせゲームを対象として既存のゲームを学習し, 遺伝的アルゴリズムによってゲームを自動的に生成, 評価することで新しいゲームを生成する仕組みになっている. 実際このシステムを使って生成したゲームの 1 つに Yavalath という五目並べに似たゲームがあり, 製品化して販売されている. しかし Ludi が生成したゲームの多くは Yavalath のように五目並べに似たルールに帰着するようだ. Ludi は面白いゲームを自動的に生み出すというアプローチに価値があることを証明したが, 既存のゲームを学習するという手法を採用していることにより学習データの不足や偏りが生じることが課題である.

2.2.1 Ludi GDL

Ludi では GGP と同じようにゲームを記述する Ludi GDL という言語がある. Ludi GDL は GDL より高水準で, 盤などのコンポーネントや四則演算などの機能が予め用意されている. そのため表現できるゲームは限られるが, 人間にとってより理解や実装がしやすい言語である. しかし GDL と互換性がないため, Ludi GDL で生成されたゲームは GGP で利用できる充実した環境を活用することはできない.

3. 表現可能なゲーム

本言語は GDL との互換性を保つことで GDL に変換することが可能になり, 開発したゲームを GGP のプレイ環境を用いてプレイすることができる. しかしながら GDL のように広い範囲のゲームを表現可能であると構造が複雑になりプログラマーの負担が大きいため, 盤などのコンポーネントを予め定義することで表現できるゲームに制限を設けた. また GDL で表現できる代表的なゲームを表現できるように設計した.

3.1 ゲームの要件

以下のように GGP と同様の要件[6]を定める.

(1) Termination
初期状態から打った合法手が有限のターンを経て終局状態に達するゲームを *terminate* するという.

(2) Playability
初期状態から終局するまでの状態でどのプレイヤーにも少なくとも 1 つの合法手があるゲームを *playable* であるという.

(3) Winnability
他のプレイヤーが終局状態だったとしても, 得点が最大値になる終局状態に持ち込むような一連の行動を何人かのプレイヤーが着手できるゲームを *strongly winnable* であるといい, それらの行動を全てのプレイヤーが持ち合わせているゲームを *weakly winnable* であるという.

(4) Well-formedness
terminate し playable かつ weakly winnable であるゲームを *well-formed* であるという. 本言語でも GDL と同様に well-formed でないゲームを表現することはできるが, GGP の環境における動作を想定した場合 well-formed であることが望ましい. しかし一般的にゲームが well-formed であることを確かめるにはゲームツリーを総当たりで探索する必要がある.

3.2 ゲームの性質

また本言語では GDL と同様[7]に以下の性質を持つゲームのみ表現できる. ゲームの分類は文献[8]を参考にした.

- 離散: プレイヤーが有限の純粋戦略から手を選ぶこと
- 確定性: 偶然の要素を含まないこと
- n 人プレイ ($n \geq 1$)

GDL と同様, プレイヤーが認知できる範囲を制限すること

で完全情報ゲームだけでなく不完全情報ゲームも実装できる. また将棋のような逐次手番ゲームだけでなくじゃんけんのような同時手番ゲームも表現できる.

3.3 コンポーネント

ターン (turn) とプレイヤー (player) に加え, 以下の 5 つのコンポーネントを使用できる. ただし以下のコンポーネントについては必ずしも使用する必要はない.

(1) ピース (Piece)
将棋の駒, 囲碁の碁石に相当するコンポーネントである. 作成できる数に制限はない.

(2) 盤 (Board)
将棋や囲碁などで使用されるコンポーネントの一つである. 2 次元のマスで構成されていて n×m (n, m は自然数)の任意のサイズを指定できる. マスは座標, ピース, ピースを所有するプレイヤーの 3 つで構成されている. 一つのマスに複数のピースを置くことはできない.

(3) 場 (Field)
将棋では持ち駒を置いておく駒台, 囲碁は相手の取り上げた石を置いておく碁笥の上蓋に相当する場所である. プレイヤーが各々の場を持ち, 有限個のピースを置くことができる.

(4) 得点 (Score)
各プレイヤーに得点を設定できる. 得点には上限がある.

(5) 数学演算 (Math)
整数を扱い四則演算や数値の比較ができる. 得点の換算式やピースを数える際に使用する.

3.4 表現できる代表的なゲーム

GGP.org に存在する以下の代表的なゲームを表現できる.

- 三目並べ
- 五目並べ
- リバーシ
- チェス
- 将棋
- エイト・クイーン

4. ルール記述

4.1 ゲーム状態

盤上のあるマスに特定のピースが置かれているといったコンポーネントの状態をゲーム状態という. ゲーム状態はあらゆるルールにおいて基本的な概念で, 2 つの状態を比較することでルールの条件を判定し, 状態の遷移を定義することで手を実装できる. 以降状態とはゲーム状態を示す.

4.2 ルールの構成

ゲームのルールを以下の 5 つに分類する.

(1) 初期状態 (Initial State)
ゲームを始める前の状態. プレイヤーやピースの定義, 盤や場の状態などを設定できる.

(2) 終了条件 (Terminal States)
ゲームが終了するときの条件(状態). この条件を満たすとゲームはそのターンで終了となる.

(3) 合法手 (Legal Moves)
プレイヤーが行ってもよい手. 状態から条件を指定し合法手を定義する. 合法手がないプレイヤーは暗黙的に何もしないことを示す noop を着手する.

(4) 状態遷移 (Game State Update)
次のターンで遷移する状態. 基本的にはプレイヤーが選択した手を条件として分岐し盤にピースを置くなどの着手を定義する. GDL では遷移しない状態を含め全ての次の状態を定義する必要があるが, 本言語ではこのルールで定義されない状態は暗黙的に次のターンへ引き継がれる.

(5) 勝利条件 (Goal States)
プレイヤーが勝利するときの条件(状態). ゲームが終了したときに判定され勝敗が決まる. なお得点が勝敗に影響するとは限らない.

4.3 ルール関数

ルール関数とはルールを記述するための関数で, 2 つのゲーム状態を比較する状態比較関数や, ゲーム状態の遷移を定義する状態定義関数の 2 つがある. ルール関数はコンポーネント, 関数名, 引数の 3 つで構成される. なお引数に不正な値を使用した場合は偽が返る. 関数は *Component.functionName(arity1, arity2, ...)* のように記述する. ここでは盤のルール関数のみ取り上げるが, 他のコンポーネントについては[9]を参照すること.

4.3.1 ゲーム状態の表記法

ルール関数においてゲーム状態を表記する方法は以下の 3 つである.

(1) 定数
定義済みのコンポーネントの名前(文字列)や整数, 多項式のことで, ゲーム状態を直接指定する方法である. 文字列の前に~を挿入するとそれ以外のコンポーネント全てを総称する. 文字列の場合, 全て大文字で記述する.

(2) 変数
変数には引数に指定できるあらゆるコンポーネントを与えることが可能で, ゲーム状態を間接的に指定する方法である. 数学演算の状態比較関数において変数に値を代入することで宣言できる. 未定義の変数には可能な限り演算子 any が代入される. any が代入不可能な引数に未定義の変数を使用することはできない. 変数は全て小文字で記述する. なお演算子で既に使われている名前は利用できない.

(3) 演算子
演算子は比較演算子, any, all などの演算子の 2 つがあり, 包括的にゲーム状態を指定する方法である. any 演算子は該当するコンポーネントのいずれか, all 演算

子は該当するコンポーネントの全てを表す．全て小文字である．

4.3.2 状態比較関数

2 つのゲーム状態を比較し条件を満たすかどうかを判定するルール関数である．例えば盤の状態比較関数は以下の 3 つである．

(1) Board.true(po, pl, pi)
特定の座標 po にプレイヤーpl が持つピース pi があるという状態を判定する．po にピースがないことは pl = NONE, pi = NONE と記述する．座標が盤の範囲外であれば必ず偽を返す．また pl と pi には any という演算子を指定することが可能で，例えばいずれかのピースでよい場合は pi = any とする．

(2) Board.count(pl, pi, n)
盤の範囲内にプレイヤーpl が持つピース pi が n 個あるという状態を判定する．

(3) Board.size(w, h)
盤のサイズが w×h であるかを判定する．w = any, h = any とすると盤のサイズを取得することができる．

4.3.3 状態定義関数

あるゲーム状態から次のゲーム状態を定義する関数である．関数の定義が正しければ遷移に成功し真を返す．偽を返す状態定義関数はルールに問題があるため推奨されない．例えば盤の状態定義関数は以下の 2 つである．

(4) Board.set(po, pl, pi)
盤にピースを設置する．引数は関数(1)と同様である．座標にピースが既にあっても必ず上書きされる．

(5) setSize(w, h)
盤のサイズを設定する．引数は関数(3)と同様である．初期状態で 1 回だけ使用できる．

4.4 ゲームステートマシン

4.4.1 形式的定義

ルールはゲームステートマシン(GSM)というε-動作を許した非決定性有限オートマトンで構成される．ただし入力が遷移によって決定されるなどの拡張を行っている．GSM は次のような要素からなる 6 組 $A = (Q, q_0, F, \Sigma, \delta, R_g)$ のことである．定義は文献[10], [11]を参考にした．

- Q: ゲーム状態の空でない有限集合
- $q_0 \in Q$ 初期状態
- $F \subseteq Q$ 受理状態の集合
- $\Sigma = \{0, 1\}$ 入力記号
- $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ 状態遷移関数
- $R_g: P \to \Sigma\ (P \subseteq Q)$ ゲーム状態gにおけるルール関数

ここではゲーム状態をオートマトンにおける状態と同一であるとみなしている．またQのような集合を状態集合という．入力は無入力の状態から開始し，状態qにおいて次に入力される文字iは以下のように自動的に決定される．

$$i = R_g(q)$$

$\delta(q, i) = \emptyset$ のとき，即ち遷移先の状態が定義されていない場合，入力は拒否されその状態に留まる．または $\delta(q, i) = \{q\}$ のとき，即ち遷移先が現在状態のみである場合も同様にその状態に留まることになる．しかし次の入力は現在状態によって決定されるため前と同じ値が入力される．つまり同じ状態に留まり続けることとなるため，GSM では現在状態が前の状態と同じ場合，その時点で入力を終了することとする．また$R_g(q)$が定義されていなければ次の入力は与えられないが，ε-動作により遷移することはできる．このようなルール関数が定義されない状態を無入力状態という．

4.4.2 GSM の分割

一つのオートマトンで全てのルールを構築すると構造が複雑になってしまう．そこであるオートマトンを 2 つのオートマトンに分割することを考える．

まずある GSM $A = (Q, q_0, F, \Sigma, \delta, R_g)$ を 2 つの部分集合に分割する．Aの定義は 4.4.1 と同様であり，状態集合 $Q^* \subset Q$を以下のように定義する．なお $\overline{Q^*} = Q - Q^*, q_0^* \in Q^*, q_f^* \in Q^*$ とする．

- $\forall q \in \overline{Q^*}, \exists \sigma \in \Sigma, \delta(q, \sigma) = P_1 \subseteq (\overline{Q^*} \cup \{q_0^*\} \cup \emptyset)$
- $\delta(q_0^*, \sigma_0^*) = P_0^*\ (P_0^* \neq \emptyset, P_0^* \neq \{q_0^*\})$
- $\forall q \in \left(Q^* - \{q_f^*\}\right), \exists \sigma \in \Sigma, \delta(q, \sigma) = P_1^* \subseteq (Q^* \cup \emptyset)$
- $\exists q \in Q^*, \delta(q, \sigma) = P_2^* \ni q_f^*$

このときのQ^*を独立状態集合と呼ぶ．分割できるオートマトンは以上の条件を満たすAである必要がある．

次に A を $\overline{Q^*}$ とある状態 q_*^B を含む GSM $B = (Q^B, q_0^B, F^B, \Sigma^B, \delta^B, R_g^B)$ と，Q^*とある受理状態q_f^Cを含む GSM $C = (Q^C, q_0^C, F^C, \Sigma^C, \delta^C, R_g^C)$ に分割する．なおゲーム状態gは全てのオートマトンで共通でなければならない．Bの定義は以下の通りである．

- $Q^B = \overline{Q^*} \cup \{q_*^B\}$
- $q_0^B = q_0$
- $F^B = F$
- $\Sigma^B = \Sigma$
- $\delta^B: Q^B \times \Sigma^B \to 2^{Q^B}$
- $R_g^B: P \to \Sigma^B\ (P \subseteq Q^B)$

Bの状態遷移関数は以下のように定義する．なお$P_1^B = \delta(q, \sigma), P_2^B = \delta(q_f^*, \sigma)$とする．

- $\forall q \in \overline{Q^*}, \delta^B(q, \sigma) = \begin{cases} (P_1^B - \{q_0^*\}) \cup \{q_*^B\}\ (\text{if } P_1^B \ni q_0^*) \\ P_1^B\ (\text{otherwise}) \end{cases}$
- $\delta^B(q_*^B, \sigma) = \begin{cases} (P_2^B - \{q_f^*\}) \cup \{q_*^B\}\ (\text{if } P_2^B \ni q_f^*) \\ P_2^B\ (\text{otherwise}) \end{cases}$

Cの定義は以下の通りである．

- $Q^C = Q^* \cup \{q_f^C\}$
- $q_0^C = q_0^*$

- $F^C = \{q_f^C\}$
- $\Sigma^C = \Sigma$
- $\delta^C: Q^C \times \Sigma^C \to 2^{Q^C}$
- $R_g^C: P \to \Sigma^C \ (P \subseteq Q^C)$

Cの状態遷移関数は以下のように定義する.

- $\forall q \in \left(Q^* - \{q_f^*\}\right), \delta^C(q,\sigma) = \delta(q,\sigma)$
- $\delta^C(q_f^*, 1) = \{q_f^C\}$

次にBの状態q_*^Bのルール関数を定義する. q_*^Bはq_0^*の遷移先をq_f^*のものに置き換えることでQ^*を統合した状態である. q_*^Bのルール関数はCの出力に依存する. 有限オートマトンの出力は受理, 非受理の二値であり, 不受理を 0, 受理を 1 とすると出力は$\omega = \{0,1\}$と表せる. 即ちゲーム状態gにおけるCの出力関数は$\lambda_g^C \to \omega$となる. $\omega = \Sigma$よりq_*^Bのルール関数は$R_g^B(q_*^B) = \lambda_g^C$と表せる. このような等式が成り立つとき, $q_*^B = C$のようにオートマトンを 1 つの状態と扱うことにする. なおある状態に対応するオートマトンがその状態を含むオートマトンであった場合, ループが発生するため許されない. BとCの定義は以上であり, gにおけるオートマトンの集合を$\mathbf{A}_g = \{B, C\}$としたとき, Aと$\mathbf{A}_g$は等価である.

Cの出力がq_*^Bのルール関数の値になるということは, Cはq_*^Bを条件としたルール関数であるといえる. 即ちq_*^Bにルール関数の条件としてゲーム状態を加えることで独自のルール関数を定義することができる. 独自のルール関数は複数のオートマトンで共通する手続きを一つにまとめることができるため構造の単純化に役立つ. ただし既存のルール関数を元に構築するため分割しても能力に差はない.

4.5 ゲームの構築

ゲームの全てのルールは GSM の集合$\mathbf{A} = \{G, A_0, A_1, A_2, A_3, A_4\}$によって構築できる. $G = \left(Q, A_0, \{q_f\}, \Sigma, \delta, R_g\right)$は全てのゲームに共通するオートマトンで次のような要素からなる. なおG以外のオートマトンは 4.2 で示したルールを表している.

(1) 状態
$A_0 \in Q$ 初期状態オートマトン
$A_1 \in Q$ 終了条件オートマトン
$A_2 \in Q$ 合法手オートマトン
$A_3 \in Q$ 状態遷移オートマトン
$A_4 \in Q$ 勝利条件オートマトン
$q_f \in Q$ 受理状態

(2) 遷移関数
$\delta(A_0, 1) = \{A_2\}$
$\delta(A_1, 0) = \{A_3\}, \delta(A_1, 1) = \{A_4\}$
$\delta(A_2, 1) = \{A_1\}$
$\delta(A_3, 1) = \{A_2\}$
$\delta(A_4, 1) = \{q_f\}$

Gはゲームの流れを表している. この流れに則って各オートマトンを実装することでゲームを構築できる. なおGの出力が非受理であった場合, それはルールにバグがあり状態が正しく遷移しなかったことを意味している.

5. ルールのビジュアル表現

有限オートマトンは状態遷移図によって視覚的に表現. 即ち GSM の遷移も図によって表現可能である. 本言語は有限オートマトンを用いて視覚的にルールを表現, 記述できるため, テキストプログラミングに慣れていないデザイナーも実装しやすい.

5.1 GSM 図

GSM の状態遷移図は図 1 のような有向グラフで示される. 初期状態は何もないところから矢印で指されている状態, 受理ノードは枠が二重線である状態である. 無入力状態は円で示される. 左上のオートマトンの名前は他のオートマトンにおいて状態として扱う際に使用する. 状態は矢印の向きに遷移する. 詳しい状態と遷移の記述は図 2 の通りである. ルール関数がある状態の一行目にはルール関数のコンポーネントの名前, 二行目には関数名と引数を記述する. コンポーネントの名前は状態がオートマトンの場合, オートマトンの名前になる. オートマトンにも関数名や引数を与えることができる. 遷移には条件として入力記号を記述する. ここでは 0 を f, 1 を t としている. 条件を書かない場合はε-動作による遷移となる. また 4.5 で示した全てのゲームに共通するオートマトンGの図を図 3 に示す.

5.2 表記の省略

GSM 図では可読性の向上と構造の単純化を目的として一部記法を拡張している. 例えば関数名が true の場合は名前と括弧を省略して引数だけで記述できるといった省略記法がいくつかあり, 特に数学演算においては数学の表記法に近づけている. 引数も同様に省略記法があり盤の座標とプレイヤーに関しては以下のような記法がある. なおこれらの概念は GSM の定義から外れているので実際は変換される.

(1) 座標
座標は絶対座標と相対座標の 2 つがある. 絶対座標は角括弧で括って[x, y]と表し相対座標は丸括弧で括って(x, y)と表す. 相対座標は[0, 0]を基準として相対的に座標を表す. ある状態のルール関数内で絶対座標を定義した場合, 遷移後にルート座標はその座標に移動する.

(2) プレイヤー
プレイヤーpl の次のターンプレイヤーを>>pl, 前のターンプレイヤーを<<pl と表記できる.

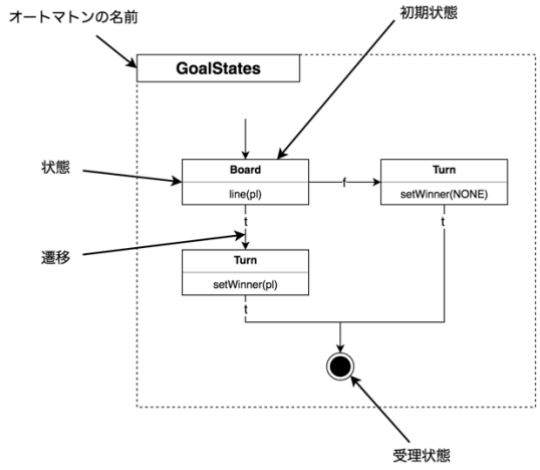

図 1. GSM 図の例

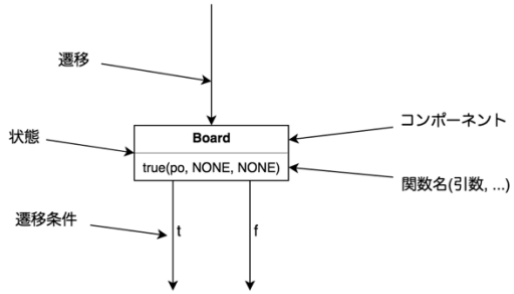

図 2. 状態と遷移の記述例

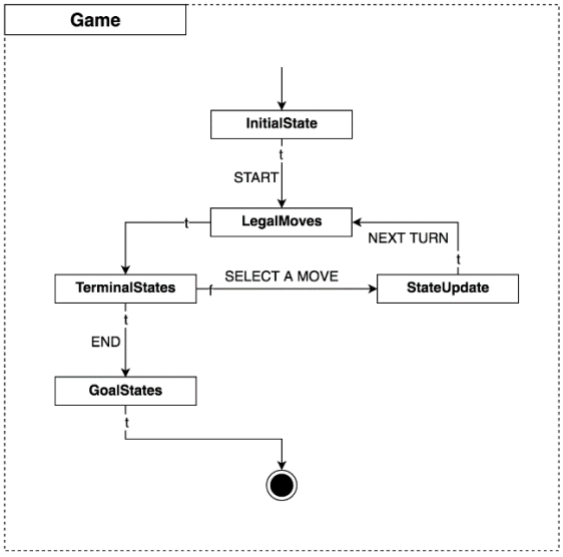

図 3. オートマトンGの図

5.3 三目並べの GSM 図

例として三目並べ(○×ゲーム)の GSM 図を付録 A に示した.

6. 課題と展望

本研究の課題としてルールの検証が困難であることが挙げられる. 本言語は定義済みのコンポーネントの範囲であれば自由にルールを記述することができるが, ルールがゲームの要件を満たす well-formed であるか検証することは難しい. 検証するには一般的にゲーム木を全探索し完全解析しなければならないので, ゲームが複雑であるほど困難になる. これは GDL でも同様に発生する課題である. また GSM の性質上, 入力が現在状態にのみ依存するため遷移の回帰が発生する可能性がある. ただしこのような課題はエージェントによる探索である程度払拭することは可能である.

今後の展望として本言語で表現できるゲームの拡張のため, GDL-II[12]への対応を検討している. GDL-II は"GDL for games with incomplete information."という意味で即ち不完全情報ゲームのために拡張された GDL である. GDL に 2 つのキーワードを追加することで逐次手番ゲームでもプレイヤーが把握できる状態を制限することが可能になる. また random というプレイヤーがランダムに手を選ぶことを事前に定義することで, 不確定ゲームにも対応している. 現状でも GSM 上で独自のルール関数を追加することで不完全情報には対応することはできるが, 言語側で対応することを考えている. GGP の環境では GDL-II に対応しており本言語でもルール関数を拡張することで容易に対応することができる. 他にもコンポーネントを追加することで表現できるゲームを拡張することができる.

また本言語を用いてゲームデザインを支援するフレームワークの設計を検討している. フレームワークの構造は図 4 の通りである. フレームワークは大きく 3 つに分かれる. フレームワーク部分は Game Design App 上でユーザーがゲームを実装し, GSM を生成する. GSM を GDL に変換するツールが GDL Factory である. GDL のプログラムは 4 つに分類され, Math Library は数学演算, Components はゲームのコンポーネント, Rule Functions はそれらのコンポーネントを使ったルール関数が記述されている. ここで定義されたルール関数を使ってゲームは構築される. 3 つのプログラムと GDL Factory が出力したプログラムを統合して GDL で記述されたゲームが完成する. 生成された GDL を GGP の環境で読み込むとゲームをプレイすることができる.

本研究のような ICT によるボードゲームのデザインを支援する研究は未開拓の分野である. 本研究は単なるゲームをデザインできるフレームワークの完成を目標としておらず, ボードゲームデザイナーがよりよいゲームを制作できる環境を目指している. 今後, ゲームを評価する手法が確立しフレームワークに実装されれば, より支援システムとしての効果が見込めると考えている.

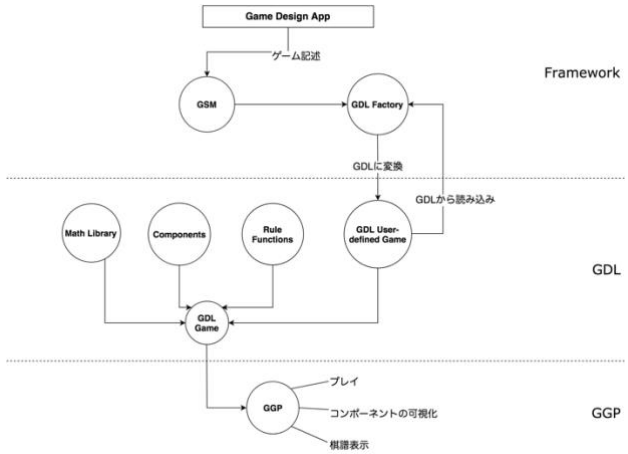

図 4. 本言語を用いたフレームワークの構造案

謝辞 本稿を執筆するにあたり，ご指導頂いた指導教員の阿原一志教授に感謝申し上げます．

参 考 文 献

[1] Silver, D., et al. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm.* : arXiv preprint arXiv:1712.01815, 2017.

[2] Michael, Cook, Simon, Colton and Jeremy, Gow. *The ANGELINA Videogame Design System—Part I.* : IEEE Transactions on Computational Intelligence and AI in Games, 2016.

[3] Michael, Genesereth and Michael, Thielscher. *General Game Playing: Synthesis Lectures on Artificial Intelligence and Machine Learning.* : Morgan & Claypool Publishers, 2014.

[4] Leland Stanford Junior University. GGP.org Games. *GGP.org.* [Online] [Cited: 2 6, 2019.] http://games.ggp.org/.

[5] Cameron, Browne and Frederic, Maire. *Evolutionary Game Design.* : IEEE Transactions on Computational Intelligence and AI in Games, 2010.

[6] Michael, Genesereth and Michael, Thielscher. *General Game Playing: Synthesis Lectures on Artificial Intelligence and Machine Learning.* : Morgan & Claypool Publishers, 2014. p. 27.

[7] Nathaniel, Love, et al. General Game Playing: Game Description Language Specification. [Online] 2008. [Cited: 2 6, 2019.] http://logic.stanford.edu/classes/cs227/2013/readings/gdl_spec.pdf.

[8] 伊藤毅志，保木邦仁 ，三宅陽一郎．ゲーム情報学概論 ―ゲームを切り拓く人工知能―．: コロナ社, 2018.

[9] 伊藤滉貴. Game Design Assistor. (オンライン) (引用日: 2019 年 2 月 6 日.) https://scrapbox.io/gda/.

[10] 藤芳明生．形式言語・オートマトン入門．: 数理工学社, 2017.

[11] Minami. オートマトン NOTE. (オンライン) (引用日: 2019 年 2 月 6 日.) http://deepwave.web.fc2.com/automata1.pdf.

[12] Michael, Thielscher. *A General Game Description Language for Incomplete Information Games.* : Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, 2010.

Appendix

A. 三目並べの GSM 図

三目並べの GSM 図を図 5 から図 10 に示す．オートマトン G は図 3 と同じである．図 10 は独自ルール関数 Board.line(pl)のオートマトンであり，プレイヤーplが三目並べたかどうかを判定する状態比較関数である．

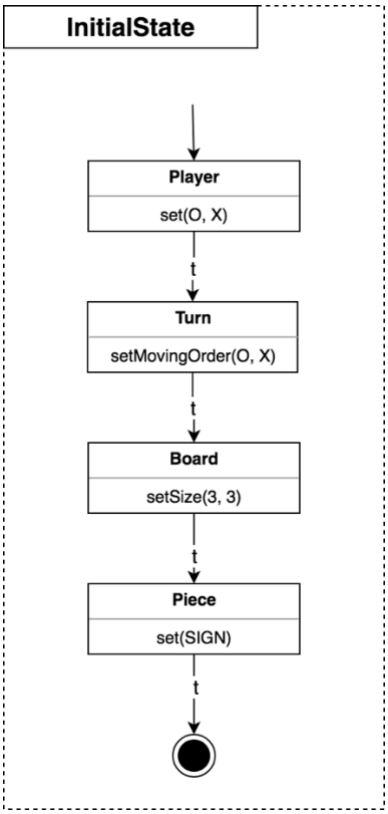

図 5. 初期状態

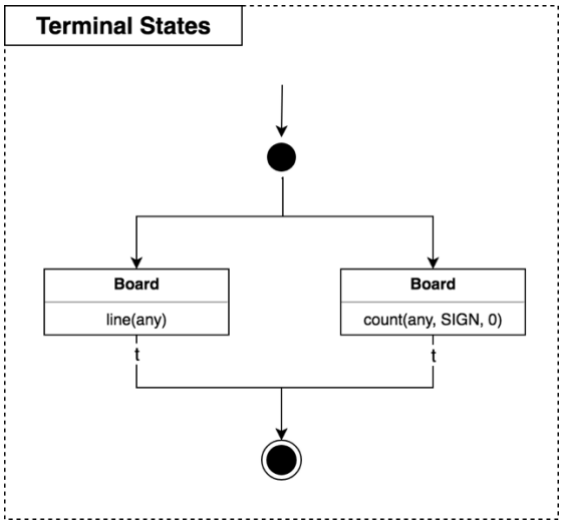

図 6. 終了条件

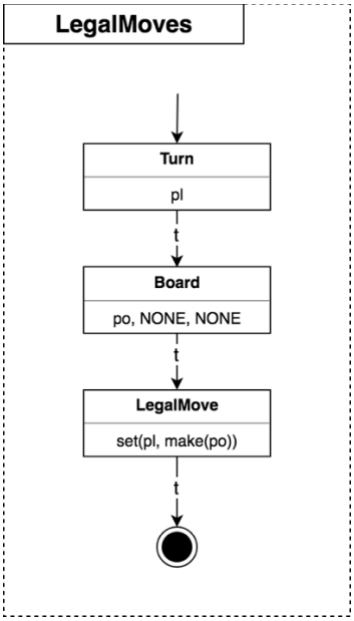

図 7. 合法手

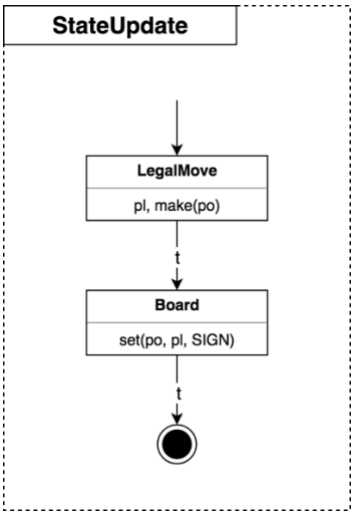

図 8. 状態遷移

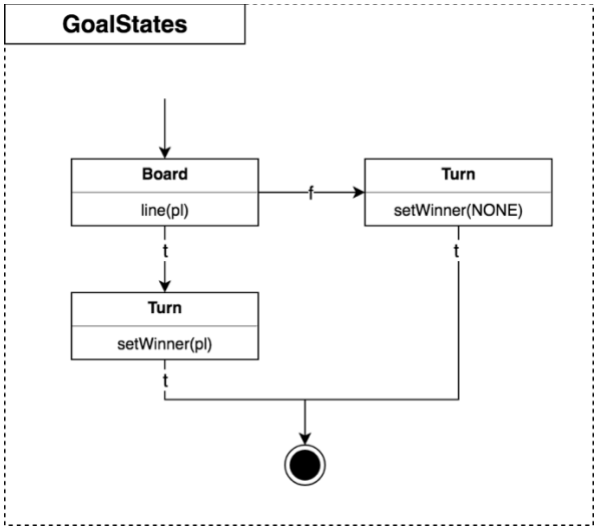

図 9. 勝利条件

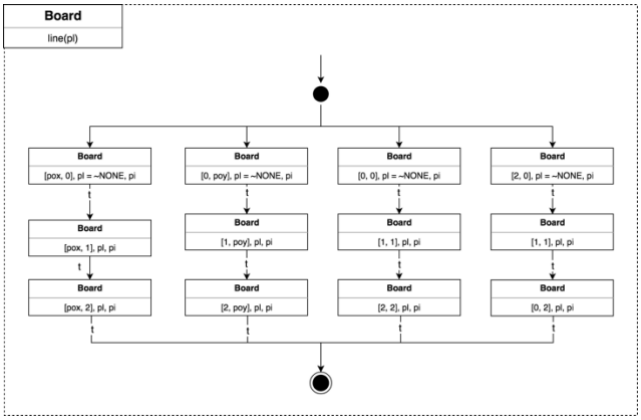

図 10. 独自ルール関数 Board.line(pl)