Vol.2020-GN-109 No.9
Vol.2020-CDS-27 No.9
Vol.2020-DCC-24 No.9
2020/1/23

落ちものパズルゲーム共通ルール記述言語を用いたゲームルールの自動生成

栗原 一浩[1,a)]　阿部 雅樹[2]　渡辺 大地[2]

概要：ゲームの自動生成技術は、主にマップやダンジョンを対象とした生成を行っているが、ゲームのルール自体を自動生成した生成実例は少ない。GDL(Game Description Language) はゲームルールを体系化し、簡易的にゲームを形作るための記述言語である。 GDL を用いることで AI によるゲームルールの自動生成や、ゲームの面白み測定という研究も存在するが、対象はボードゲームに留まっている。 そこで、我々はデジタルゲームの中から落ちものパズルゲームに着目した。落ちものパズルゲームはアクション性が高くゲームルールを体系化でき、GDL として記述できる可能性がある。本研究では落ちものパズルゲームの自動生成を目的とし、本稿では落ちものパズルゲームを GDL として記述できるシステムを開発した。その結果、本提案システムを用いて落ちものパズルゲームを幅広く実装できることが分かった。

キーワード：落ちものパズルゲーム, デジタルゲーム, パズルゲーム, 共通ルール記述言語

Automatic generation of game rules using Game Description Language for Falling block puzzle game

Abstract: Automatic game generation technology automatically generates maps and dungeons, however, there are few examples of generating game rules. GDL(Game Description Language) is a description language that is organized game rules for generating game easily. There are researches on AI uses GDL to automatically generate game rules and measure the fun of games, but the subject remains limited to board games. Therefore, we focused on Falling block puzzle games in digital games. The purpose of this research is to generate Falling block puzzle games automatically. In this paper, we developed a system that can describe Falling block puzzle games as GDL. As a result, We found that Falling block puzzle games can be widely implemented using the proposed system.

Keywords: Falling block puzzle games, Digital game, Puzzle Game, Game description language,

1. はじめに

ゲームにおける自動生成技術はマップやダンジョン、パズルの問題 [1]、ゲーム上における物語など多岐にわたり研究されてきた。音楽ゲームにおいても譜面と呼ばれる音楽のリズムに合わせてプレイヤーに対して何らかのアクションを要求する遊びの部分を自動生成する研究などが存在する [2]。このようにゲームにおける自動生成はユーザーに対してゲーム内コンテンツを提供し続けることができる。

実際のゲームにおいてダンジョンを自動生成する不思議のダンジョンシリーズ [3] や、ファンタシースターオンライン 2[4] などが存在する。物語の自動生成を行ったゲームとしてはティル・ナ・ノーグ [5] などの事例も存在する。

ゲームにおける自動生成で難しい点は、ゲームルールが成立しているかどうかを考慮しなければならないことである。特にマップやダンジョンの自動生成においてはランダムな要素を用いることが多いが、必ずスタートからゴールへ向けてゲームがクリアできるように生成されなければならない。数独においては生成された問題が解を持つように生成しなければならない。このようにゲームルールをすべて考慮したうえでのコンテンツ自動生成はかなり困難であ

[1] 東京工科大学大学院バイオ・情報メディア研究科
Graduate School of Bionics, Computer and Media Sciences, Tokyo University of Technology
[2] 東京工科大学メディア学部
School of Media Science, Tokyo University of Technology
[a)] G31180086c@edu.teu.ac.jp

ⓒ 2020 Information Processing Society of Japan

Vol.2020-GN-109 No.9
Vol.2020-CDS-27 No.9
Vol.2020-DCC-24 No.9
2020/1/23

るが、研究や作品事例は多数確認できる。

しかし、ゲームのルール自体を自動生成した事例は少なく研究段階であり、本研究ではそこに着目した。Cameron Browne ら [6] はボードゲームのルールを組み合わせ、新たなゲームルールを生成した。さらに生成されたゲームを AI によって評価をしている。この研究では”Game Description Language”（GDL）[7] という手法によってルールの体系化を行っている。

GDL とは、General Game Playing と呼ばれるゲーム AI 研究分野の中で生み出されたものである。General Game Playing は初見のゲームであったとしても、ゲームルールさえ与えられればうまくプレイできるような AI を目指した研究分野である。AI は未知のゲームルールを与えられた場合においても、自身の行動が良いものか判断する必要がある。その場合、与えるゲームルール自体を体系化した要素で構築する必要があり、そのゲームルール生成手段として GDL が開発された。GDL で再現できるゲームは、ボードゲームや Atari ゲームに多く見られた単純なアクションゲームが多い [8]。以上で述べた研究は古典的なゲームの範疇に収まるものである。

ゲームルール自体を組み合わせて作成する事例は極めて少ないが、一例として、2019 年 12 月にリリースされた「SuperMash」[9] というゲームがある。これはゲームのジャンルを 2 つ選択すると選択したジャンルを混ぜ合わせたゲームが作成できるものである。しかし、一定のルールを組み合わせた際にゲームが進行できない場合も存在し、ゲームをクリアできるように考慮した場合、ゲームのルール自体を生成するのは極めて困難であることがわかる。

そこで、我々はルールの共通性があり、体系化が行いやすい落ちものパズルゲームに着目した。落ちものパズルゲームは「アクションパズルゲーム」とも言われ、アクション性を伴うものがほとんどである。一般的に知られている落ちものパズルゲームを挙げると、ぷよぷよ [10] やテトリス [11] などがある。本研究は、落ちものパズルゲームのルール自体の自動生成を目的とし、落ちものパズルゲームにおける共通ルール記述言語を提案する。提案した落ちものパズルゲーム共通ルール記述言語を用いることにより、落ちものパズルゲームの体系的な自動生成を可能とした。

なお、本研究における「言語」とは、プログラム言語のような文法のことを指す意味ではなく、ゲームにおけるルールや決まりを列挙できるようにしたものを言語と呼ぶこととする。

落ちものパズルゲームのルールを体系化した先行事例として、「電撃コンストラクション 落ちゲーやろうぜ！」「落ちゲーデザイナー作ってポン」が存在する。それまでには存在しない落ちものパズルゲームのルールをソフト内で生成、プレイ可能なゲームである。しかし、これらのゲームタイトルでは GDL としての体系化はしておらず、ゲーム制作ツールとしての形態をとる。また、ルール体系化の範囲が限られているため、代表的な落ちものパズルゲームの一部が再現できないという問題点がある。例を挙げるとぷよぷよは実現することが可能であるが、テトリスの実現は不可能である。実際にぷよぷよとテトリスを同一体系、システム上で実現できた例はなく、理由としてテトリスのルールが他の落ちものパズルゲームよりも特殊なルールをしているためであると考えられる。その点から、落ちものパズルゲームの代表例であるテトリスとぷよぷよを同一体系上で実現することが極めて困難であることが確認できる。

本提案落ちものパズルゲーム共通ルール記述言語は、先行事例であるゲームソフトでは実現不可能であったゲームルールの実現も可能とした。本稿では Lua を用いて落ちものパズルゲームのルールを記述するよう開発し、これを Falling Puzzle Game Description Language(FPGDL) と呼称する。ルールの記述部以外の落ちものパズルゲームにおける描画や操作全般の処理は C++/Dxlib を用いて開発を行い、これを Falling Puzzle Process System(FPPS) と呼称する。特に、次のような点が大きな開発理念として存在する。

(1) 落ちものパズルゲームルールの体系的な実装。
(2) 先行事例で実現不可だったルールが実現可能。
(3) 複雑な消滅条件の簡易化。
(4) 複雑な落下処理の簡易化。
(5) ツモ形状と回転法則の柔軟性。
(6) ゲームルール体系的な自動生成。

本研究では FPGDL において、一人でプレイする場合を想定しており、対戦プレイは想定していない。

2. 落ちものパズルゲームの定義

落ちものパズルゲームとは、プレイエリア内で一定の条件を満たすようにブロックを操作し、ブロックを消滅・変化させ得点を獲得していくというアクションパズルゲームのことである。プレイエリアを四角形の格子で区切った一つ一つをマスと呼び、全体をフィールドと呼ぶ。フィールド上部から下部に向けてブロックは自然に降下し、プレイヤーは降下中に左右移動やブロックの回転操作、意図的な降下加速を行う。降下中のブロックを特にツモと呼び、ツモが下部に達したら操作不能となり、フィールド上のブロックとして積み重なっていく。ツモがフィールド上のブロックに成ることを、ブロックを配置すると呼称する。操作中のツモが下部に達したら、次のツモが上から降ってくる。プレイヤーはツモの操作を繰り返しブロックを配置し、フィールド上のブロックの配置が一定の条件を満たすことでブロックは消滅する。ブロックは複数種類存在し、それぞれ異なる色が割り当てられる。

配置した後のブロックはそれぞれ下記のような落下法則が存在する。

ⓒ 2020 Information Processing Society of Japan

Vol.2020-GN-109 No.9
Vol.2020-CDS-27 No.9
Vol.2020-DCC-24 No.9
2020/1/23

- 落下は下方向にしか行われず、斜め下方向へ落下することはない
- ブロック同士が接続関係を持たない場合、ブロック下部のマスが空いていれば落下する
- 横方向のブロック同士で接続関係を持った場合、接続関係を持つすべてのブロック下部のマスが空いていない限り落下しない
- ある特定の条件下のみで落下する

落ちものパズルゲームにおける回転とは、事前に用意した形状を順番に切り替える動作となる。ツモには複数の回転形状と呼ばれるブロック配列が用意されている。回転を実行するとツモは次の回転形状であるブロックの配列に置き換わる。回転の切り替えの順番を逆順にすることで、逆回転動作をとることも可能である。

落ちものパズルゲームから派生したアクションパズルゲームもいくつか存在するが、本研究では定義をした落ちものパズルゲーム以外は対象外とする。

3. 落ちものパズルゲームのルール体系化

落ちものパズルゲームを GDL として記述できるようにするためには、落ちものパズルゲームのゲームルールの体系化をする必要がある。

落ちものパズルゲームにはゲームを構成する必須のルールとして以下のものが存在する。本研究では、これらの要素を踏まえて FPGDL を開発した。

- ブロック
- ツモ
- ブロックの消滅条件・消滅方法
- ブロックの落下
- ゲームオーバー

ブロック:

各ブロックには基本的に”色”が振られており、色ごとで消滅条件と呼ばれるブロックが消滅するために必須とする条件が存在する。ルールによってブロックの種類や色数も異なる。

FPGDL においてブロックの定義は以下のように記述する。

```
1 --4種類のブロックを作成
2 createBlockPattern( 1 )
3 createBlockPattern( 2 )
4 createBlockPattern( 3 )
5 createBlockPattern( 4 )
```

ツモ:

ツモとは、ブロックが複数個集まったものであり、フィールド上部から出現する。例としてぷよぷよを挙げると、ツモの形状おいてはツモ 2 つ組のブロックで成り立っている。ツモの回転形状は、一つのブロックを軸として 90°ずつツモの形状を回転したようなものである。図 1 はぷよぷよの回転形状の例を示した図である。

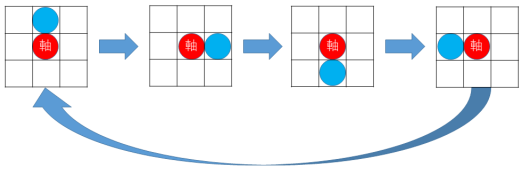

図 1 ぷよぷよの回転形状例

ブロックを軸とした回転動作の他に、マスとマスの間を軸にして回転動作を行うものもある。

図 2 はマスとマスの間を軸として回転をするぱにっくボンバーの回転形状である。

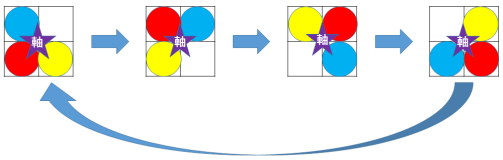

図 2 ぱにっくボンバーの回転形状例

しかし、テトリスにおけるツモの形状は一定ではない。テトリスはツモの形状が 4 つのブロックで成り立っており、7 種類のツモ形状が設定されている。ツモを 90°ずつ回転を行う事を想定した場合、その他のゲームでは通常 4 つの回転形状パターンを持つことになるが、テトリスに関しては 4 つの回転形状構成と 2 つの回転形状構成の 2 パターンが存在する。図 3 はテトリスの Z 型のツモ形状の回転形状において 4 パターンのものと 2 パターンのものを示した図である。

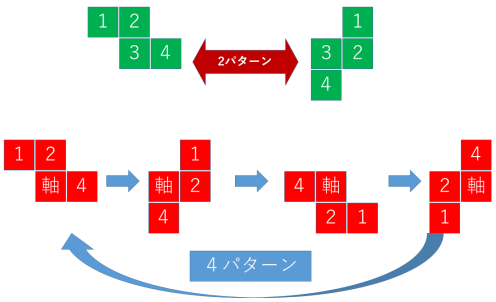

図 3 テトリスの回転形状例

また、回転動作の一種ではあるが特殊な例として、ツモのブロック同士が入れ替わるような動きをするものが存在する。図 4 はブロック同士が入れ替わるような回転動作をするものの例である。

ⓒ 2020 Information Processing Society of Japan

Vol.2020-GN-109 No.9
Vol.2020-CDS-27 No.9
Vol.2020-DCC-24 No.9
2020/1/23

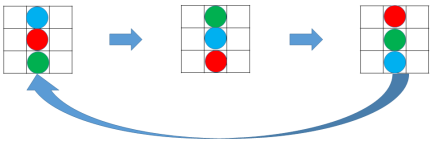

図 4　コラムス（ブロック同士が入れ替わる動作）の回転形状例

FPGDL においてツモの定義は以下のように記述する。以下のサンプルコードはテトリスの T 字型のツモを記述したものである。

```
--ツモの設定
--T=======================================
arrays[0] = { 0 , 0 , 0 , 0 }
arrays[1] = { 0 , -1 , 0 , 0 }
arrays[2] = { -1 , -1 , -1 , 0 }
arrays[3] = { 0 , 0 , 0 , 0 }
setTumo( arrays , 0 , 0 )--回転形状 1

arrays[0] = { 0 , 0 , 0 , 0 }
arrays[1] = { 0 , -1 , 0 , 0 }
arrays[2] = { 0 , -1 , -1 , 0 }
arrays[3] = { 0 , -1 , 0 , 0 }
setTumo( arrays , 0 , 1 )--回転形状 2

arrays[0] = { 0 , 0 , 0 , 0 }
arrays[1] = { 0 , 0 , 0 , 0 }
arrays[2] = { -1 , -1 , -1 , 0 }
arrays[3] = { 0 , -1 , 0 , 0 }
setTumo( arrays , 0 , 2 )--回転形状 3

arrays[0] = { 0 , 0 , 0 , 0 }
arrays[1] = { 0 , -1 , 0 , 0 }
arrays[2] = { -1 , -1 , 0 , 0 }
arrays[3] = { 0 , -1 , 0 , 0 }
setTumo( arrays , 0 , 3 )--回転形状 4
```

ブロックの消滅条件・消滅方法:

ブロックの消滅条件は色ごとで消滅判定をとる場合が多く、ぷよぷよを例に出すと全ブロックが共通して 4 つ同色のブロックが隣接していたら消滅するという消滅条件を持っている。また、テトリスなどブロックの色が 1 種類しか存在しない場合、ある形状を形成することで消滅条件が成立するものが多い。

ブロックの消滅方法は主に消滅条件が成立したら即座に消える場合が多いが、副条件と呼ばれる消滅条件とは違うもう一つの条件が設定されている場合がある。副条件を持つゲームに関してはルミネスなどが該当する。

FPGDL において消滅条件は以下のように記述する。以下のサンプルコードはぷよぷよの消滅条件を記述したものである。

```
y = getFieldHeight()
for p = 0 , getFieldHeight() do
   y = y - 1
   --4つ以上くっついていたら消える
   for x = 0 , getFieldWidth() do
      --4つ以上連結しているなおかつ落下中のブロックがいない時消滅判定を発生させる
      if getComboBlockNum( x , y ) >= 4 and
           getFieldBlock( x , y ) < 100 and
           getAllBlockDownFlag() == false then
        --ブロックを消滅させる
         setBlockDeleteFlag( x , y )
      end
   end
end
```

ブロックの落下:

フィールド上に配置したブロックは、ルール毎に落下法則の相違がいくつか存在する。したがって、様々な落下法則へ対応するため、本手法においては配置したブロック毎に任意のタイミングで落下するマス数を与えるようにすることで対応した。

FPGDL において消滅条件は以下のように記述する。以下のサンプルコードはぷよぷよの落下法則を記述したものである。

```
y = getFieldHeight()
for p = 0 , getFieldHeight() do
   for x = 0 , getFieldWidth() do
      --常に落下をさせる
      setBlockDownCellNum( x , y , getFieldHeight() )
   end
end
```

以下のサンプルコードはテトリスの落下法則を記述したものである。テトリスの落下法則はブロックの消滅時に影響するため、サンプルコード内には消滅時のコードも記載されている。

```
--落下させる量を保持する変数
downnum = 0
y = getFieldHeight()
for p = 0 , getFieldHeight() do
   y = y - 1
   checknum = 0
   --横列にブロックが何個いるか捜査
   for x = 0 , getFieldWidth() do
      if getFieldBlock( x , y ) > 0 and
           getFieldBlock( x , y ) < 100 then
         checknum = checknum + 1
      end
   end
   --横列にブロックが 10個以上いた場合消去処理をする
   if checknum >= 10 then
      checknum = 10
      downnum = downnum + 1
      for x = 0 , getFieldWidth() do
         setBlockDeleteFlag( x , y )
      end
   else
      --消去された横列より上にあるブロックに落下量を与える
      for x = 0 , getFieldWidth() do
```

ⓒ 2020 Information Processing Society of Japan

Vol.2020-GN-109 No.9
Vol.2020-CDS-27 No.9
Vol.2020-DCC-24 No.9
2020/1/23

```
23          setBlockDownCellNum( x , y , downnum )
24        end
25      end
26    end
```

ゲームオーバー:

ゲームオーバー条件は主に 3 つ存在し、1 つ目は新たなツモがフィールドに出現できないこと、2 つ目はブロックがフィールド最上部に設置されること、3 つ目はブロックがフィールド最上部に設置された後に一定時間経過することである。ツモは通常フィールドに出現する位置が固定されており、その出現位置に設置されたブロックが干渉してしまうと、新たなツモは出現できなくなり、ゲームオーバーとなる。この 1 つ目のゲームオーバー条件を採用するゲームは数多くある。2 つ目のゲームオーバー条件として、ブロックがフィールドの最上部に設置された場合、設置されたブロックがツモに干渉していなかったとしてもゲームオーバーとなる。3 つ目のゲームオーバー条件として、ブロックがフィールド最上部に設置されており、尚且つツモがフィールドに出現できる場合時間経過によってゲームオーバーとなる。一定時間の間に最上部に設置されたブロックを消滅させることができれば、そのままプレイを続行することができる。

4. FPGDL

FPGDL は、FPPS から提供した関数 (メソッド) を、Lua 言語上で用いることで実現している。

FPPS から提供した関数は以下のようなものがある。

- 初期設定系メソッド:ブロックの種類や形状の定義など
- update メソッド:ゲーム内情報の更新
- Get 系メソッド:ブロックの状態取得など
- Set 系メソッド:ブロックに対して消滅など命令を送るもの

FPGDL は大きく 2 つのフェーズがある。一つは初期設定フェーズ、もう一つはアップデートフェーズである。

初期設定フェーズでは、主に落ちものパズルゲームに用いるブロックの定義やツモの種類や形状の定義などを行う。フィールドの大きさ、ハードドロップと呼ばれるブロックの即設置の使用有無なども初期設定フェーズに記述をする。

アップデートフェーズは落ちものパズルゲームのルールを記述していく。落ちものパズルゲームの消滅条件や落下法則などのルールは FPPS から提供した update メソッド内に記述する必要がある。update メソッドはゲームの画面が 1 回更新される度に実行される。図 5 は FPGDL のフェーズを図解したものである。

記述する流れとしては、まず初期設定フェーズにブロックの種類やフィールドの大きさ、ツモの設定する。また、ハードドロップと呼ばれる即設置行動の使用可否、シャドウと呼ばれるブロックの落下予測位置の表示の有無などは初期設定フェーズにて記述をする。次にアップデートフェーズにおいて update メソッドを記述し、update メソッド内に主に消滅条件や落下法則などの条件を Set 系や Get 系メソッドを用いて記述をする。

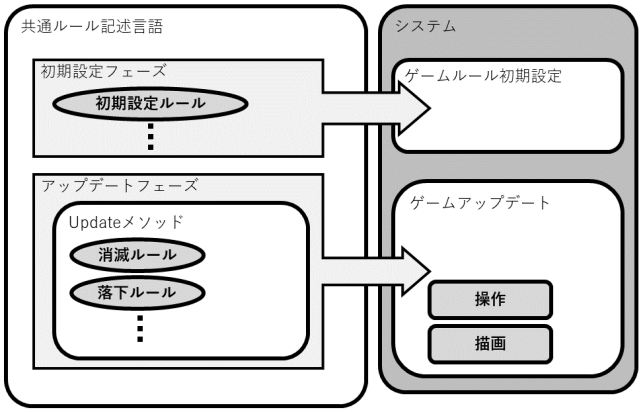

図 5 FPGDL と FPPS の概要

5. 検証

提案手法の検証方法として、代表的な落ちものパズルゲームである「ぷよぷよ」と「テトリス」のルールが実現できるかどうかを検証した。さらに、体系的に自動生成が可能であることを示すために各落ちものパズルゲームのルールをそれぞれ定義し、混合したルールの実現と、ルール同士をでたらめに組み合わせて実現ができるかどうかを検証した。

5.1 ぷよぷよのルール実現

図 6 は代表的な落ちものパズルゲームのルールであるぷよぷよの実行画面である

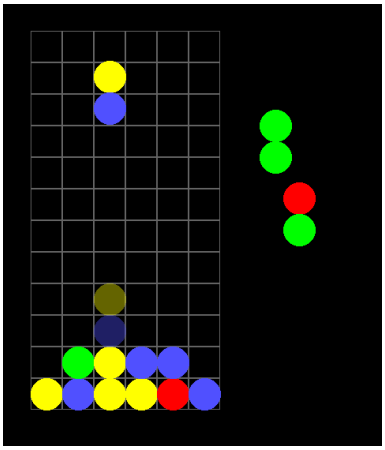

図 6 ぷよぷよの実行画面

ぷよぷよのルールは先行事例においても実現が可能であり、本手法を用いた場合でも問題なく実現することができた。以下のサンプルコードは、本手法を用いてぷよぷよの

ⓒ 2020 Information Processing Society of Japan

Vol.2020-GN-109 No.9
Vol.2020-CDS-27 No.9
Vol.2020-DCC-24 No.9
2020/1/23

ルールを記述したものである。

```
setFieldSize ( 6 , 13 ) --パズルのプレイフィールドのサイズ指定
blockSize ( 30 )  --ブロックの描画サイズ指定
setSuperRotation( false ) --回転補正を無効にする
setQuickTurn( true ) --クイックターンを有効にする
setClearMode( -1 )  --消去ルールモードを大まかに指定(自らアルゴリズムを組む場合-値)
setHardDrop( false ) --ハードドロップを無効にする

--ブロックを作成
createBlockPattern( 1 )
createBlockPattern( 2 )
createBlockPattern( 3 )
createBlockPattern( 4 )
createBlockPattern( 5 )
--ブロックの形状を指定
setBlockForm( 1 , 1 )
setBlockForm( 2 , 1 )
setBlockForm( 3 , 1 )
setBlockForm( 4 , 1 )
setBlockForm( 5 , 1 )
--ブロックに色を指定
setBlockFormColor( 1 , 255 , 0 , 0 )
setBlockFormColor( 2 , 0 , 255 , 0 )
setBlockFormColor( 3 , 80 , 80 , 255 )
setBlockFormColor( 4 , 180 , 0 , 255 )
setBlockFormColor( 5 , 255 , 255 , 0 )

--多次元配列準備
arrays = {}
for p = 1 , 3 do
   arrays[p] = {}
end

--ツモの設定
arrays[0] = { 0 , 1 , 0 }
arrays[1] = { 0 , 2 , 0 }
arrays[2] = { 0 , 0 , 0 }
setTumo( arrays , 0 , 0 )--回転形状 1
arrays[0] = { 0 , 0 , 0 }
arrays[1] = { 0 , 2 , 1 }
arrays[2] = { 0 , 0 , 0 }
setTumo( arrays , 0 , 1 )--回転形状 2
arrays[0] = { 0 , 0 , 0 }
arrays[1] = { 0 , 2 , 0 }
arrays[2] = { 0 , 1 , 0 }
setTumo( arrays , 0 , 2 )--回転形状 3
arrays[0] = { 0 , 0 , 0 }
arrays[1] = { 1 , 2 , 0 }
arrays[2] = { 0 , 0 , 0 }
setTumo( arrays , 0 , 3 )--回転形状 4

function update()

   --落下させる量を保持する変数
   downnum = 0
   y = getFieldHeight()
   for p = 0 , getFieldHeight() do
      y = y - 1
      --4つ以上くっついていたら消える
      for x = 0 , getFieldWidth() do
         --常に落下をさせる
         setBlockDownCellNum( x , y , getFieldHeight() )
         --4つ以上連結しているなおかつ落下中のブロックがいない時消滅判定を発生させる
         if getComboBlockNum( x , y ) >= 4 and getFieldBlock( x , y ) < 100 and getAllBlockDownFlag() == false then
            setBlockDeleteFlag( x , y ) --ブロックを消滅させる
         end
      end
   end

end
```

5.2 テトリスのルール実現

次に、テトリスのルールを実現ができるか検証を行った。図 7 はテトリスのルールを実現した際の実行画面である。

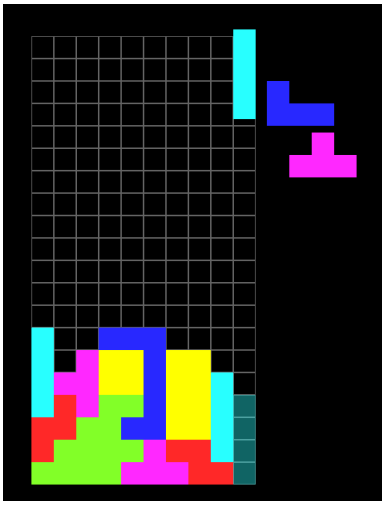

図 7 テトリスの実行画面

テトリスは先行事例において実現が不可能であったが、本手法を用いた場合テトリスも問題なく実現できることが確認できた。先行事例では複数のツモ形状を定義することが不可能であり、テトリスにおける複雑な落下法則の実現が不可能であったが本手法ではどちらも実現することができた。これにより、本手法によって実現できる落ちものパズルゲームの多様性を示すことができたと考えられる。

以下のサンプルコードは、本手法を用いてテトリスのルールを記述したものである。

```
setFieldSize ( 10 , 20 ) --パズルのプレイフィールドのサイズ指定
blockSize ( 30 )  --ブロックの描画サイズ指定
setSuperRotation( true ) --回転補正を有効にする
setQuickTurn( false ) --クイックターンを無効にする
setClearMode( -1 )  --消去ルールモードを大まかに指定(自らアルゴリズムを組む場合-値)
setHardDrop( true ) --ハードドロップを有効にする

--ブロックを作成
createBlockPattern( 1 )
```

ⓒ 2020 Information Processing Society of Japan

Vol.2020-GN-109 No.9
Vol.2020-CDS-27 No.9
Vol.2020-DCC-24 No.9
2020/1/23

```
createBlockPattern( 2 )
createBlockPattern( 3 )
createBlockPattern( 4 )
createBlockPattern( 5 )
createBlockPattern( 6 )
createBlockPattern( 7 )

--ブロックの形状を指定
setBlockForm( 1 , 3 )
setBlockForm( 2 , 3 )
setBlockForm( 3 , 3 )
setBlockForm( 4 , 3 )
setBlockForm( 5 , 3 )
setBlockForm( 6 , 3 )
setBlockForm( 7 , 3 )

--ブロックに色を指定
setBlockFormColor( 1 , 255 , 40 , 255 )
setBlockFormColor( 2 , 255 , 155 , 40 )
setBlockFormColor( 3 , 40 , 40 , 255 )
setBlockFormColor( 4 , 130 , 255 , 40 )
setBlockFormColor( 5 , 255 , 40 , 40 )
setBlockFormColor( 6 , 255 , 255 , 0 )
setBlockFormColor( 7 , 40 , 155 , 255 )

--多次元配列準備
arrays = {}
for p = 1 , 4 do
   arrays[p] = {}
end

--ツモの設定
--T=======================================
arrays[0] = { 0 , 0 , 0 , 0 }
arrays[1] = { 0 , -1 , 0 , 0 }
arrays[2] = { -1 , -1 , -1 , 0 }
arrays[3] = { 0 , 0 , 0 , 0 }
setTumo( arrays , 0 , 0 )--回転形状 1
arrays[0] = { 0 , 0 , 0 , 0 }
arrays[1] = { 0 , -1 , 0 , 0 }
arrays[2] = { 0 , -1 , -1 , 0 }
arrays[3] = { 0 , -1 , 0 , 0 }
setTumo( arrays , 0 , 1 )--回転形状 2
arrays[0] = { 0 , 0 , 0 , 0 }
arrays[1] = { 0 , 0 , 0 , 0 }
arrays[2] = { -1 , -1 , -1 , 0 }
arrays[3] = { 0 , -1 , 0 , 0 }
setTumo( arrays , 0 , 2 )--回転形状 3
arrays[0] = { 0 , 0 , 0 , 0 }
arrays[1] = { 0 , -1 , 0 , 0 }
arrays[2] = { -1 , -1 , 0 , 0 }
arrays[3] = { 0 , -1 , 0 , 0 }
setTumo( arrays , 0 , 3 )--回転形状 4

--以下テトリスにおける各形状のツモを定義するため省略--

function update()

   --落下させる量を保持する変数
   downnum = 0
   y = getFieldHeight()
   for p = 0 , getFieldHeight() do
      y = y - 1
      checknum = 0
      --横列にブロックが何個いるか捜査
      for x = 0 , getFieldWidth() do
         if getFieldBlock( x , y ) > 0 and
              getFieldBlock( x , y ) < 100 then
            checknum = checknum + 1
         end
      end
      --横列にブロックが 10個以上いた場合消去処理をする
      if checknum >= 10 then
         checknum = 10
         downnum = downnum + 1
         for x = 0 , getFieldWidth() do
            setBlockDeleteFlag( x , y )
         end
      else
         --消去された横列より上にあるブロックに落下量を与
              える
         for x = 0 , getFieldWidth() do
            setBlockDownCellNum( x , y , downnum )
         end
      end
   end

end
```

5.3 ぷよぷよ＋テトリスの混合ルール実現

次に、落ちものパズルゲームのルールを組み合わせた混合ルールが実現できるか検証した。図 8 はぷよぷよとテトリスの混合ルールを実行した画面である。

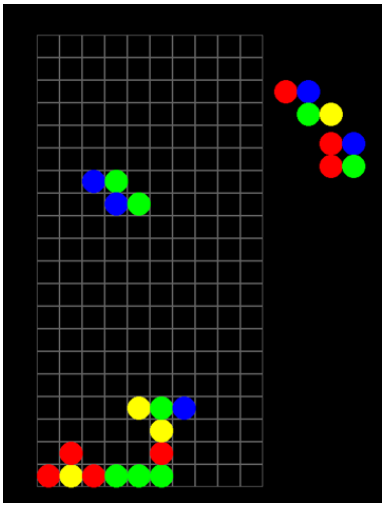

図 8　ぷよぷよとテトリスの混合ルールの実行画面

結果、複数のゲームルールを組み合わせても問題なく落ちものパズルゲームとしてプレイすることが可能であることがわかった。ぷよぷよとテトリスのルールを適用した部分は次のとおりである

- 回転法則：テトリス
- ツモ形状：テトリス
- ブロック：ぷよぷよ
- 落下法則：テトリス＋ぷよぷよ
- 消滅条件：テトリス＋ぷよぷよ

ⓒ 2020 Information Processing Society of Japan

- ゲームオーバー：両共通

5.4 でたらめに選出した複数ルールの混合ルール実現

最後に落ちものパズルゲームのルールをそれぞれ定義し、でたらめに組み合わせる検証を行った。検証方法としては、複数の落ちものパズルゲームのルールをそれぞれツモ、ブロック、消滅方法、落下法則に分けてルミネス、ぱにっくボンバー、テトリスのルールをそれぞれテンプレートとしてルールを定義した。各テンプレートの中からでたらめにルールを選出し実行を行った場合、落ちものパズルゲームとして問題なくプレイできるかを検証する。本稿では一例として以下の事例を検証結果として述べる。図 9 はルミネス、ぱにっくボンバー、テトリス 3 つのゲームルールを定義し、それぞれでたらめに適用するように実装し、実行した結果の画面となる。

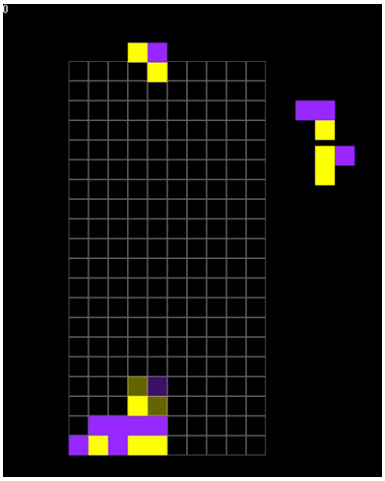

図 9 複数ルールをでたらめに適用したゲームの実行画面

結果、でたらめにルールを混合させたとしても落ちものパズルゲームとして成立しないことはなく、問題なくプレイが可能であることが分かった。ルールを適用した部分は次のとおりである

- 回転法則：ルミネス・ぱにっくボンバー共通
- ツモ形状：ぱにっくボンバー
- ブロック：ルミネス
- 落下法則：ルミネス・ぱにっくボンバー共通
- 消滅条件：ルミネス
- ゲームオーバー：全共通
- フィールドの大きさ：テトリス

今回の実行結果はテトリスの要素はフィールドの大きさしか選出されなかったが、テトリスのルール要素も混合させることができている。

6. まとめ

本研究では落ちものパズルゲームのルールを体系化し、体系化したルールを GDL として記述できるようにした。結果、先行事例において実現できなかった落ちものパズルゲームのルールや、複数のルールを定義しでたらめに選出した場合でも落ちものパズルゲームとしてルールが破綻しないものを出力することに成功した。

今後の展望として、出力されたゲームルールを AI によって面白みや難易度などを解析できれることができれば、より洗練されたものになるのではないかと考えた。

参考文献

[1] 一貴前田，博　奥乃：数独の問題作成支援システムの設計と開発，全国大会講演論文集，Vol. 70, pp. 799–800（オンライン），入手先 ⟨https://ci.nii.ac.jp/naid/110006867930/⟩ (2008).
[2] 俊宗香川，宏史手塚，真理稲葉：音楽の重要な構成要素の抽出の提案-音楽ゲーム用譜面自動生成のために-，エンタテインメントコンピューティングシンポジウム 2015 論文集，Vol. 2015, pp. 326–333 (2015).
[3] 不思議のダンジョン風来のシレン：不思議のダンジョン 風来のシレン — スパイク・チュンソフト，https://www.spike-chunsoft.co.jp/shiren_sp/. 参照:2019.11.13.
[4] ファンタシースターオンライン 2：『ファンタシースターオンライン 2』公式サイト | SEGA，http://pso2.jp/. 参照:2019.11.13.
[5] ティル・ナ・ノーグ：（PS2、PSP）ティル・ナ・ノーグ～悠久の仁～　オフィシャル WEB ページ，http://www.ss-alpha.co.jp/products/tirnanog_consumer.html. 参照:2019.11.13.
[6] Browne, C. and Maire, F.: Evolutionary Game Design, *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 2, No. 1, pp. 1–16 (2010).
[7] Love, N., Hinrichs, T. and Genesereth, M.: General Game Playing: Game Description Language Specification, *Stanford University, Tech.Rep* (LG-2006-11(2006)).
[8] Schaul, T.: A video game description language for model-based or interactive learning, *IEEE Conference on Computational Inteligence in Games (CIG)*, pp. 1–8 (2013).
[9] Super Mash: SuperMash for Nintendo Switch - Nintendo Game Details, https://www.nintendo.com/games/detail/supermash-switch/. 参照:2019.12.16.
[10] ぷよぷよ：ぷよぷよポータルサイト，http://puyo.sega.jp/portal/index.html. 参照:2018.12.18.
[11] テトリス：Tetris — The addictive puzzle game that started it all!, https://tetris.com/. 参照:2019.7.12.

ⓒ 2020 Information Processing Society of Japan