修士論文　　2019 年度

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自動生成に関する研究

東京工科大学大学院

バイオ・情報メディア研究科

メディアサイエンス専攻

栗原 一浩

修士論文　　2019 年度

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自動生成に関する研究

指導教員　渡辺 大地 准教授

東京工科大学大学院

バイオ・情報メディア研究科

メディアサイエンス専攻

栗原 一浩

論 文 の 要 旨

論文題目	落ちものパズルゲーム共通ルール記述言語を用いた ゲームルール自動生成に関する研究
執筆者氏名	栗原 一浩
指導教員	渡辺 大地 准教授
キーワード	落ちものパズルゲーム、パズルゲーム、デジタルゲーム、 ゲーム共通ルール記述言語、自動生成

[**要旨**]

ゲームにおける自動生成技術はマップやダンジョン、パズルの問題やゲーム上におけるシナリオなど多岐にわたり研究されている。ゲームにおける自動生成で困難な点は、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。特に一部ゲームに用いられる自動生成はランダムな要素を用いることが多いが、ランダムな要素を用いたうえ必ずゲームがクリアできるようにしなければならない。例として数独を挙げると、自動的に生成された問題が解を持つようになっていなければ数独として成立しない。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るが、研究や作品事例は存在する。

しかし、ゲームルール自体を自動生成した事例は少なく研究段階である。共通ルール記述言語を用いた 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をする研究も存在する。しかし対象はボードゲームやパズルゲームに留まるものであり、アクション要素のあるゲームに関してはあまり研究されていない。そこで、本研究ではゲームルールを自動生成するうえで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。本研究は、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。本提案言語を用いることにより、先行手法では実現できなかったルールの実装ができた。さらに落ちものパズルゲームのルールを複数定義し、様々なゲームルールの生成が可能となった。

A b s t r a c t

Title	Automatic generation of game rules using Game Description Language for Falling block puzzle games
Author	Kazuhiro Kurihara
Advisor	Taichi Watanabe
Key Words	Falling block puzzle games, Puzzle games, Digital games, GameDescriptionLanguage, Automatic generation

[**summary**]

The automatic generation technology in games has been studied in various fields such as maps, dungeons, puzzle problems, and game scenarios. The difficulty in automatically generating a game is that it is necessary to consider whether game rules are established. In particular, the automatic generation used in some games often uses random elements, but it is necessary to use random elements and ensure that the game can be cleared. Taking Sudoku as an example, if an automatically generated problem does not have a solution, it will not hold as Sudoku. As described above, it is quite difficult to automatically generate content in consideration of all game rules, but there are studies and examples of works.

However, there are few cases where the game rules themselves were automatically generated, and this is still in the research stage. There are also studies on automatic generation of game rules by AI using a common rule description language and measurement of game fun. However, the target is limited to board games and puzzle games, and games with action elements have not been studied much. Therefore, in this research, we focused on a Falling Block Puzzle Game that has commonality of rules in automatically generating game rules and is easy to systematize. By using our proposed language, we were able to implement rules that could not be realized by the previous method. Furthermore, a plurality of rules of a falling puzzle game are defined, and various game rules can be generated.

目 次

図 目 次

第 1 章

はじめに

1.1 研究背景と目的

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] [2] 、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 [3]。そのほかターン制ストラテジーのマップを自動生成する研究 [4] や、アクションゲームのステージを自動生成する研究 [5] も存在する。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 [6] や、ファンタシースターオンライン 2[7] などが存在する。物語の自動生成を行ったゲームも存在しており、ティル・ナ・ノーグ [8] や The Murder Mystery Machine[9] などの事例がある。

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。数独においても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る。本研究ではそこに着目し、ゲームルールの自動生成を目的とした。

Cameron Browne ら [10] 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）[11] という手法によってルールの体系化を行っている。また、GDL を用いる手法とは別の体系をとり、ゲーム自体を自動生成する研究 [12] も存在する。

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 [13] [14] [15]。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。General Game Playing は囲碁やチェスといったボードゲーム上で行われている研究分野だが、General Video Game Playing と呼ばれるビデオゲーム研究分野も存在する [16]。

Togelius ら [13] は GGP が目指している目標や手法などを述べている。Genesereth ら [14] は GGP に関する概要とその問題について述べている。Gaina ら [15] は GGP における 2 人用ゲームの AI に関する研究を行い、GDL を用いて実現している。

GDL で再現できるゲームは、ボードゲームなどのアナログなものが対象である。”Video Game Description Language”(VGDL)[17] [18] というデジタルゲームへ向けた共通ルール記述言語も存在する。

Tom[17] は 2D ビデオゲーム用の GDL である「PyVGDL」を提案している。Quiñones ら [18] は XML をベースとしたデジタルゲーム専用の記述言語「XVGDL」を提案している。

VGDL を用いた研究もいくつか存在し、その中でもゲームルールやレベルデザインの評価を行う研究などに用いられることが多い [19] [20] [21] [22]。このように VGDL はゲームの解析や GGP における AI 作成などに大きく貢献している。

Chong-U ら [19] は PuzzleScript を用いて作成されたゲームのルール解析やレベルデザインを自動的に行う手法を提案した。Barros ら [20] は VGDL でランダムにゲームルールを生成し、ゲームが最後までプレイ可能かの検証を行っている。Nielsen ら [21] は VGDL で自動生成ゲームに対し、評価する方法を提案している。Ahmed ら [22] は PuzzleScript を用いて作成されたゲー

ムのルール関係なく、ゲームレベルの評価をする手法を提案している。VGDL を用いたビデオゲームのルールの自動生成をする研究 [23] [24] も存在し、近年まで研究されてきた。

Togelius ら [23] はゲームルール自体の自動生成に焦点を当て、自動生成をする試みをしている。Ahmed Khalifa ら [24] は基本的なビデオゲームルール生成における問題点について述べ、解決するためのフレームワークを提案している。

しかし、これらの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」[25] 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 [26] やテトリス [27] 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。

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

提案した FPGDL を用いることにより、先行手法において実現できなかったゲームルールを実現できた。さらに、複数定義したゲームルールをでたらめに組み合わせ自動的にルールを生成し、落ちものパズルゲームとしてプレイが可能なゲームルールの自動生成が可能となった。

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。

1.2 論文構成

本論文は全 5 章で構成する。構成は 2 章では落ちものパズルゲームにおけるルールの定義と体系化について述べる。3 章では提案落ちものパズルゲーム共通ルール記述言語について述べ、4 章では検証と評価を述べる。そして 5 章ではまとめを述べる。

第 2 章

落ちものパズルゲーム

本章では落ちものパズルゲームにおけるルール定義と体系化について述べる。

2.1　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。

図 2.1 は落ちものパズルゲームの各部分の名称を示したものである。

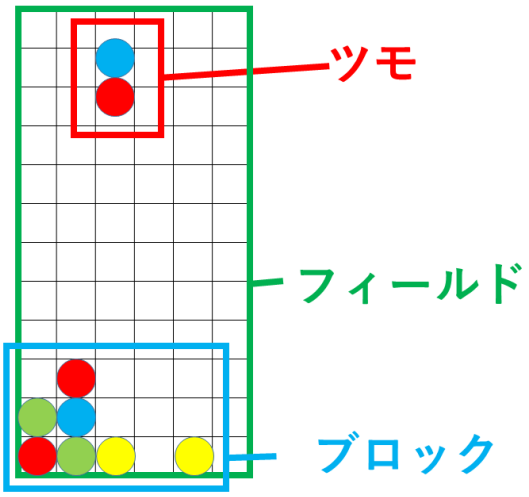

図 2.1　落ちものパズルゲームの要素図示

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。

- 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。

2.2 落ちものパズルゲームの体系化

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

- ブロックの消滅方法
- 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 では異なる種類のブロックを用意できるようにするため、複数種類のブロックを定義できるように設計した。

ツモ:

落ちものパズルゲームにおけるツモとは、複数個のブロックで成り立っている。プレイヤーはツモに対して移動や回転行動をすることができる。ツモはルールによって形状が決められており、それらをツモ形状と呼称する。ツモは回転行動を実行する際、さらに形状パターンをいくつか持っているものがほとんどである。これらの形状を回転形状と呼称する。ツモ形状には主に 2 つのパターンが存在し、「固定形ツモ」と「不定形ツモ」がある。

固定形ツモはゲームの開始から終了までツモとして出現するブロックの形状パターンが常に一定のものである。固定形ツモを用いた落ちものパズルゲームの特徴として、ブロックの色や形状が複数種類存在するものが多い。固定形ツモを用いたゲームとしてはぷよぷよが挙げられる。

不定形ツモはゲーム開始から終了までツモとして出現するブロックの形状パターンが異なる。既存落ちものパズルゲームにおいてテトリスなどが不定形ツモに該当する。テトリスは 4 つのブロックを組み合わせた 7 種類の「テトリミノ」と呼ばれるツモを使うテトリスは不定形ツモのゲー

ム代表である。不定形ツモを用いた落ちものパズルゲームの特徴としてブロック色や形状で区別をしないものが多い傾向にある。

回転法則は主に 4 つのパターンが存在し、「軸固定型」、「軸可動型」、「入れ替え型」、「変化型」である。軸固定型はツモのどこかを軸にし、回転していく方式である。既存落ちものパズルゲームにおいてぷよぷよ通やぱにっくボンバー [28] などが軸固定型にあたり、回転形状は基本的に 4 パターン存在する。図 2.2 は軸固定型のブロックを軸とした回転例である。

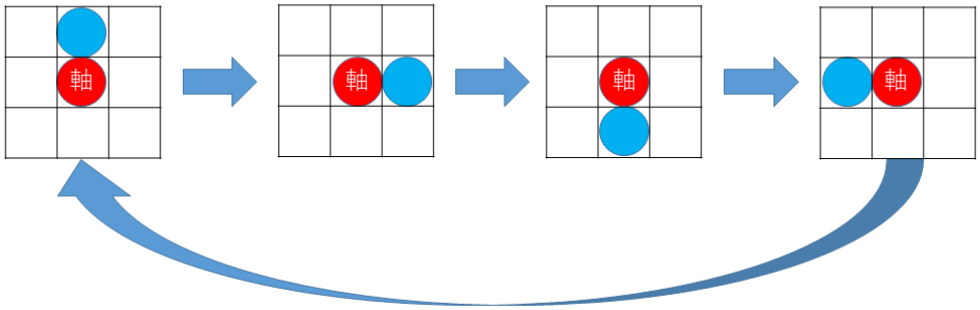

図 2.2　軸固定型のブロックを軸とした回転例

軸固定型は、ブロックを軸とするものではなく、ブロックとブロックの間を軸とすることもある。図 2.3 は軸固定型のブロック以外を軸とした回転例である。

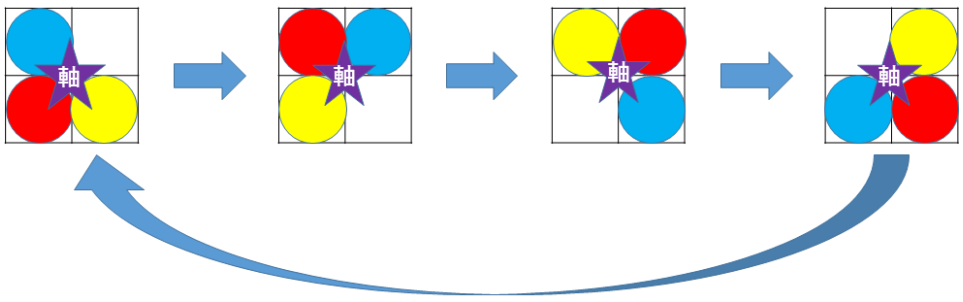

図 2.3　軸固定型のブロック以外を軸とした回転例

軸可動型は回転する際ほとんどはブロックの組み合わせを回転したようにみえるが軸が固定されていないものである。軸可動型を用いた落ちものパズルゲームは、テトリスのクラシックルールに起用されていた回転法則に多く見られる。軸可動型のクラシックルールのテトリスでは、本来 90 度ずつ回転するような動作をするため、回転形状が 4 パターンあるはずが、2 パターンの回転形状しかないツモの形状も存在する。図 2.4 はテトリスの「Z」型ツモの 4 パターンと 2 パターンの回転形状を表したものである。

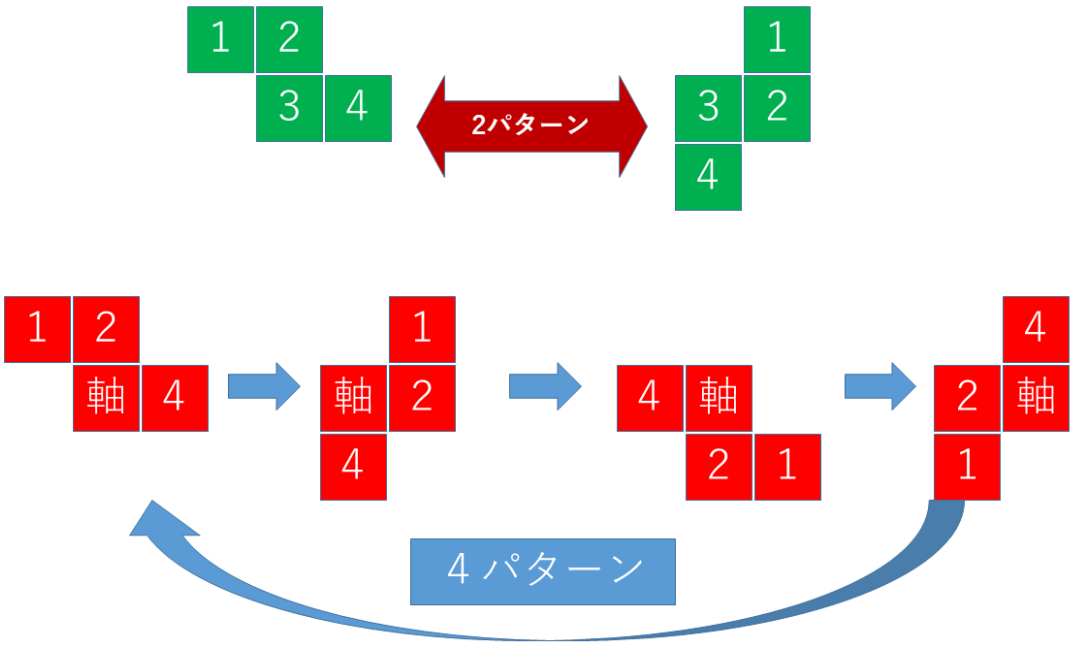

図 2.4　テトリスの「Z」型ツモの 2 パターンと 4 パターンの回転形状

入れ替え型はツモの形状自体は変化しないが、ブロックとブロックの位置をツモの中で入れ替える回転法則である。入れ替え型を用いた落ちものパズルゲームはコラムス [29] が該当する。入れ替え型のツモ形状はほとんどが縦か横に並んだツモ形状になっていることがほとんどであり、回転形状も並んでいるブロック数依存になることが多い。図 2.5 は入れ替え型の回転例である。

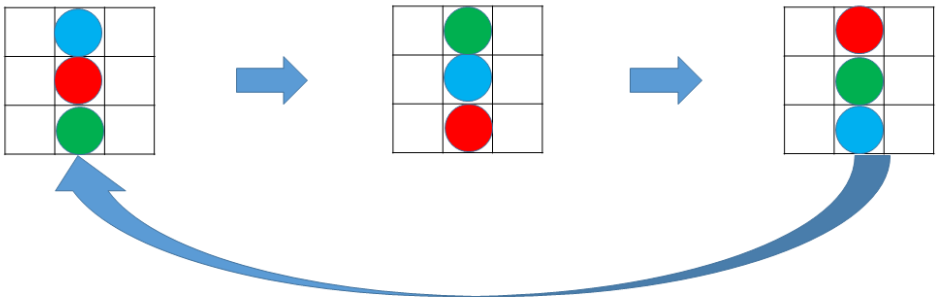

図 2.5　軸固定型の回転例

変化型はツモにあるブロックそのものの色が変化する回転法則である。変化型を用いた落ちものパズルゲームはぷよぷよフィーバーにおけるでかぷよなどが該当する。変化型をメインに使うゲームは少ない。図 2.6 は変化型の回転例である。

図 2.6　変化型の回転例

以上のようなツモ回転法則や回転形状へ対応するために、FPGDL では各ツモ形状毎に回転形状を定義できるように設計した。これによりテトリスやぷよぷよ、ぱにっくバンバーの変則的なツモへ対応が可能である。

消滅条件:

落ちものパズルゲームにおいて消滅条件はもっとも複雑化するものである。数種類に形態を纏めることはできるが、ゲーム毎に大きく異なるため分類は困難である。消滅条件を大きくまとめると「連結型」「直列型」「固定形型」「起爆型」「特殊型」に分類することができる。

連結型とは、形は問わず上下左右に同色ブロックが一定数以上繋がっているときに発生する消滅条件である。既存落ちものパズルゲームにおいてぷよぷよなどは連結型の消滅条件であり、同じ色のブロックが 4 つ以上繋がることで消滅条件が成り立つ。図 2.7 は連結型の消滅条件発生状況の一例であり、4 つ以上同色ブロックが連続で隣接すると消滅条件が発生するルールを想定している。

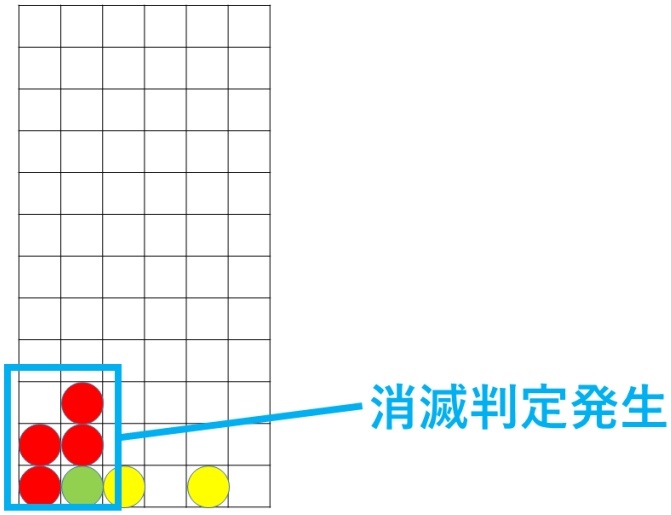

図 2.7　連結型の消滅条件発生状況の一例

直列型とは、縦、横、斜めなど一直線に同色のブロックが一定数以上並んでいるときに発生する消滅条件である。既存落ちものパズルゲームにおいてぱにっくボンバー、コラムスなどのゲームは直列型に該当する。図 2.8 は直列型の消滅条件発生状況の一例であり、3 つ以上同色のブロッ

クが連続で並んでいるときの消滅条件を想定している。

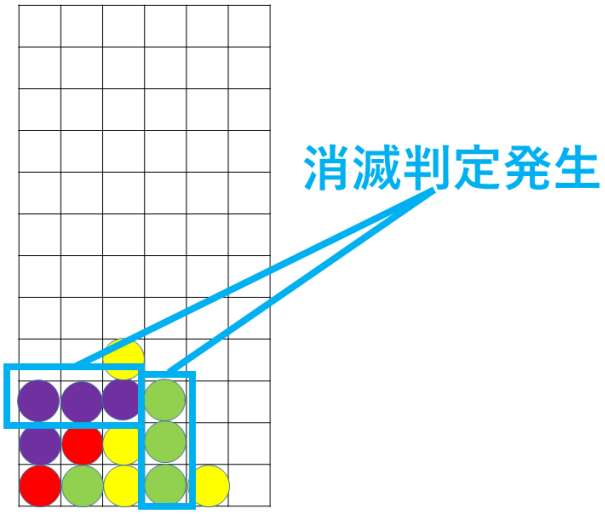

図 2.8　直列型の消滅条件発生状況の一例

固定形型とは、同色のブロックで固定の形をとることで消滅条件が成り立つものである。既存落ちものパズルゲームにおいてテトリスやルミネス [30] などが固定形型の消滅判定に該当する。図 2.9 は固定形の消滅条件発生状況の一例であり、同色のブロック同士を 2 × 2 の四角にしたときの消滅条件を想定している。

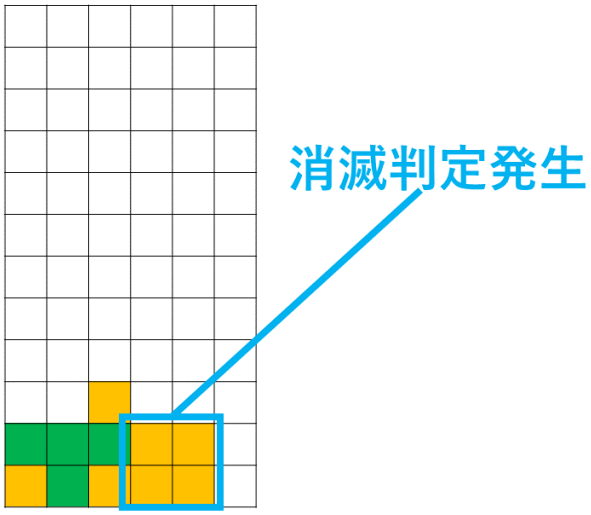

図 2.9　固定形型の消滅条件発生状況の一例

起爆型とは、導火線ブロックと呼ばれるブロックを、対の起爆剤となるブロックと隣接させることで消滅条件が成り立つものである。起爆型のほとんどは、起爆剤と対になる導火線ブロックがある。起爆剤を対になる導火線ブロックに隣接させ、同じ色の導火線ブロック同士が隣接していた場合それらも一緒に消滅する。起爆剤のブロックと導火線ブロックは同じ色で消滅条件が成り立つものが多い。既存落ちものパズルゲームにおいて起爆型の消滅条件はにょきにょき [31] やバクバクアニマル [32] などが該当する。図 2.10 は起爆型の消滅条件発生状況の一例である。

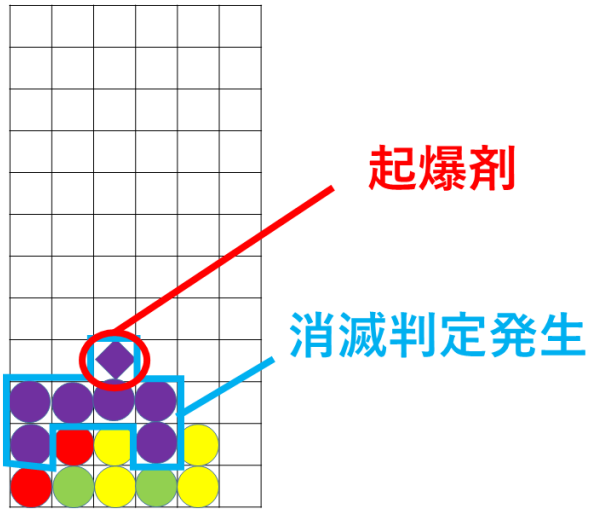

図 2.10　起爆型の消滅条件発生状況の一例

連結型、直列型、固定形型、起爆型とは別に特殊な消滅条件をもつ特殊型が存在する。特殊型は隣接しているブロックが消滅したら消滅を開始し、時間経過で消滅判定が出るものである。特殊型は、ぷよぷよにおけるおじゃまぷよなどが該当する。

消滅条件は特に複雑なものであり、様々な条件に対応できるようにする必要がある。特に FPGDL では上記の条件に当てはまらない落ちものパズルゲームの消滅条件にも対応する必要があると考えた。そこで FPGDL では消滅条件のテンプレートなどは用意せず、ブロックのある条件における隣接状態の取得を可能にすることで対応した。

ブロックの隣接状態の条件は主に隣接している同色の数、縦や横さらに斜めなど連続で直線状に並んでいる同色ブロックの数を取得できるようにした。さらに、フィールドに配置されたブロックの色状態を取得できる機能もあるため、これらの条件に当てはまらない消滅条件を用意することが可能である。

2.3　消滅方法

落ちものパズルゲームにおける消滅方法は「即時消滅」「変化」「貯蓄」の 3 パターンに分類することができる。

即時消滅とは消滅条件が成立したブロックが即消滅アクションに入る消滅方法のことを指す。消滅アクションに入ったブロックは消滅アニメーションを再生しつつフィールドから消滅する。既存落ちものパズルゲームにおいて即時消滅はぷよぷよやテトリスなどが該当する。

変化とは消滅条件が成立したブロックが他の色に変化する消滅方法のことを指す。消滅方法が変化である場合、たいていの落ちものパズルゲームにおいて数回変化を繰り返すと即時消滅する。変化段階が最大 6 段階である場合、1 段階目から変化させていくと 5 回変化を繰り返すとブロックは即時消滅をし、3 段階目の色をそのまま消滅条件成立させると 4 段階目に移行し、3 段階目から変化させた場合の変化は 2 回のみである。消滅方法が変化である場合はブロックの色ごとで消滅条件が違うものがほとんどである。

貯蓄は消滅条件が成立しても一定時間あるいは追加消滅条件が成立するまでフィールド上にブロックが留まる消滅方法のことを指す。追加消滅条件は時間によるものや、ゲーム中のギミックを使用するものなどさまざまである。既存落ちものパズルゲームにおいて貯蓄はルミネスなどが該当する。

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。

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。

第 3 章

落ちものパズル共通ルール記述言語

本章では提案する落ちものパズル共通ルール記述言語 FPGDL について述べる。

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。

次のコードは FPGDL において初期設定フェーズに記述をした際の例である。

```
1  --初期設定フェーズ
2  setFieldSize ( 6 , 13 ) --パズルのプレイフィールドのサイズ指定
3  blockSize ( 30 )  --ブロックの描画サイズ指定
4  setSuperRotation( false ) --回転補正を無効にする
5  setQuickTurn( true ) --クイックターンを有効にする
6  setClearMode( -1 )  --消去ルールモードを大まかに指定(自ら組む場合-値)
7  setHardDrop( false ) --ハードドロップを無効にする
8
9  --ブロックを作成
10 createBlockPattern( 1 )
11 createBlockPattern( 2 )
12 createBlockPattern( 3 )
13
14 --ブロックの形状を指定
15 setBlockForm( 1 , 1 )
16 setBlockForm( 2 , 1 )
17 setBlockForm( 3 , 1 )
18
19 --ブロックに色を指定
20 setBlockFormColor( 1 , 255 , 0 , 0 )
21 setBlockFormColor( 2 , 0 , 255 , 0 )
22 setBlockFormColor( 3 , 80 , 80 , 255 )
```

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。アップデートフェーズの記述は以下のように行う。

```
1
2  --アップデートフェーズ
3  function update()
4
5     --落下させる量を保持する変数
6     downnum = 0
7     y = getFieldHeight()
8
9     for p = 0 , getFieldHeight() do
10
11       y = y - 1
12
13       --4つ以上くっついていたら消える
14       for x = 0 , getFieldWidth() do
15
16          --常に落下をさせる
17          setBlockDownCellNum( x , y , getFieldHeight() )
18
19          --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
20          if getComboBlockNum( x , y ) >= 4 and getFieldBlock( x , y ) < 100 and
                getAllBlockDownFlag() == false then
21             setBlockDeleteFlag( x , y ) --ブロックを消滅させる
22          end
23
24       end
25
26    end
27
28 end
```

FPGDL から提供しているメソッドの細かい機能に関しては、付録のリファレンスを参照してほしい。

図 3.1 は FPGDL のフェーズを図解したものである。

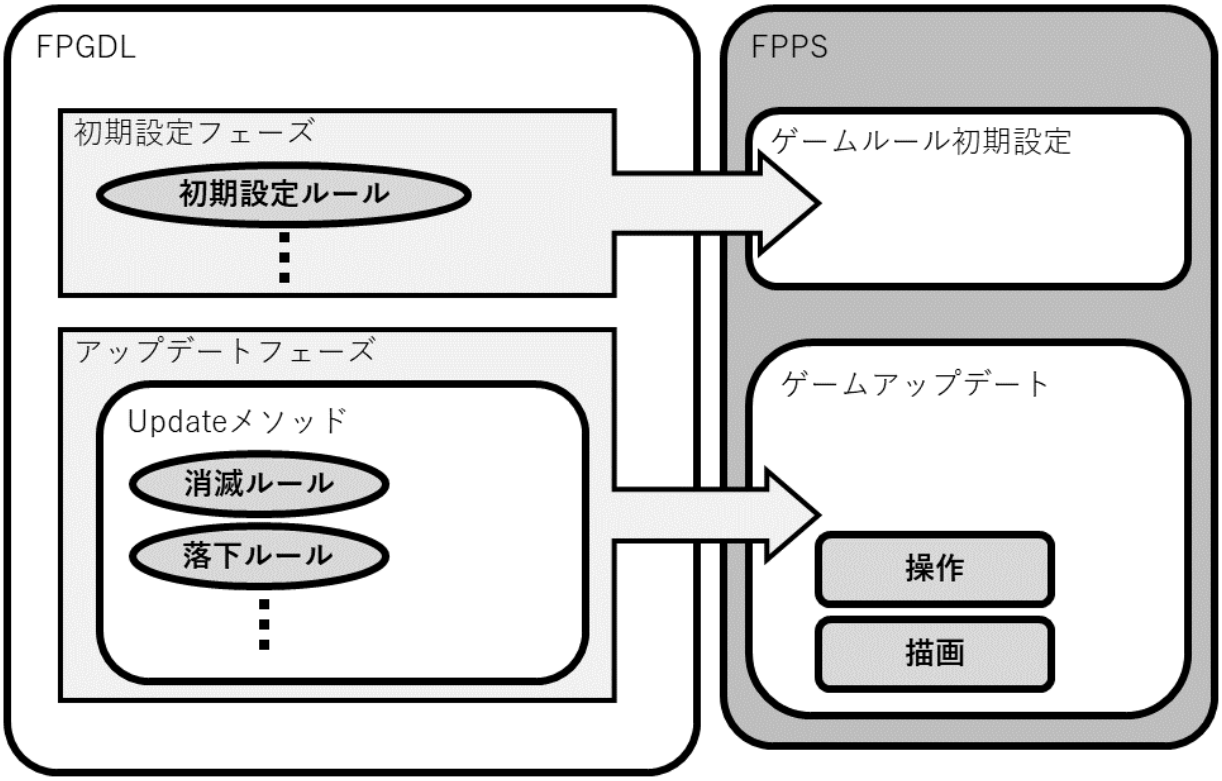

図 3.1　FPGDL と FPPS の概要

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。

第 4 章

検証・評価

この章では提案した FPGDL の有用性の評価と、共通ルール記述言語を用いたゲームルールの自動生成が可能であるか検証を行う。FPGDL の有用性は既存の一般的な落ちものパズルゲームが実装できるか、既存手法の落ちものパズルゲームを作成するゲームソフトにて再現不可能だったルールの再現が可能かどうかで評価を行う。

4.1 FPGDL の有用性評価

今回有用性を評価するため、一般的な落ちものパズルゲーム「ぷよぷよ」「コラムス」「テトリス」の 3 つを FPGDL を用いて行った。

4.1.1 ぷよぷよの実装

ぷよぷよはほとんどの既存手法である落ちものパズルゲームを作成するゲームソフトで実現が可能である。FPGDL を用いて記述したものは付録 B.1 を参照してほしい。

図 4.1 は FPGDL を用いて実装したぷよぷよの画面である。

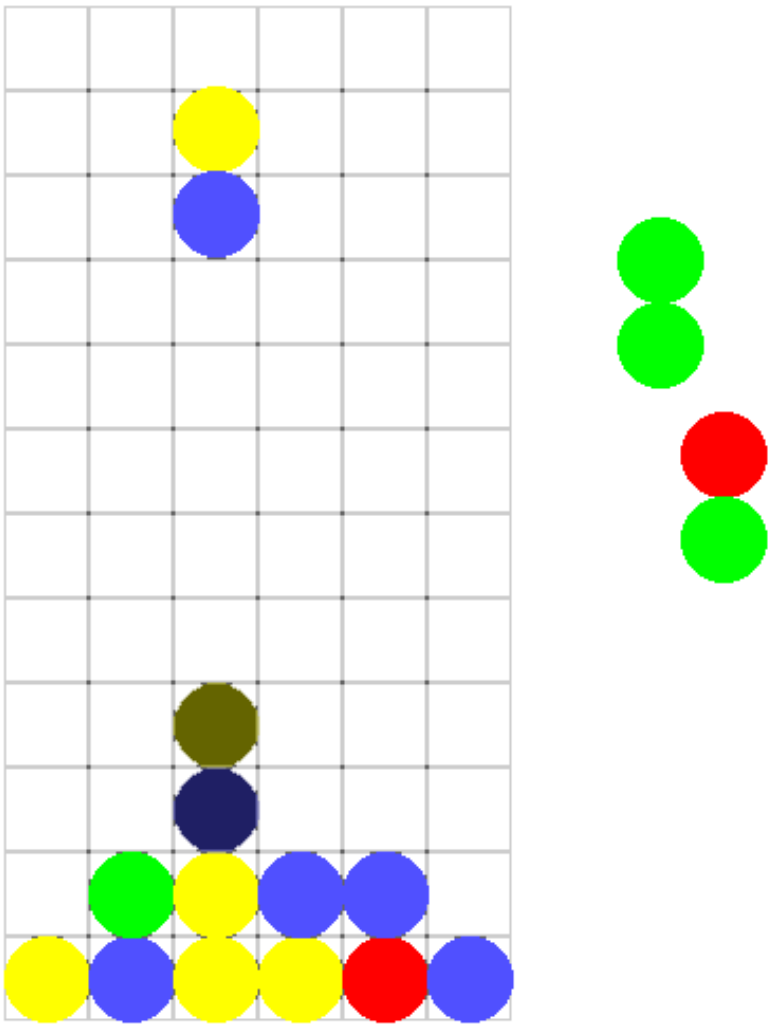

図 4.1　ぷよぷよの実行画面

4.1.2　コラムスの実装

コラムスは落ちものパズルゲームの中でもぷよぷよやテトリスと並び代表的と言えるゲームルールである。したがって、FPGDL においても実現ができるか検証を行った。

図 4.2 は FPGDL を用いて実装したコラムスの画面である。

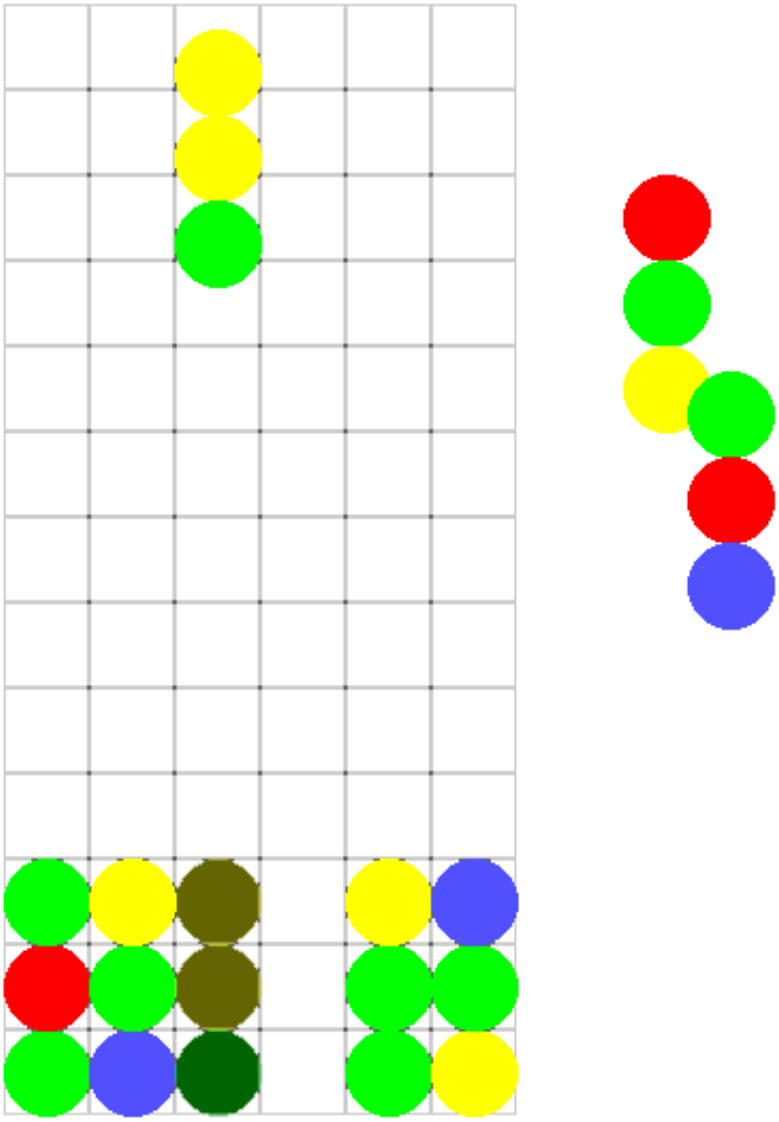

図 4.2　コラムスの実行画面

4.1.3　テトリスの実装

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。FPGDL を用いて記述したものは付録 B.2 を参照してほしい。

図 4.3 は FPGDL を用いて実装したテトリスの画面である。

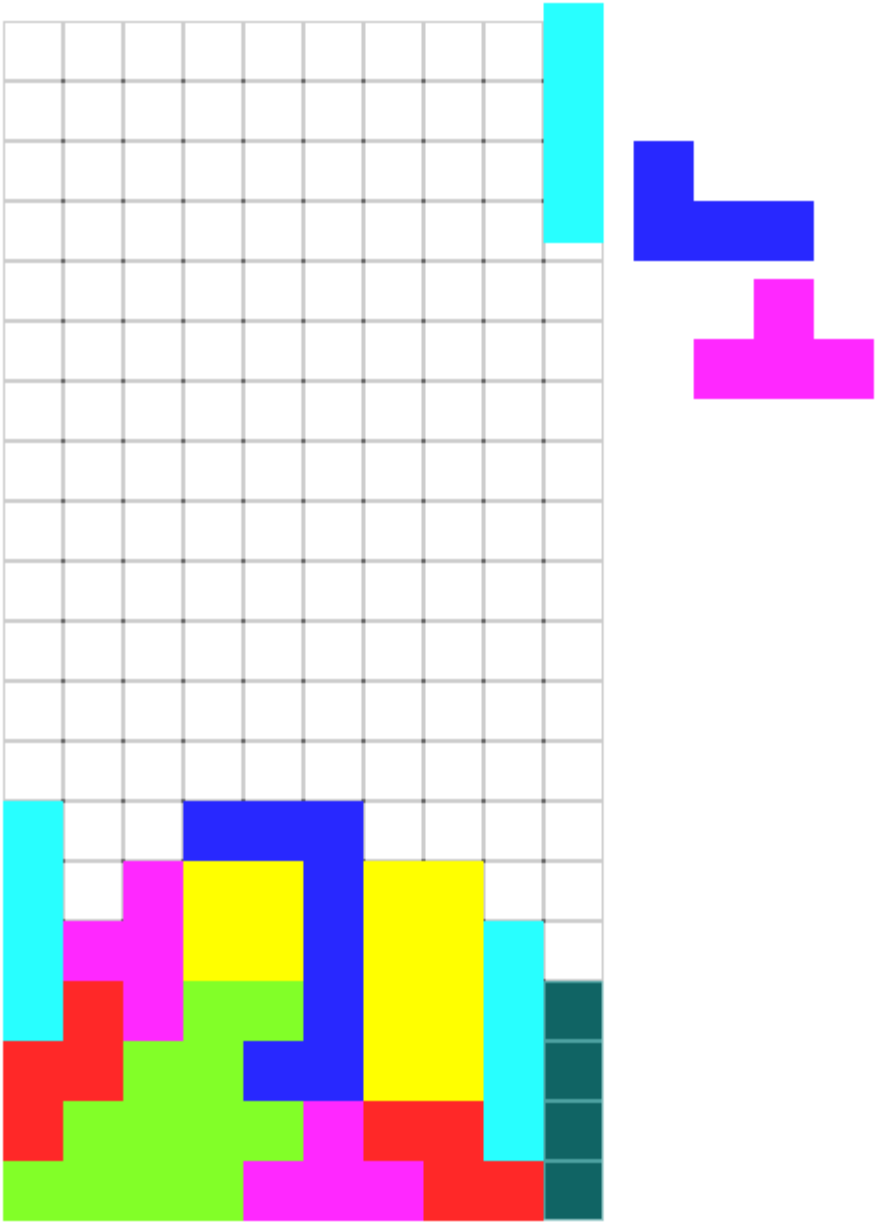

図 4.3　テトリスの実行画面

一般的な落ちものパズルゲームであるぷよぷよ、コラムス、テトリスの実装をし、ゲームルー

ルの実現が問題なくできた。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。

4.2 FPGDL を用いたルール自動生成の検証

落ちものパズルゲームのゲームルールを自動生成するにあたって、FPGDL を提案した。FPGDL を用いて落ちものパズルゲームのゲームルール自動生成が可能かどうかを検証する。体系的な自動生成が可能かどうかを検証するにあたり、複数の落ちものパズルゲームを組み合わせたルールを作成できるか検証を行った。さらに、細分化したルールをでたらめに選出をし、落ちものパズルゲームのルールとして成立するか検証を行った。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以下の区分で細分化し FPGDL に記述する。

- 回転法則
- ツモ形状
- ブロック
- 落下法則
- 消滅条件
- ゲームオーバー

今回混合させる落ちものパズルゲームのルールとして定義したルールはルミネス、ぱにっくボンバー、テトリスの 3 種である。

4.2.1 複数の落ちものパズルゲームを組み合わせたルール

今回落ちものパズルゲームの体系的な自動生成が可能かどうかを検証するにあたり、ぷよぷよとテトリスを組み合わせたルールを実装した。結果、ぷよぷよの消滅条件に加え、テトリスの消

滅条件の実行、ならびにブロックの役割をぷよぷよにし、ツモ形状をテトリスにすることを試みた。結果、ぷよぷよとテトリスの混合ルールの作成に成功した。FPGDL を用いて記述したものは付録 B.4 を参照してほしい。

図 4.4 は FPGDL を用いて実装したぷよぷよとテトリスの混合ルールの実行画面である。

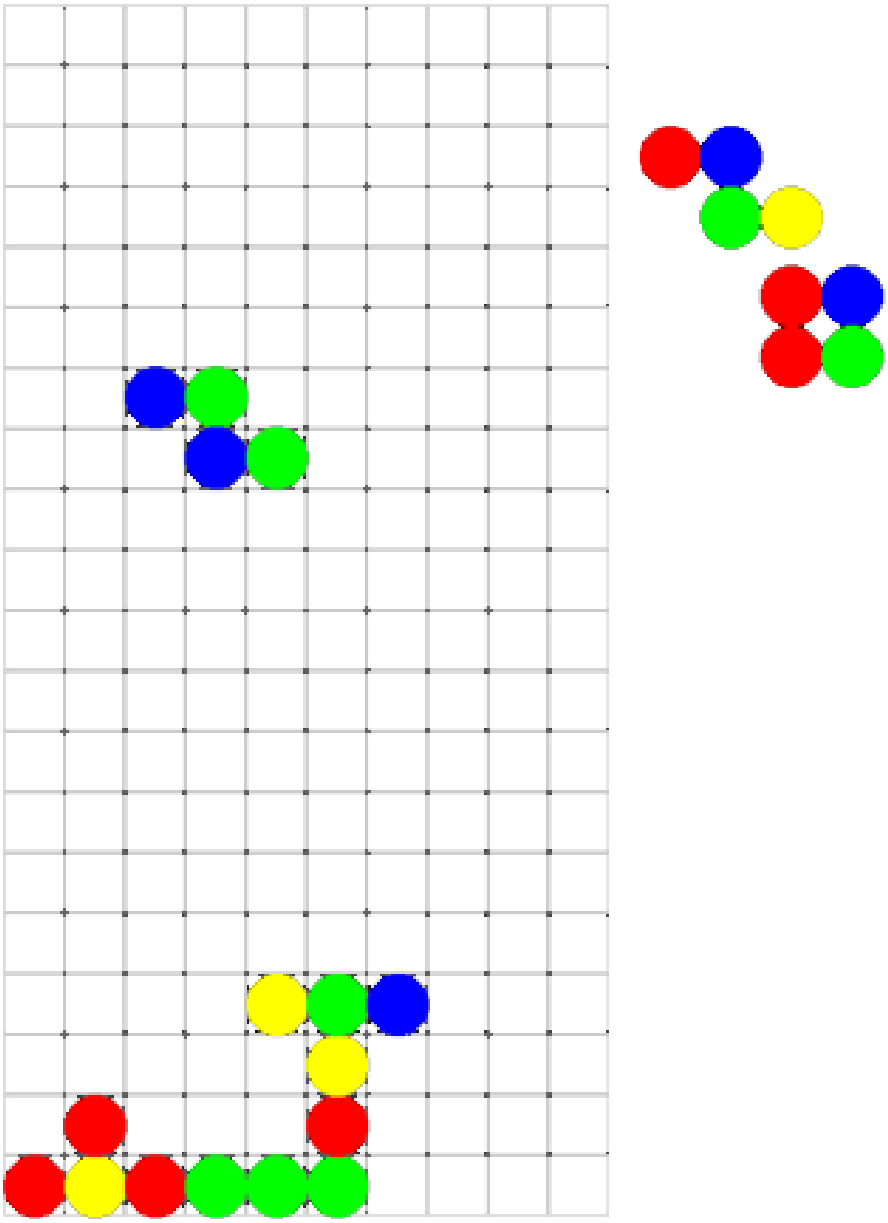

図 4.4　ぷよぷよとテトリスの混合ルールを実行した画面

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
- ゲームオーバー：両共通

4.2.2　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 4.5 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。

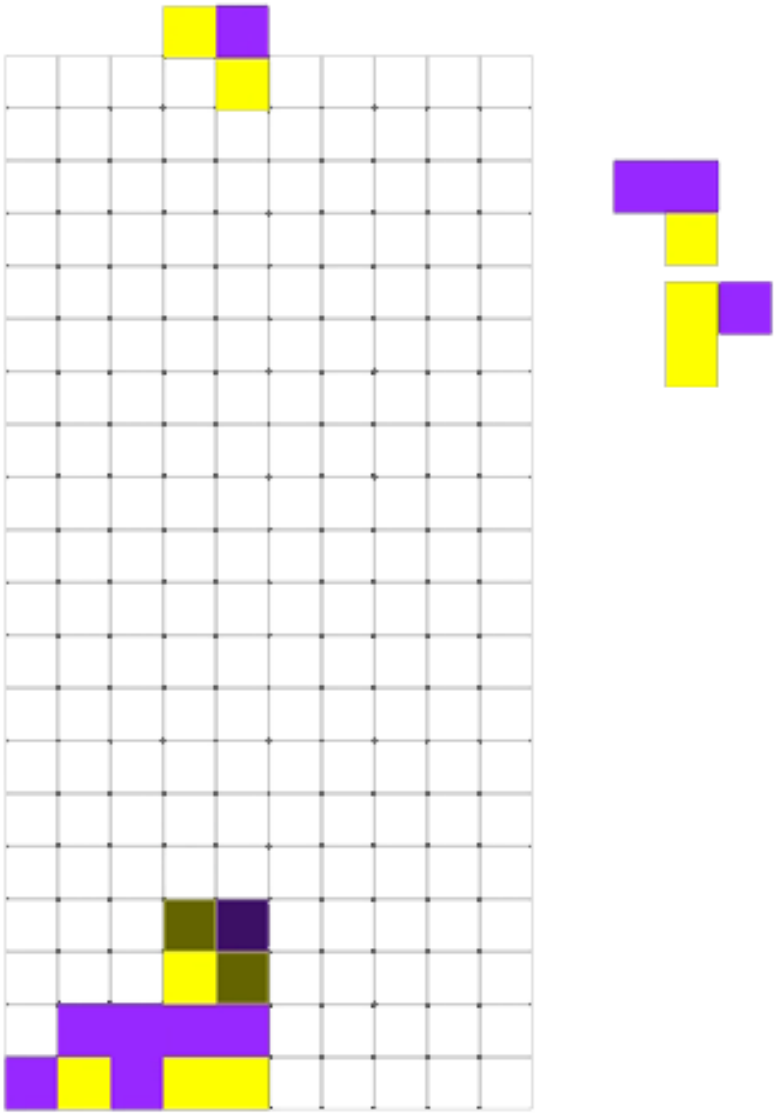

図 4.5　複数ルールをでたらめに適用したゲームの実行画面

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

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。

第 5 章

まとめ

本章では本研究のまとめを述べる。本研究は落ちものパズルゲームの自動生成を目的とし、落ちものパズルゲームの共通ルール記述言語を提案し、落ちものパズルゲームの体系的なゲームルール自動生成が可能であるかを検証した。提案 FPGDL は先行手法では不可能だった落ちものパズルゲームのルールを再現することに成功した。さらに、落ちものパズルゲームの体系的なゲームルールを自動生成が可能であるかの検証を行うために、落ちものパズルゲームのゲームルール同士を組み合わせたものが記述できるか検証したところ、落ちものパズルゲームのルール同士を組み合わせたルールを記述することも可能であることがわかった。さらに複数のゲームルール同士を自動で組み合わせるような仕組みを作成し実行したところ、様々なルールの落ちものパズルゲームが生成され、実行されることを確認した。今後の展望として、生成された落ちものパズルゲームの面白みの評価の実現を挙げる。生成された落ちものパズルゲームの面白みを評価することで、新しいコンテンツの生成および入力したゲームルールがプレイヤーにとって面白いかを事前検証することができるシステムが作成できると考えられる。

謝辞

本論文を執筆するにあたりご指導してくださった先生方、学会発表を行った際に意見をくださった皆様、この世に生み出された落ちものパズルゲームたちおよびに制作にかかわった皆様に、この場を借りて感謝をいたします。

参考文献

[1] 前田一貴, 奥乃博. 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, mar 2008.

[2] 山崎隆介, Grimbergen Reijer. 連鎖型パズルゲームにおけるパズル問題の自動創作. ゲームプログラミングワークショップ 2013 論文集, pp. 118–121, nov 2013.

[3] 香川俊宗, 手塚宏史, 稲葉真理. 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, 第 2015 巻, pp. 326–333, Sep 2015.

[4] 田口紫織, 佐藤直之. Ga を用いたターン制ストラテジーゲームのマップ自動生成. Technical Report 20, 佐世保工業高等専門学校, 佐世保工業高等専門学校, mar 2019.

[5] 清水智行, 橋山智訓, 江崎朋人, 市野順子, 田野俊一. フローチャンネルを利用したゲームステージのオンライン自動生成. 日本知能情報ファジィ学会 ファジィ システム シンポジウム講演論文集, Vol. 27, pp. 127–127, 2011.

[6] 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.

[7] 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.

[8] 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.

[9] The Murder Mystery Machine. Steam：the murder mystery machine. https://store.steampowered.com/app/930280/The_Murder_Mystery_Machine/. 参照:2020.2.15.

[10] C. Browne and F. Maire. Evolutionary game design. In *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 2, pp. 1–16, 2010.

[11] Nathaniel Love, Timothy Hinrichs, and Michael Genesereth. General game playing:

Game description language specification. In *Stanford University, Tech.Rep*, LG-2006-11(2006).

[12] J. Togelius and J. Schmidhuber. An experiment in automatic game design. In *2008 IEEE Symposium On Computational Intelligence and Games*, pp. 111–118, Dec 2008.

[13] J. Togelius and G. N. Yannakakis. General general game ai. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–8, Sep 2016.

[14] Genesereth M, Love N, and Pell B. General game playing: Overview of the aaai competition. In *AI Magazine*, Vol. 26(2), pp. 62–72, Jun 2005.

[15] R. D. Gaina, D. Pérez-Liébana, and S. M. Lucas. General video game for 2 players: Framework and competition. In *2016 8th Computer Science and Electronic Engineering (CEEC)*, pp. 186–191, Sep 2016.

[16] John Levine, Clare Bates Congdon, Marc Ebner, Graham Kendall, Simon M. Lucas, Risto Miikkulainen, Tom Schaul, and Tommy Thompson. General video game playing. In Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius, editors, *Artificial and Computational Intelligence in Games*, Vol. 6 of *Dagstuhl Follow-Ups*, pp. 77–83. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2013.

[17] Tom Schaul. A video game description language for model-based or interactive learning. In *IEEE Conference on Computational Inteligence in Games (CIG)*, pp. 1–8, Aug 2013.

[18] J. R. Quiñones and A. J. Fernández-Leiva. Xml-based video game description language. *IEEE Access*, Vol. 8, pp. 4679–4692, 2020.

[19] Chong-U Lim and D. F. Harrell. An approach to general videogame evaluation and automatic generation using a description language. In *2014 IEEE Conference on Com-*

putational Intelligence and Games, pp. 1–8, Aug 2014.

[20] G. A. B. Barros and J. Togelius. Exploring a large space of small games. In *2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–2, Aug 2014.

[21] T. S. Nielsen, G. A. B. Barros, J. Togelius, and M. J. Nelson. Towards generating arcade game rules with vgdl. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 185–192, Aug 2015.

[22] Ahmed Khalifa and Magda Fayek. Automatic puzzle level generation : A general approach using a description language. 2015.

[23] J. Togelius and J. Schmidhuber. An experiment in automatic game design. In *2008 IEEE Symposium On Computational Intelligence and Games*, pp. 111–118, Dec 2008.

[24] A. Khalifa, M. C. Green, D. Perez-Liebana, and J. Togelius. General video game rule generation. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 170–177, Aug 2017.

[25] Super Mash. Supermash for nintendo switch - nintendo game details. `https://www.nintendo.com/games/detail/supermash-switch/`. 参照:2019.12.16.

[26] 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.

[27] テトリス. Tetris — the addictive puzzle game that started it all! `https://tetris.com/`. 参照:2019.7.12.

[28] ぱにっくボンバー. ボンバーマン ぱにっくボンバー ¿ バーチャルコンソール — virtual console. `https://web.archive.org/web/20090714031130/http://vc-pce.com:80/jpn/j/title/panibom.html`. 参照:2020.2.16.

[29] コラムス. コラムス — 株式会社セガ・インタラクティブ. `https://sega-interactive.`

co.jp/product/9226/. 参照:2020.2.16.

[30] ルミネス. ルミネス　エレクトロニック シンフォニー — トップページ — ubisoft. https://ubisoft.co.jp/lumines-es/. 参照:2020.2.16.

[31] にょきにょき. にょきにょき — compile-o. https://www.compile-o.com/blank-6. 参照:2020.2.16.

[32] ばくばくアニマル. ばくばくアニマル［android］ - 4gamer.net. https://www.4gamer.net/games/242/G024296/. 参照:2020.2.16.

付録 A

リファレンス

ここではFPGDLのリファレンスやサンプルコードを記述する。

A.1 初期設定系

初期設定系は、落ちものパズルゲームの事前に指定しておくルールを決定するものである。初期設定系の機能は以下のとおりである。

A.1.1 setFieldSize(x , y)

フィールドの大きさを決定する。xには横の大きさ、yには縦の大きさをそれぞれ整数値で指定する。以下のソースコードはsetFieldSizeの記述例である。

```
1
2 --フィールドの大きさを横幅 6、縦幅 12に指定
3 setFieldSize( 6 , 12 )
```

A.1.2 blockSize(size)

画面に描画するブロック単体のサイズを変更する。sizeにはブロックの沖差を整数値で指定する。以下のソースコードはblockSizeの記述例である。

```
1
2 --画面に描画するブロック単体のサイズを 40に指定
3 blockSize( 40 )
```

A.1.3 setClearMode(mode)

ルールテンプレートの使用有無を決定する。modeにはルールテンプレートの識別番号を整数値で指定する。ルールテンプレートの識別番号は0を指定することで消滅条件連結型、なおかつ消滅方法即時消滅のものを適用することができ、1を指定することで消滅条件固定形型（横一列)、なおかつ消滅方法即時消滅のものが適用できる。なお、ルールテンプレートの識別番号に0を指定した際はclearBlockNumで消滅条件が成立する同役割のブロックの連結数を指定することが

可能である。ルールテンプレートの識別番号に 1 を指定した際、ブロックの役割関係なくブロックの有無を確認し、消滅条件を適用する。ルールテンプレートの識別番号にマイナス値を指定することによって自身で指定したルールを適用する。以下のソースコードは setClearMode の記述例である。

```
1
2 --ルールテンプレートを 0に設定する
3 setClearMode( 0 )
```

A.1.4 clearBlockNum(num)

setClearMode にてルールテンプレートを 0 に指定した際の、消滅条件が成立する同役割のブロックの連結数を決定する。num には消滅条件が成立する同役割のブロックの連結数を整数値で指定する。なお、この機能は setClearMode にて 0 以外を指定した際は無効となる。以下のソースコードは clearBlockNum の記述例である。

```
1
2 --ルールテンプレートを 0に設定する
3 setClearMode( 0 )
4 --ブロックの連結数は 4
5 clearBlockNum( 4 )
```

A.1.5 createBlockPattern(num)

ゲーム中に使用するブロックを作成する。num にはゲーム中に使用するブロックの役割識別番号を整数値で指定する。なお、num に 0 を指定するとブロックは作成されない。以下のソースコードは createBlockPattern の記述例である。

```
1
2 --ブロックを 4種類用意
3 createBlockPattern( 1 )
4 createBlockPattern( 2 )
5 createBlockPattern( 3 )
6 createBlockPattern( 4 )
```

A.1.6 setSuperRotation(flag)

回転時に補正をかけた回転の使用有無を決定する。flag には使用する場合は true、使用しない場合は false を指定する。flag に true を指定すると、通常回転操作をした際に、回転後の回転形状がフィールドに配置されたブロックと被ってしまう場合回転動作は行われないが、ツモの座標に移動補正をかけて回転が可能だった場合、ツモの座標に移動補正をかけ回転動作を行うようになる。以下のソースコードは setSuperRotation の記述例である。

```
1
2 --回転時に補正をかけた回転を使用する
3 setSuperRotation( true )
```

A.1.7 setQuickTurn(flag)

2 回連続で回転操作をすると、1 つ先の回転形状が回転不可だった場合 2 つ先の回転形状で回転が可能かを試みる機能の使用有無を決定する。このような動作を、落ちものパズルゲームにおいて「クイックターン」と呼称する。flag には使用する場合は true、使用しない場合は false を指定する。以下のソースコードは setQuickTurn の記述例である。

```
1
2 --クイックターンを使用する
3 setQuickTurn( true )
```

A.1.8 setHardDrop(flag)

特定のキー操作をした際に、即配置動作を行うかどうかを決定する。flag には使用する場合は true、使用しない場合は false を指定する。flag に true を指定した場合、通常下キーを押してフィールドにツモのブロックを配置しようとすると、フィールドの一番下まで落下しツモのブロックがどれか一つでもフィールドの一番下に衝突するまたはツモのブロックどれか一つでもフィールドに配置したブロックに対して下方向に衝突した際に即フィールドにブロックを配置するが、

この条件を満たしていた場合でも猶予時間が発生し、下キーを押してもブロックを即配置しなくなる。上キーを押すと即配置する動作をとるが、ツモのブロックが配置される状況ではなかったとしても、強制的にフィールドの一番下にツモのブロックどれか一つでも衝突するまで、もしくはツモのブロックが一つでもフィールドに配置したブロックに対して下方向に衝突するまでツモを落下し、落下した先でブロックを即配置をする。このような動作を、落ちものパズルゲームにおいて「ハードドロップ」と呼称する。以下のソースコードは setHardDrop の記述例である。

```
1
2 --ハードドロップを使用する
3 setHardDrop( true )
```

A.1.9 setTumo(array , pattern , returnnum)

ゲーム中に使用するツモの形状を作成する。array にはツモの形状を整数値で表した 2 次元配列を指定する。2 次元配列にて正の値を代入しておくと createBlockPattern で作成したブロックの役割識別番号をランダムで代入する。同じ数値にすると同じ役割のブロックにすることができるため、すべてばらばらの役割を持つブロックとして出現させたい場合はそれぞれ違う数値にする必要がある。2 次元配列にて数値を代入する際、createBlockPattern で作成したブロックの種類以上の数値は無視する。負の値を 2 次元配列に代入しておくと、出現するブロックの役割を固定することが可能である。例として、-2 と指定した部分は、役割識別番号が 2 のブロックが、-3 と指定した部分は、役割識別番号が 3 のブロックが必ずその位置に出現するようになる。pattern はツモの種類を整数値で指定する。pattern が複数指定されていた場合、ゲーム中ランダムで pattern で指定した分の形状が出現する。returnnum は何番目のツモの回転形状にするかを整数値で指定する。returnnum で指定した回転形状パターンは 0 番目の形状を初期値として代入する、ゲーム上で右回転操作をした際に 0 番目の形状から 1 番目の形状となり、1 番目の回転形状時に右回転操作をすると 2 番目の回転形状、同じく右回転操作をすると 3 番目の回転形状と変化

する。returnnum で指定した回転形状パターンの数を超えると再び 0 番目の回転形状になる。左回転操作をすると右回転操作の逆の動作をする。returnnum、pattern の値は複数指定がある場合 0 から 1 ずつ加算するように指定する。以下のソースコードは setTumo の記述例である。

```
1
2  --多次元配列準備
3  --Lua の多次元配列を用意し、array に指定する。
4  for p = 1 , 4 do
5
6     arrays[p] = {}
7
8  end
9
10 --ツモの設定
11 --T 字形状のツモを作成し、ブロックの役割はバラバラにする。
12 --それぞれ違う数値を入れているが､ランダムに指定されたブロックの役割によっては、
13 --同じ役割として出現する場合もある。
14 arrays[0] = { 0 , 0 , 0 , 0 }
15 arrays[1] = { 0 , 1 , 0 , 0 }
16 arrays[2] = { 2 , 3 , 4 , 0 }
17 arrays[3] = { 0 , 0 , 0 , 0 }
18
19 setTumo( arrays , 0 , 0 ) --0回転目
20
21 arrays[0] = { 0 , 0 , 0 , 0 }
22 arrays[1] = { 0 , 2 , 0 , 0 }
23 arrays[2] = { 0 , 3 , 1 , 0 }
24 arrays[3] = { 0 , 4 , 0 , 0 }
25
26 setTumo( arrays , 0 , 1 ) --1回転目
27
28 arrays[0] = { 0 , 0 , 0 , 0 }
29 arrays[1] = { 0 , 0 , 0 , 0 }
30 arrays[2] = { 4 , 3 , 2 , 0 }
31 arrays[3] = { 0 , 1 , 0 , 0 }
32
33 setTumo( arrays , 0 , 2 ) --2回転目
34
35 arrays[0] = { 0 , 0 , 0 , 0 }
36 arrays[1] = { 0 , 4 , 0 , 0 }
37 arrays[2] = { 1 , 3 , 0 , 0 }
38 arrays[3] = { 0 , 2 , 0 , 0 }
39
40 setTumo( arrays , 0 , 3 ) --3回転目
41
42 --複数のツモ形状を作り､ブロックの役割は固定する
43 --L 字型の形状として 2 つめの形状を作成。
44 --負の値(-2)を指定しているため､必ずブロックの役割識別番号が 2のものが出現する
45 arrays[0] = { 0 , 0 , 0 , 0 }
46 arrays[1] = { 0 , 0 , -2 , 0 }
47 arrays[2] = { -2 , -2 , -2 , 0 }
48 arrays[3] = { 0 , 0 , 0 , 0 }
49
50 setTumo( arrays , 1 , 0 ) --0回転目
51
```

```
52 arrays[0] = { 0 , 0 , 0 , 0 }
53 arrays[1] = { 0 , -2 , 0 , 0 }
54 arrays[2] = { 0 , -2 , 0 , 0 }
55 arrays[3] = { 0 , -2 , -2 , 0 }
56
57 setTumo( arrays , 1 , 1 ) --1回転目
58
59 arrays[0] = { 0 , 0 , 0 , 0 }
60 arrays[1] = { 0 , 0 , 0 , 0 }
61 arrays[2] = { -2 , -2 , -2 , 0 }
62 arrays[3] = { -2 , 0 , 0 , 0 }
63
64 setTumo( arrays , 1 , 2 ) --2回転目
65
66 arrays[0] = { 0 , 0 , 0 , 0 }
67 arrays[1] = { -2 , -2 , 0 , 0 }
68 arrays[2] = { 0 , -2 , 0 , 0 }
69 arrays[3] = { 0 , -2 , 0 , 0 }
70
71 setTumo( arrays , 1 , 3 ) --3回転目
72
73 --数値が同じなため、同じ役割のブロックが出現するが、正の値を代入したため色は出現ごとにランダムとなる。
74 --回転しても形状が変わらない場合は一つのみ登録すればよい。
75 arrays[0] = { 0 , 0 , 0 , 0 }
76 arrays[1] = { 0 , 2 , 2 , 0 }
77 arrays[2] = { 0 , 2 , 2 , 0 }
78 arrays[3] = { 0 , 0 , 0 , 0 }
79
80 setTumo( arrays , 2 , 0 ) --0回転目
81
82 --入れ替え型のツモの作成例
83 arrays[0] = { 0 , 0 , 0 , 0 }
84 arrays[1] = { 0 , 1 , 0 , 0 }
85 arrays[2] = { 0 , 2 , 0 , 0 }
86 arrays[3] = { 0 , 3 , 0 , 0 }
87
88 setTumo( arrays , 3 , 0 ) --0回転目
89
90 arrays[0] = { 0 , 0 , 0 , 0 }
91 arrays[1] = { 0 , 3 , 0 , 0 }
92 arrays[2] = { 0 , 1 , 0 , 0 }
93 arrays[3] = { 0 , 2 , 0 , 0 }
94
95 setTumo( arrays , 3 , 1 ) --1回転目
96
97 arrays[0] = { 0 , 0 , 0 , 0 }
98 arrays[1] = { 0 , 2 , 0 , 0 }
99 arrays[2] = { 0 , 3 , 0 , 0 }
100 arrays[3] = { 0 , 1 , 0 , 0 }
101
102 setTumo( arrays , 3 , 2 ) --2回転目
103
104 --変化型ツモの作成例
105 arrays[0] = { 0 , 0 , 0 , 0 }
106 arrays[1] = { 0 , 1 , 1 , 0 }
107 arrays[2] = { 0 , 1 , 1 , 0 }
108 arrays[3] = { 0 , 0 , 0 , 0 }
109
```

```
110 setTumo( arrays , 4 , 0 ) --0回転目
111
112 arrays[0] = { 0 , 0 , 0 , 0 }
113 arrays[1] = { 0 , 2 , 2 , 0 }
114 arrays[2] = { 0 , 2 , 2 , 0 }
115 arrays[3] = { 0 , 0 , 0 , 0 }
116
117 setTumo( arrays , 4 , 1 ) --1回転目
118
119 arrays[0] = { 0 , 0 , 0 , 0 }
120 arrays[1] = { 0 , 3 , 3 , 0 }
121 arrays[2] = { 0 , 3 , 3 , 0 }
122 arrays[3] = { 0 , 0 , 0 , 0 }
123
124 setTumo( arrays , 4 , 2 ) --2回転目
```

A.2 取得・制御系

取得・制御系は、主にゲームの状況取得を制御を行う機能である。初期設定系とは違い、取得・制御系は update メソッド内で処理を行う。記述方法は各機能の記述例を参考してほしい。

A.2.1 getFieldWidth()

フィールドの横幅を取得する。戻り値は setFieldSize の x で指定した横幅の整数値となる。以下のソースコードは getFieldWidth の記述例である。

```
1
2 --フィールドの横幅分の回数ループを回す
3 function update()
4   for x = 0 , getFieldWidth() do
5     i ++ ;
6   end
7 end
```

実行結果は、setFieldSize の x に 6 を入れていた場合、変数 i は 6 となる。

A.2.2 getFieldHeight()

フィールドの縦幅を取得する。戻り値は setFieldSize の y で指定した縦幅の整数値となる。以下のソースコードは getFieldWidth の記述例である。

```
1
```

```
2 --フィールドの横幅分の回数ループを回す
3 function update()
4   for x = 0 , getFieldHeight() do
5     i ++ ;
6   end
7 end
```

実行結果は、setFieldSize の y に 12 を入れていた場合、変数 i は 12 となる。

A.2.3 getFieldBlock(x , y)

フィールド上の指定座標に配置したブロックの役割識別番号を取得する。x は取得したいフィールド上の横座標を整数値で指定する。y は取得したいフィールド上の縦座標を整数値で指定する。返り値はフィールド上の指定した座標のブロックの役割識別番号を返し、フィールド上の指定座標にブロックが存在していなかった場合返り値は 0 を返す。以下のソースコードは getFieldBlock の記述例である。

```
1
2 --フィールド上一番左下の座標に配置したブロックの役割識別番号を取得する
3 function update()
4   color = getFieldBlock( 0 , getFieldHeight()-1 )
5 end
```

実行結果は、変数 color にフィールド上一番左下に配置したブロックの役割識別番号を代入したものとなる。

A.2.4 getComboBlockNum(x , y)

フィールド上の指定座標に配置したブロックに隣接する同役割のブロックがいくつ連結しているかを取得する。x にはフィールド上の横座標の整数値を指定する。y にはフィールド上の縦座標の整数値を指定する。フィールド上の指定座標にブロックが存在していなかった場合返り値は 0 を返す。以下のソースコードは getComboBlockNum の記述例である。

```
1
2 --フィールド上一番左下の座標に配置した同役割ブロックがいくつ連結しているかを取得する
3 function update()
4   combo = getComboBlockNum( 0 , getFieldHeight()-1 )
```

```
5 end
```

実行結果は、変数 combo にフィールド上一番左下に配置したブロックに隣接する同役割のブロックの数を代入したものとなる。

A.2.5 getComboSeriesHorizontalBlockNum(x , y)

フィールド上の指定座標に配置したブロックと同役割のブロックが横の直線上にいくつ並んでいるかを取得する。x にはフィールド上の横座標の整数値を指定する。y にはフィールド上の縦座標の整数値を指定する。フィールド上の指定座標にブロックが存在していなかった場合返り値は 0 を返す。以下のソースコードは getComboSeriesHorizontalBlockNum の記述例である。

```
1
2 --フィールド上一番左下の座標に配置したブロックと同役割のブロックが横の直線上にいくつ並んでいるかを取得する
3 function update()
4   combo = getComboSeriesHorizontalBlockNum( 0 , getFieldHeight()-1 )
5 end
```

実行結果は、変数 combo にフィールド上一番左下に配置したブロックと同役割のブロックが横の直線上に並んでいる数が代入される。

A.2.6 getComboSeriesVerticalBlockNum(x , y)

フィールド上の指定座標に配置したブロックと同役割のブロックが縦の直線上にいくつ並んでいるかを取得する。x にはフィールド上の横座標の整数値を指定する。y にはフィールド上の縦座標の整数値を指定する。フィールド上の指定座標にブロックが存在していなかった場合返り値は 0 を返す。以下のソースコードは getComboSeriesVerticalBlockNum の記述例である。

```
1
2 --フィールド上一番左下の座標に配置したブロックと同役割のブロックが縦の直線上にいくつ並んでいるかを取得する
3 function update()
4   combo = getComboSeriesVerticalBlockNum( 0 , getFieldHeight()-1 )
5 end
```

実行結果は、変数 combo にフィールド上一番左下に配置したブロックと同役割のブロックが縦の直線上に並んでいる数が代入される。

A.2.7 getComboSeriesRightUpDiagonalBlockNum(x , y)

フィールド上の指定座標に配置したブロックと同役割のブロックが右上がり斜めの直線上にいくつ並んでいるかを取得する。x にはフィールド上の横座標の整数値を指定する。y にはフィールド上の縦座標の整数値を指定する。フィールド上の指定座標にブロックが存在していなかった場合返り値は 0 を返す。以下のソースコードは getComboSeriesRightUpDiagonalBlockNum の記述例である。

```
1
2 --フィールド上一番左下の座標に配置したブロックと同役割のブロックが右上がり斜めの直線上にいくつ並んでいる
      かを取得する
3 function update()
4   combo = getComboSeriesRightUpDiagonalBlockNum( 0 , getFieldHeight()-1 )
5 end
```

実行結果は、変数 combo にフィールド上一番左下に配置したブロックと同役割のブロックが右上がり斜めの直線上に並んでいる数が代入される。

A.2.8 getComboSeriesLeftUpDiagonalBlockNum(x , y)

フィールド上の指定座標に配置したブロックと同役割のブロックが左上がり斜めの直線上にいくつ並んでいるかを取得する。x にはフィールド上の横座標の整数値を指定する。y にはフィールド上の縦座標の整数値を指定する。フィールド上の指定座標にブロックが存在していなかった場合返り値は 0 を返す。以下のソースコードは getComboSeriesLeftUpDiagonalBlockNum の記述例である。

```
1
2 --フィールド上一番右下の座標に配置したブロックと同役割のブロックが左上がり斜めの直線上にいくつ並んでいる
      かを取得する
3 function update()
4   combo = getComboSeriesLeftUpDiagonalBlockNum( getFieldWidth()-1 , getFieldHeight
        ()-1 )
```

```
5 end
```

実行結果は、変数 combo にフィールド上一番右下に配置したブロックと同役割のブロックが左上がり斜めの直線上に並んでいる数が代入される。

A.2.9 getBlockDownFlag(x , y)

フィールド上の指定座標に配置したブロックが落下中かを取得する。x にはフィールド上の横座標の整数値を指定する。y にはフィールド上の縦座標の整数値を指定する。返り値は、指定座標のブロックが落下中であった場合 true を返し、落下中でない場合またはブロックが指定座標に存在しない場合は false を返す。以下のソースコードは getBlockDownFlag の記述例である。

```
1
2  --フィールドに存在している個別のブロックが落下中かどうかを調べる
3  function update()
4
5    for x = 0 , getFieldWidth() do
6      for y = 0 , getFieldHeight() do
7        downflag = getBlockDownFlag( x , y )
8      end
9    end
10
11 end
```

A.2.10 getAllBlockDownFlag()

フィールド上に配置したすべてのブロックが落下中かを取得する。返り値は、フィールド上に配置したブロックのどれか一つでも落下中であった場合 true を返し、一つも落下中ではないまたはフィールド上にブロックが存在しない場合は false を返す。以下のソースコードは getAllBlockDownFlag の記述例である。

```
1
2  --フィールドに存在している個別のブロックが落下中かどうかを調べる
3  function update()
4     downflag = getAllBlockDownFlag()
5  end
```

A.2.11 setBlockDeleteFlag(x , y)

フィールド上の指定座標に配置したブロックに対して消滅命令を出す。x にはフィールド上の横座標の整数値を指定する。y にはフィールド上の縦座標の整数値を指定する。指定座標にブロックが存在していなかった場合は処理を無視する。消滅命令を出したブロックは消滅条件が成立したとし、直ちに消滅状態へ移行する。以下のソースコードは setBlockDeleteFlag の記述例である。

```
1
2  --同役割のブロックが 4つ以上連結していた場合、そのブロックに対して消滅命令を出す。
3  function update()
4
5    for x = 0 , getFieldWidth() do
6      for y = 0 , getFieldHeight() do
7        if getComboBlockNum( x , y ) >= 4 then --4つ以上連結していたら
8          setBlockDeleteFlag( x , y ) --消滅命令を送る
9        end
10     end
11   end
12
13 end
```

A.2.12 setBlockGroupDeleteFlag(x , y)

フィールド上の指定座標に配置したブロックと隣接しており、連結している同役割のブロックすべて対して消滅命令を出す。x にはフィールド上の横座標の整数値を指定する。y にはフィールド上の縦座標の整数値を指定する。指定座標にブロックが存在していなかった場合は処理を無視する。消滅命令を出したブロックは消滅条件が成立したとし、直ちに消滅状態へ移行する。以下のソースコードは setBlockGroupDeleteFlag の記述例である。

```
1
2  --同役割のブロックが 4つ以上連結していた場合、そのブロックに対して消滅命令を出す。
3  function update()
4
5    for x = 0 , getFieldWidth() do
6      for y = 0 , getFieldHeight() do
7        if getComboBlockNum( x , y ) >= 4 then --4つ以上連結していたら
8          setBlockGroupDeleteFlag( x , y ) --消滅命令を送る
9        end
10     end
```

```
11    end
12
13  end
```

A.2.13 setBlockDownCellNum(x , y , downcell)

フィールド上の指定座標に配置したブロックに対して任意段数分落下命令を出す。x にはフィールド上の横座標の整数値を指定する。y にはフィールド上の縦座標の整数値を指定する。downcell には落下する段数を整数値で指定する。downcell に指定した値が実際に落下可能段数を超えていた場合、落下可能限界まで落下した後残った段数分は処理を無視する。また、指定座標にブロックが存在していなかった場合も処理を無視する。以下のソースコードは setBlockDownCell の記述例である。

```
1
2  --下側に隙間ができないように敷き詰められるように落下をさせる
3  function update()
4
5    for x = 0 , getFieldWidth() do
6      for y = 0 , getFieldHeight() do
7        setBlockDownCellNum( x , y , getFieldHeight() )
8      end
9    end
10
11 end
```

A.2.14 setColorBlock(x , y , num)

フィールド上の指定座標に対して指定ブロックを配置する。x にはフィールド上の横座標の整数値を指定する。y にはフィールド上の縦座標の整数値を指定する。num には配置したいブロックの役割識別番号を整数値で指定する。以下のソースコードは setColorBlock の記述例である。

```
1
2  --消滅方法変化型のブロックの変化を実装する
3  function update()
4
5  for x = 0 , getFieldWidth() do
6    for y = 0 , getFieldHeight() do
7
8      --ブロック役割識別番号が 1であり 4つ以上同役割ブロックが連結していた場合
9      if getComboBlockNum( x , y ) >= 4 and getFieldBlock( x , y ) == 1 then
```

```
10      setColorBlock( x , y , 2 ) --ブロックを 2の役割の物へ変化させる
11    end
12
13    --ブロック役割識別番号が 2であり 5つ以上同役割ブロックが連結していた場合
14    if getComboBlockNum( x , y ) >= 5 and getFieldBlock( x , y ) == 2 then
15      setColorBlock( x , y , 3 ) --ブロックを 3の役割の物へ変化させる
16    end
17
18    --ブロック役割識別番号が 3であり 6つ以上同役割ブロックが連結していた場合
19    if getComboBlockNum( x , y ) >= 6 and getFieldBlock( x , y ) == 3 then
20      setBlockDeleteFlag( x , y ) --消滅命令を送る
21    end
22  end
23 end
24
25 end
```

A.3 描画系機能

A.3.1 setBlockGraph(num , graphpass)

指定役割のブロックに対して画像ファイルを指定できる。num は画像を適用したいブロックの役割識別番号を整数値で指定する。num が 0 の場合は処理を無視する。graphpass は適用したい画像のパスを指定する。パスは exe ファイルから見た相対パスまたは絶対パスの指定をする。画像ファイルは n*n の正方形をした解像度のものが好ましい。以下のソースコードは setBlockGraph の記述例である。

```
1
2 --1の役割をもつブロックに対して画像を適用する
3 setBlockGraph( 1 , "block1.png" )
4
5 --2の役割をもつブロックに対して画像を適用する
6 setBlockGraph( 2 , "block2.png" )
7
8 --3の役割をもつブロックに対して画像を適用する
9 setBlockGraph( 3 , "block3.png" )
```

A.3.2 setBlockForm(num , formnum)

指定役割のブロックに対して正方形や円などのプリミティブな形状を指定できる。num は形状を適用したいブロックの役割識別番号を整数値で指定する。num が 0 の場合は処理を無視す

る。formnum は適用したい形状の ID を数値で指定する。formnum で適用できる形状は 4 種類あり、0 を指定すると塗りつぶされていない円、1 を指定すると塗りつぶされた円、2 を指定すると塗りつぶされていない正方形、3 を指定すると塗りつぶされた正方形が適用される。なお、setBlockGraph を用いて画像を適用した場合は、この機能を無視する。以下のソースコードは setBlockForm の記述例である。

```
1
2 --1の役割をもつブロックに対して塗りつぶされた円を形状として指定する
3 setBlockForm( 1 , 1 )
4
5 --2の役割をもつブロックに対して塗りつぶされていない正方形を形状として指定する
6 setBlockForm( 2 , 2 )
7
8 --3の役割をもつブロックに対して塗りつぶされた正方形を形状として指定する
9 setBlockForm( 3 , 3 )
```

A.3.3 setBlockFormColor(num , r , g , b)

指定役割のブロックに対して色を決定する。num には色を指定したいブロックの役割識別番号を整数値で指定する。r、g、b はそれぞれ赤、青、緑の色成分であり、0〜255 の範囲を整数値で指定する。なお、この機能は setBlockForm が有効なブロックにしか適用できない。以下のソースコードは setBlockFormColor の記述例である。

```
1
2 --1の役割をもつブロックに対して塗りつぶされた円を形状として指定する
3 setBlockForm( 1 , 1 )
4
5 --2の役割をもつブロックに対して塗りつぶされていない正方形を形状として指定する
6 setBlockForm( 2 , 2 )
7
8 --3の役割をもつブロックに対して塗りつぶされた正方形を形状として指定する
9 setBlockForm( 3 , 3 )
10
11 --1の役割を持つブロックを赤色にする
12 setBlockFormColor( 1 , 255 , 0 , 0 )
13
14 --2の役割を持つブロックを緑色にする
15 setBlockFormColor( 2 , 0 , 255 , 0 )
16
17 --3の役割を持つブロックを青色にする
18 setBlockFormColor( 3 , 0 , 0 , 255 )
```

A.3.4 setVerticalSmoothFlag(flag)

縦方向に対して滑らかに座標を補完した表示をするかの有無を決定する。flag には使用する場合は true、使用しない場合は false を指定する。flag を true とした場合、フィールドのマスごとの描画ではなく、落下中などマスの間にいる表現がされる。以下のソースコードは setVerticalSmoothFlag の記述例である。

```
1
2 --縦方向の描画位置補完を有効にする
3 setVerticalSmoothFlag( true )
```

A.3.5 setHorizontalSmoothFlag(flag)

横方向に対して滑らかに座標を補完した表示をするかの有無を決定する。flag には使用する場合は true、使用しない場合は false を指定する。flag を true とした場合、フィールドのマスごとの描画ではなく、横移動中などマスの間にいる表現がされる。以下のソースコードは setHorizontalSmoothFlag の記述例である。

```
1
2 --横方向の描画位置補完を有効にする
3 setHorizontalSmoothFlag( true )
```

A.4 その他機能

A.4.1 setTumoDownSpeed(speed)

ツモが自然落下するスピードを決定する。speed にはツモが自然落下するスピードを小数点数値で指定する。1.0 を指定すると画面が 1 度更新される度フィールド上の 1 マス分進む。0.0 を指定すると自然落下が無効となる。以下のソースコードは setTumoDownSpeed の記述例である。

```
1
2 --ツモの自然落下の速度を決定する
3 setTumoDownSpeed( 0.01 )
```

A.4.2 setTumoMoveWait(wait)

横方向キーを押しっぱなしにした際の高速移動中に、1 回移動するたびにかかるウェイトを設定する。wait にはウェイトをどれくらいかけるかの値を整数値で指定する。ウェイトの値が小さければ小さいほど押しっぱなしにした際の移動が速くなる。以下のソースコードは setTumoMoveWait の記述例である。

```
1
2 --ツモの高速移動中のウェイトを 3に指定する
3 setTumoDownSpeed( 3 )
```

A.4.3 setTumoMoveStartWait(frame)

横方向キーを押しっぱなしにした際の高速移動に移行するまでのウェイトを設定する。wait にはウェイトをどれくらいかけるかの値を整数値で指定する。ウェイトの値が小さければ小さいほど高速移動に移行するのが早くなる。以下のソースコードは setTumoMoveStartWait の記述例である。

```
1
2 --ツモの高速移動に移行するまでのウェイトを 10に指定する
3 setTumoMoveStartWait( 10 )
```

付録 B

サンプルコード

B.1 ぷよぷよのサンプルコード

```
1
2  setFieldSize ( 6 , 13 ) --パズルのプレイフィールドのサイズ指定
3  blockSize ( 30 )  --ブロックの描画サイズ指定
4  setSuperRotation( false ) --回転補正を無効にする
5  setQuickTurn( true ) --クイックターンを有効にする
6  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)
7  setHardDrop( false ) --ハードドロップを無効にする
8
9  --ブロックを作成
10 createBlockPattern( 1 )
11 createBlockPattern( 2 )
12 createBlockPattern( 3 )
13 createBlockPattern( 4 )
14 createBlockPattern( 5 )
15
16 --ブロックの形状を指定
17 setBlockForm( 1 , 1 )
18 setBlockForm( 2 , 1 )
19 setBlockForm( 3 , 1 )
20 setBlockForm( 4 , 1 )
21 setBlockForm( 5 , 1 )
22
23 --ブロックに色を指定
24 setBlockFormColor( 1 , 255 , 0 , 0 )
25 setBlockFormColor( 2 , 0 , 255 , 0 )
26 setBlockFormColor( 3 , 80 , 80 , 255 )
27 setBlockFormColor( 4 , 180 , 0 , 255 )
28 setBlockFormColor( 5 , 255 , 255 , 0 )
29
30 --多次元配列準備
31 arrays = {}
32 for p = 1 , 3 do
33
34    arrays[p] = {}
35
36 end
37
38 --ツモの設定
39 arrays[0] = { 0 , 1 , 0 }
40 arrays[1] = { 0 , 2 , 0 }
41 arrays[2] = { 0 , 0 , 0 }
42
43 setTumo( arrays , 0 , 0 )
44
45 arrays[0] = { 0 , 0 , 0 }
46 arrays[1] = { 0 , 2 , 1 }
47 arrays[2] = { 0 , 0 , 0 }
48
49 setTumo( arrays , 0 , 1 )
50
51 arrays[0] = { 0 , 0 , 0 }
52 arrays[1] = { 0 , 2 , 0 }
53 arrays[2] = { 0 , 1 , 0 }
54
55 setTumo( arrays , 0 , 2 )
56
```

```
arrays[0] = { 0 , 0 , 0 }
arrays[1] = { 1 , 2 , 0 }
arrays[2] = { 0 , 0 , 0 }

setTumo( arrays , 0 , 3 )


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
```

B.2 テトリスのサンプルコード

```

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

```
16 createBlockPattern( 7 )
17
18 --ブロックの形状を指定
19 setBlockForm( 1 , 3 )
20 setBlockForm( 2 , 3 )
21 setBlockForm( 3 , 3 )
22 setBlockForm( 4 , 3 )
23 setBlockForm( 5 , 3 )
24 setBlockForm( 6 , 3 )
25 setBlockForm( 7 , 3 )
26
27 --ブロックに色を指定
28 setBlockFormColor( 1 , 255 , 40 , 255 )
29 setBlockFormColor( 2 , 255 , 155 , 40 )
30 setBlockFormColor( 3 , 40 , 40 , 255 )
31 setBlockFormColor( 4 , 130 , 255 , 40 )
32 setBlockFormColor( 5 , 255 , 40 , 40 )
33 setBlockFormColor( 6 , 255 , 255 , 0 )
34 setBlockFormColor( 7 , 40 , 155 , 255 )
35
36 --多次元配列準備
37 arrays = {}
38 for p = 1 , 4 do
39
40    arrays[p] = {}
41
42 end
43
44
45 --ツモの設定
46 --T=======================================
47 arrays[0] = { 0 , 0 , 0 , 0 }
48 arrays[1] = { 0 , -1 , 0 , 0 }
49 arrays[2] = { -1 , -1 , -1 , 0 }
50 arrays[3] = { 0 , 0 , 0 , 0 }
51
52 setTumo( arrays , 0 , 0 )
53
54 arrays[0] = { 0 , 0 , 0 , 0 }
55 arrays[1] = { 0 , -1 , 0 , 0 }
56 arrays[2] = { 0 , -1 , -1 , 0 }
57 arrays[3] = { 0 , -1 , 0 , 0 }
58
59 setTumo( arrays , 0 , 1 )
60
61 arrays[0] = { 0 , 0 , 0 , 0 }
62 arrays[1] = { 0 , 0 , 0 , 0 }
63 arrays[2] = { -1 , -1 , -1 , 0 }
64 arrays[3] = { 0 , -1 , 0 , 0 }
65
66 setTumo( arrays , 0 , 2 )
67
68 arrays[0] = { 0 , 0 , 0 , 0 }
69 arrays[1] = { 0 , -1 , 0 , 0 }
70 arrays[2] = { -1 , -1 , 0 , 0 }
71 arrays[3] = { 0 , -1 , 0 , 0 }
72
73 setTumo( arrays , 0 , 3 )
```

```
74
75 --L=======================================
76 arrays[0] = { 0 , 0 , 0 , 0 }
77 arrays[1] = { 0 , 0 , -2 , 0 }
78 arrays[2] = { -2 , -2 , -2 , 0 }
79 arrays[3] = { 0 , 0 , 0 , 0 }
80
81 setTumo( arrays , 1 , 0 )
82
83 arrays[0] = { 0 , 0 , 0 , 0 }
84 arrays[1] = { 0 , -2 , 0 , 0 }
85 arrays[2] = { 0 , -2 , 0 , 0 }
86 arrays[3] = { 0 , -2 , -2 , 0 }
87
88 setTumo( arrays , 1 , 1 )
89
90 arrays[0] = { 0 , 0 , 0 , 0 }
91 arrays[1] = { 0 , 0 , 0 , 0 }
92 arrays[2] = { -2 , -2 , -2 , 0 }
93 arrays[3] = { -2 , 0 , 0 , 0 }
94
95 setTumo( arrays , 1 , 2 )
96
97 arrays[0] = { 0 , 0 , 0 , 0 }
98 arrays[1] = { -2 , -2 , 0 , 0 }
99 arrays[2] = { 0 , -2 , 0 , 0 }
100 arrays[3] = { 0 , -2 , 0 , 0 }
101
102 setTumo( arrays , 1 , 3 )
103
104 --J=======================================
105 arrays[0] = { 0 , 0 , 0 , 0 }
106 arrays[1] = { -3 , 0 , 0 , 0 }
107 arrays[2] = { -3 , -3 , -3 , 0 }
108 arrays[3] = { 0 , 0 , 0 , 0 }
109
110 setTumo( arrays , 2 , 0 )
111
112 arrays[0] = { 0 , 0 , 0 , 0 }
113 arrays[1] = { 0 , -3 , -3 , 0 }
114 arrays[2] = { 0 , -3 , 0 , 0 }
115 arrays[3] = { 0 , -3 , 0 , 0 }
116
117 setTumo( arrays , 2 , 1 )
118
119 arrays[0] = { 0 , 0 , 0 , 0 }
120 arrays[1] = { 0 , 0 , 0 , 0 }
121 arrays[2] = { -3 , -3 , -3 , 0 }
122 arrays[3] = { 0 , 0 , -3 , 0 }
123
124 setTumo( arrays , 2 , 2 )
125
126 arrays[0] = { 0 , 0 , 0 , 0 }
127 arrays[1] = { 0 , -3 , 0 , 0 }
128 arrays[2] = { 0 , -3 , 0 , 0 }
129 arrays[3] = { -3 , -3 , 0 , 0 }
130
131 setTumo( arrays , 2 , 3 )
```

```
132
133 --S=======================================
134
135 arrays[0] = { 0 , 0 , 0 , 0 }
136 arrays[1] = { 0 , -4 , -4 , 0 }
137 arrays[2] = { -4 , -4 , 0 , 0 }
138 arrays[3] = { 0 , 0 , 0 , 0 }
139
140 setTumo( arrays , 3 , 0 )
141
142 arrays[0] = { 0 , 0 , 0 , 0 }
143 arrays[1] = { 0 , -4 , 0 , 0 }
144 arrays[2] = { 0 , -4 , -4 , 0 }
145 arrays[3] = { 0 , 0 , -4 , 0 }
146
147 setTumo( arrays , 3 , 1 )
148
149 arrays[0] = { 0 , 0 , 0 , 0 }
150 arrays[1] = { 0 , 0 , 0 , 0 }
151 arrays[2] = { 0 , -4 , -4 , 0 }
152 arrays[3] = { -4 , -4 , 0 , 0 }
153
154 setTumo( arrays , 3 , 2 )
155
156 arrays[0] = { 0 , 0 , 0 , 0 }
157 arrays[1] = { -4 , 0 , 0 , 0 }
158 arrays[2] = { -4 , -4 , 0 , 0 }
159 arrays[3] = { 0 , -4 , 0 , 0 }
160
161 setTumo( arrays , 3 , 3 )
162
163 --Z=======================================
164
165 arrays[0] = { 0 , 0 , 0 , 0 }
166 arrays[1] = { -5 , -5 , 0 , 0 }
167 arrays[2] = { 0 , -5 , -5 , 0 }
168 arrays[3] = { 0 , 0 , 0 , 0 }
169
170 setTumo( arrays , 4 , 0 )
171
172 arrays[0] = { 0 , 0 , 0 , 0 }
173 arrays[1] = { 0 , 0 , -5 , 0 }
174 arrays[2] = { 0 , -5 , -5 , 0 }
175 arrays[3] = { 0 , -5 , 0 , 0 }
176
177 setTumo( arrays , 4 , 1 )
178
179 arrays[0] = { 0 , 0 , 0 , 0 }
180 arrays[1] = { 0 , 0 , 0 , 0 }
181 arrays[2] = { -5 , -5 , 0 , 0 }
182 arrays[3] = { 0 , -5 , -5 , 0 }
183
184 setTumo( arrays , 4 , 2 )
185
186 arrays[0] = { 0 , 0 , 0 , 0 }
187 arrays[1] = { 0 , -5 , 0 , 0 }
188 arrays[2] = { -5 , -5 , 0 , 0 }
189 arrays[3] = { -5 , 0 , 0 , 0 }
```

```
190
191 setTumo( arrays , 4 , 3 )
192
193 --O=======================================
194 arrays[0] = { 0 , 0 , 0 , 0 }
195 arrays[1] = { 0 , -6 , -6 , 0 }
196 arrays[2] = { 0 , -6 , -6 , 0 }
197 arrays[3] = { 0 , 0 , 0 , 0 }
198
199 setTumo( arrays , 5 , 0 )
200
201
202 --I=======================================
203 arrays[0] = { 0 , 0 , 0 , 0 }
204 arrays[1] = { -7 , -7 , -7 , -7 }
205 arrays[2] = { 0 , 0 , 0 , 0 }
206 arrays[3] = { 0 , 0 , 0 , 0 }
207
208 setTumo( arrays , 6 , 0 )
209
210 arrays[0] = { 0 , 0 , -7 , 0 }
211 arrays[1] = { 0 , 0 , -7 , 0 }
212 arrays[2] = { 0 , 0 , -7 , 0 }
213 arrays[3] = { 0 , 0 , -7 , 0 }
214
215 setTumo( arrays , 6 , 1 )
216
217 arrays[0] = { 0 , 0 , 0 , 0 }
218 arrays[1] = { 0 , 0 , 0 , 0 }
219 arrays[2] = { -7 , -7 , -7 , -7 }
220 arrays[3] = { 0 , 0 , 0 , 0 }
221
222 setTumo( arrays , 6 , 2 )
223
224 arrays[0] = { 0 , -7 , 0 , 0 }
225 arrays[1] = { 0 , -7 , 0 , 0 }
226 arrays[2] = { 0 , -7 , 0 , 0 }
227 arrays[3] = { 0 , -7 , 0 , 0 }
228
229 setTumo( arrays , 6 , 3 )
230
231
232 function update()
233
234    --落下させる量を保持する変数
235    downnum = 0
236
237    y = getFieldHeight()
238    for p = 0 , getFieldHeight() do
239
240       y = y - 1
241
242       checknum = 0
243
244       --横列にブロックが何個いるか捜査
245       for x = 0 , getFieldWidth() do
246
247          if getFieldBlock( x , y ) > 0 and getFieldBlock( x , y ) < 100 then
```

```
248             checknum = checknum + 1
249
250           end
251
252       end
253
254       --横列にブロックが 10個以上いた場合消去処理をする
255       if checknum >= 10 then
256
257           checknum = 10
258
259           downnum = downnum + 1
260
261           for x = 0 , getFieldWidth() do
262             setBlockDeleteFlag( x , y )
263           end
264
265       else
266
267           --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
268           for x = 0 , getFieldWidth() do
269
270             setBlockDownCellNum( x , y , downnum )
271
272           end
273
274       end
275
276     end
277
278 end
```

B.3 コラムスのサンプルコード

```
1
2  setFieldSize ( 6 , 13 ) --パズルのプレイフィールドのサイズ指定
3  blockSize ( 30 )  --ブロックの描画サイズ指定
4  setSuperRotation( false ) --回転補正を無効にする
5  setQuickTurn( false ) --クイックターンを無効にする
6  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)
7  setHardDrop( false ) --ハードドロップを無効にする
8
9  --ブロックを作成
10 createBlockPattern( 1 )
11 createBlockPattern( 2 )
12 createBlockPattern( 3 )
13 createBlockPattern( 4 )
14 createBlockPattern( 5 )
15
16 --ブロックの形状を指定
17 setBlockForm( 1 , 1 )
18 setBlockForm( 2 , 1 )
19 setBlockForm( 3 , 1 )
20 setBlockForm( 4 , 1 )
21 setBlockForm( 5 , 1 )
```

```
22
23 --ブロックに色を指定
24 setBlockFormColor( 1 , 255 , 0 , 0 )
25 setBlockFormColor( 2 , 0 , 255 , 0 )
26 setBlockFormColor( 3 , 80 , 80 , 255 )
27 setBlockFormColor( 4 , 180 , 0 , 255 )
28 setBlockFormColor( 5 , 255 , 255 , 0 )
29
30 --多次元配列準備
31 arrays = {}
32 for p = 1 , 3 do
33
34    arrays[p] = {}
35
36 end
37
38 --ツモの設定
39 arrays[0] = { 0 , 1 , 0 }
40 arrays[1] = { 0 , 2 , 0 }
41 arrays[2] = { 0 , 3 , 0 }
42
43 setTumo( arrays , 0 , 0 )
44
45 arrays[0] = { 0 , 3 , 0 }
46 arrays[1] = { 0 , 1 , 0 }
47 arrays[2] = { 0 , 2 , 0 }
48
49 setTumo( arrays , 0 , 1 )
50
51 arrays[0] = { 0 , 2 , 0 }
52 arrays[1] = { 0 , 3 , 0 }
53 arrays[2] = { 0 , 1 , 0 }
54
55 setTumo( arrays , 0 , 2 )
56
57
58 function update()
59
60
61    --落下させる量を保持する変数
62    downnum = 0
63
64    y = getFieldHeight()
65    for p = 0 , getFieldHeight() do
66
67       y = y - 1
68
69       for x = 0 , getFieldWidth() do
70
71          --常に落下をさせる
72          setBlockDownCellNum( x , y , getFieldHeight() )
73
74          --落下中のブロックがいなかったら
75          if getFieldBlock( x , y ) < 100 and getAllBlockDownFlag() == false then
76         --3つ以上直線状に並んでいたら消滅させる
77         if getComboSeriesHorizontalBlockNum( x , y ) >= 3 or
               getComboSeriesVerticalBlockNum( x , y ) >= 3 or
               getComboSeriesRightUpDiagonalBlockNum( x , y ) >= 3 or
```

```
            getComboSeriesLeftUpDiagonalBlockNum( x , y ) >= 3 then
78            setBlockDeleteFlag( x , y ) --ブロックを消滅させる
79        end
80
81         end
82
83      end
84
85   end
86
87 end
```

B.4 ぷよぷよとテトリスの混合ルールサンプルコード

```
1
2  setFieldSize ( 10 , 20 ) --パズルのプレイフィールドのサイズ指定
3  blockSize ( 30 )  --ブロックの描画サイズ指定
4  setSuperRotation( true ) --回転補正を有効にする
5  setQuickTurn( false ) --クイックターンを無効にする
6  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)
7  setHardDrop( true ) --ハードドロップを有効にする
8
9  --ブロックを作成
10 createBlockPattern( 1 )
11 createBlockPattern( 2 )
12 createBlockPattern( 3 )
13 createBlockPattern( 4 )
14 createBlockPattern( 5 )
15
16
17
18 --ブロックに色を指定
19 setBlockFormColor( 1 , 150 , 40 , 255 )
20 setBlockFormColor( 2 , 255 , 255 , 0 )
21 setBlockFormColor( 3 , 255 , 0 , 0 )
22 setBlockFormColor( 4 , 0 , 255 , 0 )
23 setBlockFormColor( 5 , 40 , 40 , 255 )
24
25
26 --多次元配列準備
27 arrays = {}
28 for p = 1 , 4 do
29
30    arrays[p] = {}
31
32 end
33
34 --ツモの設定
35 --T=======================================
36 arrays[0] = { 0 , 0 , 0 , 0 }
37 arrays[1] = { 0 , 1 , 0 , 0 }
38 arrays[2] = { 2 , 3 , 4 , 0 }
39 arrays[3] = { 0 , 0 , 0 , 0 }
40
41 setTumo( arrays , 0 , 0 )
```

```
42
43 arrays[0] = { 0 , 0 , 0 , 0 }
44 arrays[1] = { 0 , 2 , 0 , 0 }
45 arrays[2] = { 0 , 3 , 1 , 0 }
46 arrays[3] = { 0 , 4 , 0 , 0 }
47
48 setTumo( arrays , 0 , 1 )
49
50 arrays[0] = { 0 , 0 , 0 , 0 }
51 arrays[1] = { 0 , 0 , 0 , 0 }
52 arrays[2] = { 4 , 3 , 2 , 0 }
53 arrays[3] = { 0 , 1 , 0 , 0 }
54
55 setTumo( arrays , 0 , 2 )
56
57 arrays[0] = { 0 , 0 , 0 , 0 }
58 arrays[1] = { 0 , 4 , 0 , 0 }
59 arrays[2] = { 1 , 3 , 0 , 0 }
60 arrays[3] = { 0 , 2 , 0 , 0 }
61
62 setTumo( arrays , 0 , 3 )
63
64 --L=======================================
65 arrays[0] = { 0 , 0 , 0 , 0 }
66 arrays[1] = { 0 , 0 , 1 , 0 }
67 arrays[2] = { 2 , 3 , 4 , 0 }
68 arrays[3] = { 0 , 0 , 0 , 0 }
69
70 setTumo( arrays , 1 , 0 )
71
72 arrays[0] = { 0 , 0 , 0 , 0 }
73 arrays[1] = { 0 , 2 , 0 , 0 }
74 arrays[2] = { 0 , 3 , 0 , 0 }
75 arrays[3] = { 0 , 4 , 1 , 0 }
76
77 setTumo( arrays , 1 , 1 )
78
79 arrays[0] = { 0 , 0 , 0 , 0 }
80 arrays[1] = { 0 , 0 , 0 , 0 }
81 arrays[2] = { 4 , 3 , 2 , 0 }
82 arrays[3] = { 1 , 0 , 0 , 0 }
83
84 setTumo( arrays , 1 , 2 )
85
86 arrays[0] = { 0 , 0 , 0 , 0 }
87 arrays[1] = { 1 , 4 , 0 , 0 }
88 arrays[2] = { 0 , 3 , 0 , 0 }
89 arrays[3] = { 0 , 2 , 0 , 0 }
90
91 setTumo( arrays , 1 , 3 )
92
93 --J=======================================
94 arrays[0] = { 0 , 0 , 0 , 0 }
95 arrays[1] = { 1 , 0 , 0 , 0 }
96 arrays[2] = { 4 , 3 , 2 , 0 }
97 arrays[3] = { 0 , 0 , 0 , 0 }
98
99 setTumo( arrays , 2 , 0 )
```

```
100 |
101 | arrays[0] = { 0 , 0 , 0 , 0 }
102 | arrays[1] = { 0 , 4 , 1 , 0 }
103 | arrays[2] = { 0 , 3 , 0 , 0 }
104 | arrays[3] = { 0 , 2 , 0 , 0 }
105 |
106 | setTumo( arrays , 2 , 1 )
107 |
108 | arrays[0] = { 0 , 0 , 0 , 0 }
109 | arrays[1] = { 0 , 0 , 0 , 0 }
110 | arrays[2] = { 2 , 3 , 4 , 0 }
111 | arrays[3] = { 0 , 0 , 1 , 0 }
112 |
113 | setTumo( arrays , 2 , 2 )
114 |
115 | arrays[0] = { 0 , 0 , 0 , 0 }
116 | arrays[1] = { 0 , 2 , 0 , 0 }
117 | arrays[2] = { 0 , 3 , 0 , 0 }
118 | arrays[3] = { 1 , 4 , 0 , 0 }
119 |
120 | setTumo( arrays , 2 , 3 )
121 |
122 | --S=======================================
123 |
124 | arrays[0] = { 0 , 0 , 0 , 0 }
125 | arrays[1] = { 0 , 2 , 1 , 0 }
126 | arrays[2] = { 4 , 3 , 0 , 0 }
127 | arrays[3] = { 0 , 0 , 0 , 0 }
128 |
129 | setTumo( arrays , 3 , 0 )
130 |
131 | arrays[0] = { 0 , 0 , 0 , 0 }
132 | arrays[1] = { 0 , 4 , 0 , 0 }
133 | arrays[2] = { 0 , 3 , 2 , 0 }
134 | arrays[3] = { 0 , 0 , 1 , 0 }
135 |
136 | setTumo( arrays , 3 , 1 )
137 |
138 | arrays[0] = { 0 , 0 , 0 , 0 }
139 | arrays[1] = { 0 , 0 , 0 , 0 }
140 | arrays[2] = { 0 , 3 , 4 , 0 }
141 | arrays[3] = { 1 , 2 , 0 , 0 }
142 |
143 | setTumo( arrays , 3 , 2 )
144 |
145 | arrays[0] = { 0 , 0 , 0 , 0 }
146 | arrays[1] = { 1 , 0 , 0 , 0 }
147 | arrays[2] = { 2 , 3 , 0 , 0 }
148 | arrays[3] = { 0 , 4 , 0 , 0 }
149 |
150 | setTumo( arrays , 3 , 3 )
151 |
152 | --Z=======================================
153 |
154 | arrays[0] = { 0 , 0 , 0 , 0 }
155 | arrays[1] = { 1 , 2 , 0 , 0 }
156 | arrays[2] = { 0 , 3 , 4 , 0 }
157 | arrays[3] = { 0 , 0 , 0 , 0 }
```

```
158 |
159 | setTumo( arrays , 4 , 0 )
160 |
161 | arrays[0] = { 0 , 0 , 0 , 0 }
162 | arrays[1] = { 0 , 0 , 1 , 0 }
163 | arrays[2] = { 0 , 3 , 2 , 0 }
164 | arrays[3] = { 0 , 4 , 0 , 0 }
165 |
166 | setTumo( arrays , 4 , 1 )
167 |
168 | arrays[0] = { 0 , 0 , 0 , 0 }
169 | arrays[1] = { 0 , 0 , 0 , 0 }
170 | arrays[2] = { 4 , 3 , 0 , 0 }
171 | arrays[3] = { 0 , 2 , 1 , 0 }
172 |
173 | setTumo( arrays , 4 , 2 )
174 |
175 | arrays[0] = { 0 , 0 , 0 , 0 }
176 | arrays[1] = { 0 , 4 , 0 , 0 }
177 | arrays[2] = { 2 , 3 , 0 , 0 }
178 | arrays[3] = { 1 , 0 , 0 , 0 }
179 |
180 | setTumo( arrays , 4 , 3 )
181 |
182 | --O=======================================
183 | arrays[0] = { 0 , 0 , 0 , 0 }
184 | arrays[1] = { 0 , 1 , 2 , 0 }
185 | arrays[2] = { 0 , 4 , 3 , 0 }
186 | arrays[3] = { 0 , 0 , 0 , 0 }
187 |
188 | setTumo( arrays , 5 , 0 )
189 |
190 | arrays[0] = { 0 , 0 , 0 , 0 }
191 | arrays[1] = { 0 , 4 , 1 , 0 }
192 | arrays[2] = { 0 , 3 , 2 , 0 }
193 | arrays[3] = { 0 , 0 , 0 , 0 }
194 |
195 | setTumo( arrays , 5 , 1 )
196 |
197 | arrays[0] = { 0 , 0 , 0 , 0 }
198 | arrays[1] = { 0 , 3 , 4 , 0 }
199 | arrays[2] = { 0 , 2 , 1 , 0 }
200 | arrays[3] = { 0 , 0 , 0 , 0 }
201 |
202 | setTumo( arrays , 5 , 2 )
203 |
204 | arrays[0] = { 0 , 0 , 0 , 0 }
205 | arrays[1] = { 0 , 2 , 3 , 0 }
206 | arrays[2] = { 0 , 1 , 4 , 0 }
207 | arrays[3] = { 0 , 0 , 0 , 0 }
208 |
209 | setTumo( arrays , 5 , 3 )
210 |
211 | --I=======================================
212 | arrays[0] = { 0 , 0 , 0 , 0 }
213 | arrays[1] = { 1 , 2 , 3 , 4 }
214 | arrays[2] = { 0 , 0 , 0 , 0 }
215 | arrays[3] = { 0 , 0 , 0 , 0 }
```

```
216
217 setTumo( arrays , 6 , 0 )
218
219 arrays[0] = { 0 , 0 , 1 , 0 }
220 arrays[1] = { 0 , 0 , 2 , 0 }
221 arrays[2] = { 0 , 0 , 3 , 0 }
222 arrays[3] = { 0 , 0 , 4 , 0 }
223
224 setTumo( arrays , 6 , 1 )
225
226 arrays[0] = { 0 , 0 , 0 , 0 }
227 arrays[1] = { 0 , 0 , 0 , 0 }
228 arrays[2] = { 4 , 3 , 2 , 1 }
229 arrays[3] = { 0 , 0 , 0 , 0 }
230
231 setTumo( arrays , 6 , 2 )
232
233 arrays[0] = { 0 , 4 , 0 , 0 }
234 arrays[1] = { 0 , 3 , 0 , 0 }
235 arrays[2] = { 0 , 2 , 0 , 0 }
236 arrays[3] = { 0 , 1 , 0 , 0 }
237
238 setTumo( arrays , 6 , 3 )
239
240
241 function update()
242
243    downnum = 0 ;
244
245    y = getFieldHeight()
246    for p = 0 , getFieldHeight() do
247
248       y = y - 1
249
250       checknum = 0
251
252       --横列にブロックが何個いるか捜査
253       for x = 0 , getFieldWidth() do
254
255          if getFieldBlock( x , y ) > 0 and getFieldBlock( x , y ) < 100 then
256             checknum = checknum + 1
257
258          end
259
260       end
261
262       --横列にブロックが 10個以上いた場合消去処理をする
263       if checknum >= 10 then
264
265          checknum = 10
266
267          downnum = downnum + 1
268
269          for x = 0 , getFieldWidth() do
270             setBlockDeleteFlag( x , y )
271          end
272
273       else
```

```
274
275         --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
276         for x = 0 , getFieldWidth() do
277
278           setBlockDownCellNum( x , y , downnum )
279
280         end
281
282       end
283
284
285       --4つ以上くっついていたら消える
286       for x = 0 , getFieldWidth() do
287
288         --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
289         if getComboBlockNum( x , y ) >= 4 and getFieldBlock( x , y ) < 100 and
              getAllBlockDownFlag() == false then
290
291           setBlockDeleteFlag( x , y ) --ブロックを消滅させる
292           --消えたブロックよりも上のブロックに落下をさせる
293           for ys = y , 0 , -1 do
294             if getFieldBlock( x , ys ) > 0 and getFieldBlock( x , ys ) < 100 then
295               setBlockDownCellNum( x , ys , getFieldHeight() )
296             end
297           end
298
299         end
300
301       end
302
303     end
304
305 end
```