F-056

遺伝的プログラミングによる球技ゲーム生成

Generating Ball Sports Game by Usinng Genetic Programming

伊藤 黎† 三宅 陽一郎‡

Ray Ito Youichiro Miyake

1. はじめに

ゲームは，人々が娯楽や競争の場を楽しむための重要な要素として，世界中で人気を博している．特に，球技ゲームはフィジカルスポーツとしてもデジタルゲームとしても幅広い人々に親しまれている．テニスやバドミントンなどは，ボールと地面の接触という事象を軸とした単純なルールから生まれる刺激的なプレイ体験から，人気スポーツゲームの筆頭に挙げられる．

近年，情報技術の進歩により，ゲーム開発の領域でも革新的な進展が見られている．例えばデジタルゲームではキャラクター管理・制御，マップ生成など，多様な形態で先進技術がゲーム開発者を支えている [1].

その究極的な姿として，ゲームデザイン自体をアルゴリズムによって創発させようとする人工知能研究が進んでいる．Riedl, Bulitko (2012) は文脈を人工知能によって生成することで，ゲームの物語を生成する研究をしている [2]. 一方で，新しい取り組みとして，Lanzi, Loiacono (2023) が LLM を用いたゲームデザインフレームワークを策定している [3].

しかし，人工知能によってデザインされたゲームが実際に遊べる形で提供された例はまだ少ない．本研究は，実際にデジタル上でもフィジカルスポーツとしても遊べる，人工知能により生成されたゲームを創発することを目的としている．

本研究では，その創発のフレームワークとして，遺伝的プログラミングを用いた．遺伝的プログラミングは，木構造同士の枝葉を交叉・変異させることで進化論的に最適な木構造を導き出すアルゴリズムである．これを利用するために，本稿の 2 章で関連研究について述べた上で，3 章にて球技ゲームを木構造で表現する．4 章はその木構造を用いて遺伝的プログラミングによる創発を試みる．5 章は木構造で表されたゲームルールを実際にデジタル上で遊べるようにゲームエンジンで実装したことを紹介する．

2. 関連研究

遺伝的プログラミングによるゲーム生成の先行研究として Crowne, Marie (2010)の研究がある． [4]これはボードゲームの生成アルゴリズムである．彼らは，ボードゲームのルールを木構造で表現した．この木構造表現を対象として，遺伝的プログラミングを適用することで新しい木構造表現を獲得した． しかし，創発されたゲームが実際にプレイ可能なのか，ゲームとして成立しているか，或いはゲームとして面白いか，を判定する必要がある．それに対し自動プレイによって「プレイ可能性」「面白さ」の評価をつけた．この結果として，有名なボードゲームである「Yavalath[1]」が生まれた．これは，4 目並べであるが，3 個連続で碁石を置くと負けるという単純かつ斬新なゲームである．

本研究は，手法として Crowne, Marie と同じプロセスをたどるが，これまで探求されてこなかったアクション性のあるゲームである球技ゲームの創発に取り組む．

3. 球技ゲームの表現

ゲーム同士を合成するために，ゲームルールの過不足の無い木構造の表現を策定する必要がある．そこで，まずは球技ゲームを明確に定義したのち，その表現フレームワークである GDL（Game Description Language）について述べる．

3.1 球技ゲームの定義

まずは，本稿で対象とする球技ゲームの定義を明示する．定義として，次の性質を持つゲームを球技ゲームとする：

(3.1.1) プレイヤーとボールが存在する．
(3.1.2) プレイヤーがボールを操作することにより，ゲームの勝敗が決する．

これに加え，単純化のために，考察対象を下記のような性質をもつ球技ゲームに制約する：

(3.1.3) プレイヤーが 2 人のみ存在し，その二人が敵同士として対戦する．
(3.1.4) ボールは 1 つのみ
(3.1.5) コートは左右対称で，左右それぞれに「内側」と「外側」の空間が定義され，左右のいずれかがプレイヤーから見て「味方側」で，もう片方が「敵側」となる．
(3.1.6) 一度のゲームで同じイテレーション[2]が繰り返され，そのイテレーション内で勝敗が判定され，その繰り返しの結果として最終的にゲーム全体の勝敗が判定される．

(3.1.1) - (3.1.6) を満たす具体例として，テニス，卓球[3]，バドミントン[4]のシングルスルールが挙げられる．

これ以降，球技ゲームはこれらの制約を満たすものを指すこととする．

† 東京大学工学部システム創成学科 Systems Innovation, Faculty of Engineering, The University of Tokyo
‡ 東京大学生産技術研究所 Institute of Industrial Science, The University of Tokyo

1 https://boardgamegeek.com/boardgame/33767/yavalath
2 テニスにおける「ラリー」が具体例として挙げられる．
3 テーブルの上面を「内側」，地面を「外側」と解釈している．
4 シャトルを空気抵抗の強く受けるボールと解釈している．

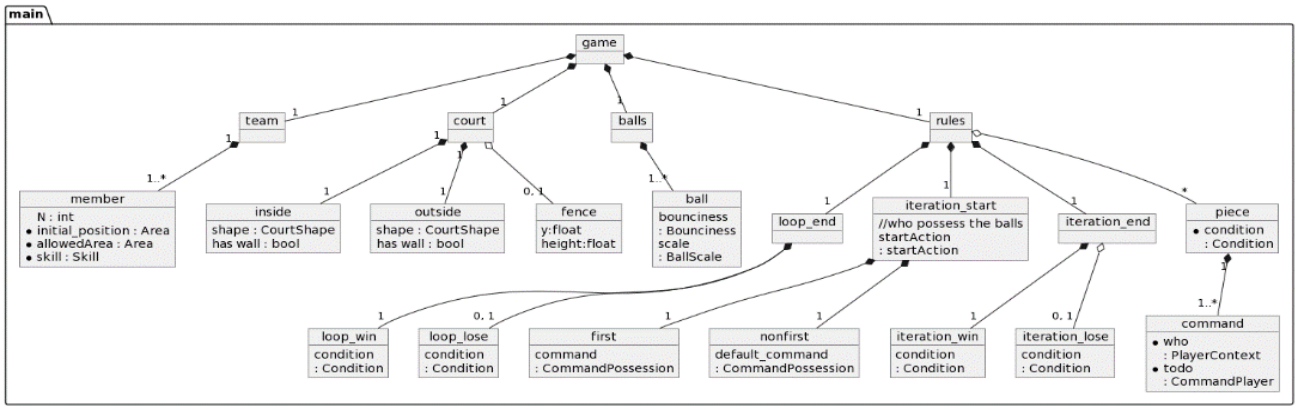

図 1 game ludeme の全体構造. 各長方形が ludeme を表し. 長方形の中のリストにおいて, コロンの左がパラメーター ludeme の名称, 右が型を表す.

3.2 GDL (Game Description Language)

遺伝的プログラミングを適用する要件として，ゲームルールが木構造で表される必要がある．これを実現するために，木構造でゲームルールを表現する文法 が必要となる．ここでは，ゲームルールを情報単位 ludeme に分解し，それを LISP に似た文法で構成する表現文法 GDL (Game Description Language)を策定した．これは木構造であり，遺伝的プログラミングによる合成が可能となっている．

3.2.1 Ludeme

Crowne, Marie (2010) の ludeme の考え方を採用する. これは,「meme」の派生語として, 最初に Borvo (1977) によって考案された [4] [5].

ludeme とは, ゲームルールの情報単位を表す表現形式である. ゲームはこの ludeme をコンポーネントとして構成されていると考えることができる.

ここでの ludeme には, 型と値が定義されている. ludeme の型とは, この ludeme が何の情報を表すかの枠組みである. 型の値とは, その型の中で予め定義されている具体的な値のことである. 例えば, courtShape という型に対して square という値が存在する. ただし, 後述する構造 ludeme のように値を持たない ludeme も存在する. また, 数式, 数値も ludeme 型の一つであると考えている. これにより, ludeme の文法的な一貫性を得ることができるためである.

この ludeme の文法は, 下記の通り LISP を踏襲している：

(タイトル パラメーターludeme0 パラメーターludeme1 …)

タイトルとは, 型名あるいはその型の値を示す. 値の無い型の場合には型名が, 値が存在する場合には値が記入される. 全ての値は型間で重複しないように定義されているため, 値から型を取得することが可能である.

パラメーターludeme は, 情報を具体化する構成要素として, ludeme の中での入れ子構造として保持されている. 型によって, パラメーターludeme の位置関係からそれが何を表すのかを決定する位置パラメータータイプと, パラメーターludeme の型から逆算してその意味を決定する非位置パラメータータイプがある.

また, パラメーターludeme が存在しない場合, 括弧は省略できる.

具体例として, テニスのコートを表す ludeme は次のようになる:

```
(court
   (inside
      (square 12.0 8.2 0.0)
      false
   )
   (中略)
)
```

これは, court 型は非位置パラメーターとして inside 型を要請している. これにより, court ludeme 内の inside ludeme は「コートの内側」であると認知される. パラメーター inside 型は非位置パラメーターとして courtShape 型, bool 型を要請することで, それぞれ「内側の形状」,「フェンスの有無」の情報であると解釈される. パラメーターとなっている square は courtShape 型の値であり, 位置パラメーターとして float 型 3 つを要請し, それぞれの値が指定された軸での大きさを表す.

この文法は LISP の文法を踏襲しているため, この入れ子構造が木構造であるといえることを強調する. このため, 遺伝的プログラミングによる合成が可能となっている.

3.2.2 Game Ludeme

ゲームルールは, 3.2 で述べた ludeme のうち, game ludeme である. game ludeme はゲームルール全体を表す. 3.2 で述べた通り ludeme は木構造であるので, 遺伝的プログラミングを適用することができる. このため, game ludeme の合成はゲームの合成と等価であり, ゲーム同士の合成が可能となったといえる.

ここで, game ludeme の全体像を概説する. 図 1 は game ludeme の基本構造を表す. 各長方形の繋がり方が全ゲームルールの共通構造となる. この図の長方形のタイトルとして挙げられている ludeme を構造 ludeme と呼ぶこととする. この構造 ludeme はゲームルール全体の中でどの箇所を記述するのかを指定している性質を持ち、これを中心にゲームルールを読み解くこととなる. ゲームルールは, まず 1. チーム, 2. コート, 3. ボール, 4. 進行ルールの 4 つに大別される. これは, game ludeme が team, court, balls, rule ludeme を保持するという形で実装されている. この 4 つは次のような情報を保持している：

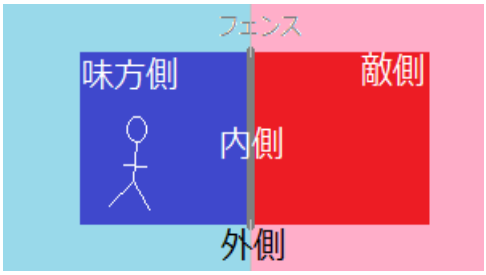

図 2 テニスコートの例における「内側」「敵側」「味方側」「敵側」の定義.「味方側」「敵側」は棒人間の立場から記述している.

1. チーム

team ludeme には, プレイヤーの情報が記述される. 複数プレイヤーに対応するため「チーム」という呼称にしたが, 本稿では制約 (3.1.3) により各チーム一人のみに限定される. team ludeme にはプレイヤーの初期位置, 行動可能範囲, そして許可されているアクションが明示される.

初期位置, 行動可能範囲は, コートとの位置関係によって示される. 例えば, テニスの場合, プレイヤーの初期位置は「味方側[1]の内側より後ろ」というように指定される. 初期位置はこのように位置を表現する area ludeme を一つ, 行動可能範囲は複数の area ludeme の和集合によって表現される.

許可されているアクションは, 移動以外にプレイヤーができるアクションを指している. これは「スキル」という名称で実装されている. 例えば, bounce スキルが許可されている場合, プレイヤーは飛んできたボールを弾き返すことができる. プレイヤーはこのスキルを複数個保持することができる.

2. コート

court ludeme には, コートの情報が記述される. 制約 (3.1.5) により左右対称であるので, 片側の情報が記述される. 同制約によりコートの内側と外側が定義される. 図 2 はテニスの例を示す. court ludeme には内側の形状, 外側の形状, フェンスの有無とその形状が明示される.

内側, 外側の形状は courtShape 型の ludeme により表される. courtShape は形状とそのメートル単位での大きさを含んでいる.

フェンスの有無は, court ludeme に fence ludeme が含まれているかどうかで判定される. fence ludeme にはフェンスの高さ, 横幅の長さが表されている.

3. ボール

balls ludeme には, ゲームに登場するボールの情報が記述される. 複数ボールに対応するため「Balls」という呼称にされているが, 本稿では制約 (3.1.4) によりボールが一つのみ存在しているため, balls ludeme には ball ludeme が一つのみ存在することとなる.

ball ludeme にはボールの弾力性, 大きさ, 空気抵抗の受けやすさが記述される.

4. 進行ルール

rules ludeme には, ゲーム進行に関する情報が記述される. 制約 (3.1.6)により, ゲームの進行は「イテレーション」とその「ループ」で構成される. rules ludeme では, イテレーション開始時のボール分配方法 (iteration_start ludeme), イテレーション終了条件とその勝敗 (iteration_end ludeme), ループ終了条件とその勝敗 (loop_end ludeme), そして補足ルール(piece ludeme)が記述される.

イテレーション開始時のボール分配方法は, 初回イテレーションと非初回イテレーションで区別される. この理由は, 初回イテレーションでは定義できないボール分配方法（「前イテレーション勝者がサーブ権を持つ」など）が存在するためである.

iteration_end ludeme は勝利条件をイテレーション勝利条件を表す iteration_win ludeme, イテレーション敗北条件を表す iteration_lose ludeme を保持する. この２つは条件式を表す condition ludeme (後述) が与えられていて, もしプレイヤーが iteration_win ludeme の条件を満たすと, そのイテレーションで勝利し, その敵プレイヤーはそのイテレーションで敗北したと判定される. 逆に, iteration_lose ludeme の条件を満たすと, そのイテレーションで敗北し, その敵プレイヤーは勝利したと判定される.

loop_end ludeme は iteration_end ludeme と同様に, loop_win ludeme と loop_lose ludeme を保持し, ループの勝利条件, 敗北条件を表す. ループの勝利条件を満たしたプレイヤーが, そのゲームの最終的な勝者であると判定され, 敗北条件を満たすと敗者と判定される.

piece ludeme は条件式 condition ludeme と命令 command ludeme により構成される. 条件式を満たしたプレイヤーが存在すると, 命令が発動される. 例えば, 日本のドッジボールのルールには「内野のプレイヤーが敵のボールに当たったら, 外野へ行く」というルールがあるが, 「内野のプレイヤーが敵のボールに当たる」を条件式, 「外野へ行く」を命令として説明することができる.

3.2.3 Condition Ludeme

重要な ludeme 型として, condition ludeme がある. これは, 先述の通り, 条件を表す. condition ludeme は主語となる player ludeme, 条件式を表す equation[2] ludeme を保持している.

主語とは, どのような性質のプレイヤーが条件式の適用対象となるかを表す. 例えば「コート内側にいるプレイヤー」などがある.

[1] 「味方側」「敵側」の定義は**図 2** 参照

[2] 「Equation」とあるが, 不等式も含まれる.

equation ludeme には, 毎フレーム[1]真偽を判定される条件式が記述される. 数値も ludeme として定義されているため, ludeme の入れ子構造として条件式が表現されるが, LISP 型言語による数式の記述のようだと類推できる. また, プレイヤーから見た「自分の点数」や「自分が飛ばしたボールのバウンドした位置」など, プレイヤーとの文脈上の変数も用意している.

例えば, 次はテニスの「4 ポイント以上かつ相手より 2 点以上差をつけるとそのゲームで勝利する」という条件は, 次のように表される.

```
(and
    (>= my_point 4)
    (>= (- my_point enemy_point) 2)
)
```

and が equation ludeme の一種として処理されていて, パラメーターludeme として二つの equation ludeme を保持している. 二つとも equation 型のうち”>=”の値の, LISP と同様の不等式を表す.

4. 遺伝的プログラミングによる合成

前章で表現されたゲームルールを, 遺伝的プログラミングにより合成する. 現実に存在する球技ゲームのうち, 制約(3.1.1) – (3.1.6)を満たす球技ゲーム 9 点[2]を種（遺伝的プログラミングにおいて最初にプールに登録されるゲームルール）として, 次に述べる Synthesizer クラスにより 10 世代に渡り合成された.

4.1 Synthesizer

Synthesizer クラスは, 遺伝的プログラミングによる合成を進行させるプログラムである. これは C#により構成されている.

最初に種ゲームと, 後述する Mater クラスと Mutater クラスを登録する. そして合成する世代数を指定することにより, 1 世代当たり与えられた 20 個のゲーム表現を生成することを, 指定された世代に到達するまで繰り返す.

1 つのゲームを生成する手順は次の通りである：

(4.1.1) テンプレートとなるゲームルール A をプールから選択する.
選択する方法は, 乱数によって決定される. 各ゲームが選出される確率は, ゲームiのスコア(後述)をs_iとして,

$$\frac{s_i}{\sum_k s_k}$$

としている.
また, 選択対象は, n 世代目のゲームルール生成時は n-1 世代目までのゲームの中から 1 つである. すなわち, 生成するゲームルールと同じ世代のゲームルールをテンプレートとして選択されることはない.

(4.1.2) Mater クラスにゲームルール A を提出し, Mater クラスから交叉した結果のゲームルール B と, Mater クラス内での合成履歴を受け取る.
合成履歴には, 交叉が発生した ludeme の型名と場所を表す[3]ludeme, どのゲームから交叉を得たのかが記録される.

(4.1.3) Mutater クラスにゲームルール B を提出し, Mutater クラスから変異した結果のゲームルール C と, Mutater クラス内での合成履歴を受け取る.
合成履歴には, 交叉が発生した ludeme の型名と, その場所を表す構造 ludeme の型名が記録される.

(4.1.4) ゲームルール C をスコア評価する.
本稿では, このスコアはゲームルール A のスコアの 0.8 倍と定義している. これは, 改変する度にルールの妥当性[4]が失われていく可能性が高まっているので, 妥当性の高いものが操作(4.1.1)にて選択されやすくするための操作である.

(4.1.5) ゲームルール C をプールに保存する.
ゲームルール C とともに, 変更履歴, スコアが紐づけられて保存される.

4.2 Mater

Mater クラスは, Synthesizer から渡されたゲームルールを, 他のゲームと交叉させる役割を持つ.

各 ludeme が交叉する可能性を持っている. その操作工程は次のようになる.

(4.2.1) 対象が構造 ludeme の場合, 10%の確率で他ゲームルールの同一構造 ludeme のコピーと入れ替わる.
非構造 ludeme の場合, 他ゲームルールの同じ型の ludeme をランダムに選び, そのコピーと入れ替わる.

(4.2.2) 1.で変異が発生しなかった場合, そのパラメーターludeme に関して 1.の操作に戻る. パラメータ—ludeme が存在しない場合は終了する.

(4.2.3) 複数可の ludeme に関しては, 10%の確率で他ゲームの同じ型の ludeme のコピーを追加する.

他ゲームルールの選択は, 前世代までのゲームルールからランダムに選択される.

ludeme の入れ替わり, すなわち変異が発生したときは, 先述の通り入れ替わった ludeme と, その場所を表す構造 ludeme が合成履歴として記録される.

4.3 Mutater

Mutater クラスは, Synthesizer から渡されたゲームルールを部分的に変異させる役割を持つ.

構造 ludeme 自体は変異が行われない. 各非構造 ludeme のパラメーターludeme が変異の対象となる. 非構造 ludeme は 10%の確率で変異される. 変異が決定された際の処理は次の通りである：

[1] フレームとはゲームエンジンにおいて各演算を行う周期のことである.
[2] 具体的には, バトミントン, テニス, 卓球, シングルバレーボール, deck tennis, headis, padel tennis, pickleball, speedminton を使用した.
[3] 図 1 上の長方形として表される ludeme を指す. この ludeme を知ることで, ゲームルールのうちどの部分が変更を受けたかを知ることができる.
[4] 「妥当性」に関しては 6 章参照のこと.

(4.3.1) 該当する ludeme 型の値が列挙型[1]の場合は, 他の列挙子から, パラメーターludeme が存在しないか int ludeme を保持するものにランダムに変更する. 変更した際に新たなパラメーターint ludeme が必要になった場合は, (4.3.2)の int ludeme 生成を行う.

(4.3.2) 該当する ludeme の型が int ludeme の場合, 10%の確率で新しい int ludeme を生成する. 50%の確率で int ludeme, 20%の確率で整数同士の足し算引き算, 30%の確率でプレイヤー文脈の整数[2]がランダムに生成される.

(4.3.3) (4.3.2)の変異が行われなかった場合の int ludeme は等確率で+1 あるいは-1 加算される.

(4.3.4) float ludeme は等確率で+0.5 あるいは-0.5 加算される.

また, 複数可の ludeme は, 10%の確率で一つ削除する. ただし, この削除により最低限必要な構造 ludeme が消失する場合（member ludeme が一つもなくなるなど）は行われない.[3]

4.4 結果

以上のゲームルール合成を 10 回行い、第 10 世代のゲーム GDL を計 200 点得られた. 生成物には多様性が見られた. 現在のところ, 本研究の主目的は様々な球技ゲームを自動生成するところにあり, 各ゲームの面白さを客観的な指標を用いて評価する研究はこれからである. そこで, 今回は, 主観的ではあるが, 面白さが見込めそうな構造 ludeme の生成物, 不良といえる生成物をそれぞれ 2 点ずつ紹介する.

4.4.1 面白そうな ludeme の例 1

```
(iteration_win
   (condition
      all
      (at my_ball_bounced_last enemy_inside_back)
   )
)
(iteration_lose
   (condition
      all
      (> my_ball_bounce_n 3)
   )
)
```

上記は
勝利条件：自分の飛ばしたボールがコート敵側内側の後方にバウンドする.
敗北条件：自分が飛ばしたボールが, 3 回以上バウンドする.
というルールを示している.

自分の飛ばしたボールが該当範囲に当たるように飛ばし、逆に飛ばされた敵のボールが該当範囲に当たるか見極めないといけないゲームとなっている.

4.4.2 面白そうな ludeme の例 2

```
(outside
   (square 2.5000 2.5000 0.0000)
   True
)
(iteration_win
   (condition
      all
      (or
         (>= my_ball_bounce_n 3)
         (>= my_ball_bounce_ground_n 2)
      )
   )
)
```

上記は, 外側コートが 2.5m ×2.5m の正方形で, 壁がある中, 自分が飛ばしたボールが壁含め 3 回以上バウンドするか, 地面を 2 回バウンドすると勝利できるという内容である.

狭いコートの中で頻繁にバウンドするボールを追うとてもスピーディーなゲームと予想できる。

4.4.3 不良な例

```
(member
   1
   my_inside_back
   my_side
   bounce, catch
)
```

上記はボールを撃ち返すかキャッチが許可されているプレイヤーを表すが, 保持したボールを操作するスキルがないため, キャッチするとゲームが進行不可能になる.

4.4.4 不良な例

```
(loop_win
   (condition
      all
      (>= my_ball_bounce_ground_n 20)
   )
)
```

上記は, ループ（ゲーム全体）の勝利条件が自分のボールが 20 回以上バウンドすることになっている. すなわち, これが満たされるまでゲームが永続する. このようなゲームも現実的ではないものと想像できる.

5. ゲームエンジンによる再現

4.で紹介した遺伝的プログラミングの実装においては, 生成されたゲームルールの評価に関しては不完全である. 理想として, 「面白い」ゲームが高スコアで評価され, より選ばれやすくする仕組みが必要となる. その評価を実施するためには, 実際にそのルールのゲームを再現し, プレイヤーの動作をシミュレーションする必要がある. その第一段階として, ゲームエンジンによるゲームの再現を行った.

3.で述べた GDL で表現されたゲームルールを, ゲームエンジン Unity によって再現する基盤を開発した. この基盤の処理の流れは次の通りである：

[1] 列挙型・列挙子は, C#の機能の enum に相当する. 固定された一連の定数を表すために使用されるデータ型であり, プログラム内で値の選択肢を明示的に定義するために利用される.
[2] 3.2.3 で述べたプレイヤー文脈の変数のことである.
[3] Mater で増えた分を相殺する狙いで実装されている.

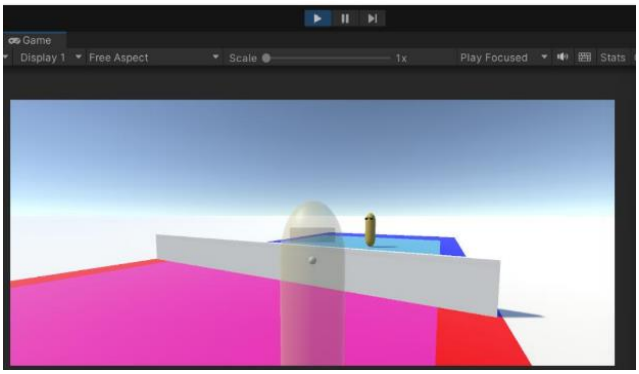

図 3 テニス GDL を再現した Unity 画面（ゲームビュー）．ゲームビューとして, Unity 操作者が操作できるプレイヤーの視点を表示している．

(5.1) GDL の読み込み

GDL をコンパイルして C#のデータに変換する．これは Synthesizer の GDL 読み取り器を流用している．

(5.2) オブジェクト配置

データ変換された各構造 ludeme を対応する Allocater クラスに渡される．Allocater クラスにより，プレイヤー・コート・ボールのオブジェクトは ludeme に指定された形式で配置され，進行ルールとして後述する Observer オブジェクトが配置される．

(5.3) Observer によるゲーム進行監視

Observer オブジェクトによって進行ルールが再現されている．各条件（イテレーション勝利・敗北条件，ループ勝利・敗北条件，Piece 発動条件）ごとに Observer オブジェクト[1]が設置される．Observer は条件式を保持する．

各 Observer は，毎フレーム，全プレイヤーに対して，そのプレイヤーが条件式を満たしたかを確認する．もし満たされた場合，その Observer が対応する条件をプレイヤーが満たしたとみなし，ゲーム進行処理がなされる．

図 3 はデモンストレーションとしてテニス GDL をコンパイルしたゲーム画面である．まだプレイ自動化がされていないため，実験者がプレイヤーを操作できるようにしている．操作方法はマウス移動で視線を回転でき，キー入力で移動や各スキルを発生させることができる．

図 4 はテニス GDL において，実験者がプレイヤーを操作して，敵側内側コートにボールを 2 回バウンドさせた．コンソールログによりイテレーション勝利条件を満たしたことが通知され，Observer が機能していることが確認できる．

[1] Observer オブジェクトは Observer クラスのインスタンスを指していて，ゲーム空間上に物理的なオブジェクトを配置したということではない．

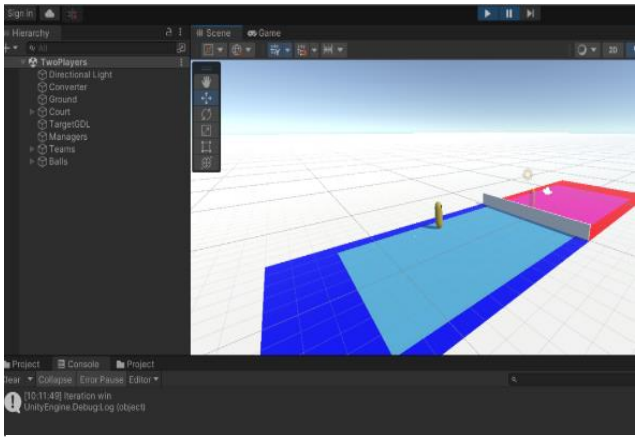

図 4 テニス GDL を再現した Unity 画面（シーンビュー）．自由に視点を操作できる．実験者がボールを敵側内側コートに 2 回バウンドさせたことにより，画面下部に Unity がイテレーション勝利条件を認知したことを通知するコンソールログを出力していることを確認できる．

6. おわりに

球技ゲームスポーツを LISP 型の木構造 GDL で表現し，遺伝的プログラミングを適用することで，多様なゲームルール表現を得ることに成功した．また，ゲームエンジンにより GDL から動的にゲームを生成する基盤を開発した．

ただし，4 章で見たように，不良なゲーム，すなわち進行不可能となかったり，ゲーム終了が現実的でなかったりといった，妥当性のないゲームルールも創成された．中には「面白くない」ゲームもあるであろう．今後の課題として，静的解析や自動テストプレイによって，創発されたゲームルールの妥当性・面白さを評価し，適切なスコアをつけることを目指す．これにより，遺伝的プログラミングによる「面白い」ゲームの創発能力を向上させることが期待される．

参考文献

[1] 三宅陽一郎，“ディジタルゲームにおける人工知能技術の応用，” *人工知能学会誌,* 第 巻 23, 第 1, pp. 44-51, 2008.

[2] V. B. Mark Riedl, “Interactive Narrative: A Novel Application of Artificial Intelligence for Computer Games,” *AAAI Conference on Artificial Intelligence,* 第 巻 26, 第 1, pp. 2160-2165, 2012.

[3] P. L. Lanzi , D. Loiacono, “ChatGPT and Other Large Language Models as Evolutionary Engines for Online Interactive Collaborative Game Design,” arxiv, 2023.

[4] C. Browne , F. D. Marie, “Evolutionary Game Design,” *IEEE Transactions on Computational Intelligence and AI in Games,* 第 巻 2, 第 1, pp. 1-16, 2010.

[5] A. Borvo, Anatomie d'n jeu de cartes: L'aluette ou jeu de la vache, Paris: Librairie Nantaise Yves Vachon, 1977.