卒業研究報告書

題目

三次元チェスのアプリケーション開発およびUIの操作性の向上

指導教員

石水 隆 講師

報告者

10-1-037-0029

土井 悠人

近畿大学理工学部情報学科

平成27年1月31日提出

概要

現在の一般家庭で用いられるパーソナルコンピュータの性能は日々進化し続けている。特にCPUの処理能力の向上は際立っている。こうした優れたCPUが普及したことにより、多くのアプリケーションで3Dグラフィックが用いられるようになった。多くのアプリケーションは、画面上のオブジェクトをマウスで選択することで動作する。しかし、3Dグラフィックで表示されるオブジェクトは、2次元の画面上では重なって表示がされる場合があり選択しにくい。そこで本研究ではユーザにとって使いやすい3Dグラフィックアプリケーションとしての機能を向上させる為には、どのように開発すればよいのかを研究する。

具体的な内容として3Dグラフィックを用いて3Dチェスのアプリケーションを開発する。今回は1907年、ドイツのフェルディナント・マック(Ferdinand Maack)によって開発されたラオムシャッハ(Raumschach)と呼ばれる変則チェスゲームの対戦アプリケーションを作成することにした。ラオムシャッハは3次元方向へ拡張されたチェスであり5×5×5の計125マスからなる立方体状のチェス盤を利用したもので、通常のグラフィック表示では駒の位置関係や駒の移動などが理解しにくく、現実のチェスボードで行うには駒の位置関係の維持等の問題から困難である。

目次

1　序論

1.1 背景

本研究では 3D グラフィックを用いたアプリケーションとして、3D チェスのアプリケーションを開発する。3D チェスとは本来のチェスが平面上の 2 次元的に行われるのに対して、上下方向の 3 次元的に移動が行われる変則的なチェスの一種である。今回はその中でも最も古いものの一つであるラオムシャッハ(Raumschach)[1]と呼ばれる、1907 年にドイツのフェルディナント・マック(Ferdinand Maack)によって開発された三次元チェスのアプリケーション開発を行う。

既存の三次元チェスには 3D CHESS[2]がある。また、既存のラオムシャッハ AI には Jcfrog, Three-dimensional Chess: Raumschach(2014)[3]、L. Lynn Smith, Game: Emperor Raumschach(2005)[4]、Robert Price, Game: Raumschach(2000)[5]等がある。L. Lynn Smith, Game: Emperor Raumschach、Robert Price, Game: Raumschach についてはかなり古い AI であるため、描画性能においては平面上で描画されているチェスを 3 次元的に見せようとしているなど、かなり劣っていると評価した。Three-dimensional Chess: Raumschach については一見優れているラオムシャッハアプリケーションに感じるが、駒の操作から視点の変更までを全てマウスで操作する為、操作性に優れていないアプリケーションであると評価した。

1.2 ゲーム AI の手法

チェスのように可能な局面数が多いゲームに対して完全解析を行うことは困難である。ゲームに対しては確実な最適手を得ることはできないが、局面の評価値計算、定跡データベース。一定手数の先読み、終盤での必勝読みと完全読み、モンテカルロ法[6]などを用いてより有利だと思われる手を選択することができる。

- 局面の評価値計算

局面の評価値計算とは数手先までゲーム木を探索し、その結果を元に評価関数で点数化する手法である。点数化した結果から最善手だと思われる手を判断する。AI の強さは評価関数の作り方に応じて決まるため、評価関数はできるだけ戦局を適切に評価できるように工夫して作る必要がある。

- 定跡データベース

定跡データベースとは過去の多くの試行から今の局面と同じような局面を見つけ出す手法である。その見つけ出した局面の出現頻度や勝率などから最善手を選ぶ事である。しかし、相手があえて定跡以外の手を指すなどして、データベースに無い局面が出てきた時にはこの手法は使えない。

- 一定手数の先読み

先読みとは、可能な範囲で一定数の先の手を読み、その手から作られる局面の評価値を求め、最も評価値が高い手を採用することである。ゲーム木を深くまで探索できると、評価値の精度も上昇する。一方、先読みの手数の増加に伴い探索時間が指数的に増えるため、適度に枝刈りをして探索範囲を減らす工夫をする必要がある。

- 終盤での必勝読みと完全読み

ゲーム終盤になると、そこから勝負がつくまでの手数が少なくなり、また挿せる手が限定されるため勝負が付くまで読み切ることが可能である。終盤での読みは、必勝読みと完全読みがある。必勝読みとはゲーム終盤で勝敗のみを読み切り、必ず勝てる手を指すことを言う。完全読みとは、そこから得られる全ての局面を読み、最も点数の高くなる手を指すことを言う。

必勝読みのほうが計算時間が少なくてすむため、一般にまず必勝読みで勝ちを確定させた上で、残り手数が少なくなると完全読みに切り替えてより点数の高い勝ちを指すことが多い。

- モンテカルロ法

モンテカルロ法とは、シミュレーションや数値計算に乱数を用いて行う手法の総称である。乱数を用いたシミュレーションを何度も行うことにより近似解を求める計算手法。解析的に解くことが出来ない問題でも、十分多くの回収シミュレーションを繰り返すことにより、近似的に解を求めることのできる手法である。

以上の手法を用いることにより、完全解析を行わなくてもある程度の強さのプログラムを作ることは可能であり、ゲームによってはプロに勝つこともできる。

1.3 本報告書の目的

本報告書の 1 つ目の目的として本研究で作成したアプリケーションの利点を周知してもらい、より多くの人間に本アプリケーションを使ってもらう事が挙げられる。また 2 つ目の目的として、仕様や問題点などの自分自身の理解をより深め、整理するために本報告書を製作した。

1.4 本報告書の構成

本報告書では 2 章でラオムシャッハの基本的なルールについて述べる。また、3 章では本アプリケーションを作成した際の仕様を述べる。Unity3D の Inspector の数値の説明、そして各オブジェクトのスクリプトの流れをそれぞれ付録にて説明する。

2 ラオムシャッハ(Raumschach)

本章では本研究で作成するアプリケーションのラオムシャッハ(Raumschach)について述べる。

チェス(Chess)とは世界中で最も広くプレイされているボードゲームである。その中でも変則的なルールに含まれるのがラオムシャッハであり、通常のチェスを三次元に拡張したものである。

ラオムシャッハとは通常のチェスと同じく、二人のプレイヤーがお互いのキングを取り合うゲームである。チェス盤は三次元上に配置され、各駒が移動できる範囲も三次元に拡張されている。

2.1 基本的なルール

ラオムシャッハのチェス盤は縦、横、高さ方向に 5×5×5 に分けられた計 125 マスで構成される。各階層(Column)は下層より順に大文字の A~E で表し、通常のチェス同様(File)は小文字の a～e、行(Rank)は数字の 1～5 で表す。

各マスは Bb5 のように階層、列、行を順に並べて表記される。駒は通常のチェスのキング(King),クイーン(Queen),ビショップ(Bishop),ナイト(Knight),ルーク(Rook),ポーン(Pawn)に加え、ラオムシャッハ独自の駒であるユニコーン(Unicorn)を含めた計 7 種類が用いられる。表 1 にラオムシャッハの駒を示す。白駒は階層 A,B の 1,2 行に、黒駒は階層 D,E の 4,5 列に配置される。憑依地にラオムシャッハで用いられる駒を示す。また、図 1 にラオムシャッハのチェス盤、またゲーム開始時の駒の初期位置を示す。図 1 中の白抜き文字は白駒の初期位置、黒文字は黒駒の初期位置を示している。

ゲームは白と黒に別れ二人のプレイヤーによって行われる。先手は白の駒を一回動かすと次に後手が黒の駒を一回動かす。パスすることは出来ない。自分の手番時に自分の駒の動ける範囲に敵の駒が存在するのなら、その駒をチェスボードから取り除いて自分の駒を取り除いた敵駒の位置に移動させることが出来る。しかしポーンはこれに該当しない。この移動のことを以下"攻撃"と記述する。また敵駒のキングを攻撃できる状態を“チェック(Check)”と呼ぶ。いかなる場合においてもキングはチェックされる位置に移動することは出来ない。キングの動ける位置が存在しないように追い詰めたチェックの事を“チェックメイト(Checkmate)”と呼ぶ。双方のプレイヤーは相手のキングをチェックメイトすることを目指す。

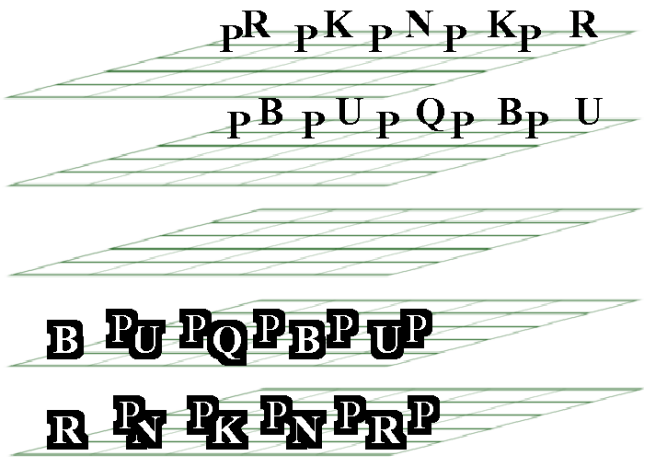

図 1 ラオムシャッハのチェス盤と駒の初期配置

2.2 各駒の動き方

ラオムシャッハの駒の動き方は、通常のチェスの動きを三次元的に拡張したものである。以下では(x, y, z)(a≦x≦e,1≦y≦5,A≦z≦E)をラオムシャッハのチェス盤の(行,列,階層)座標とする。

- キング

キングは全ての方向に 1 マス移動できる。つまりキングが(x, y, z)にいる時、$(x \pm 1, y \pm 1, z \pm 1), (x, y \pm 1, z \pm 1), (x \pm 1, y, z \pm 1), (x, y, z \pm 1), (x, y \pm 1, z), (x \pm 1, y, z)$のいずれか一箇所に移動できる。

ビショップ、ルーク、ナイトは xy 平面、yz 平面、zx 平面の全てに対し従来のチェスト同様の動きをする。

- ビショップ

ビショップを立体部の中心に置いた場合、各辺の中央に向かう 12 方向へ移動できる。すなわち、ビショップは初期位置からベクトル$(0, \pm 1, \pm 1), (\pm 1, 0, \pm 1), (\pm 1, \pm 1, 0)$のいずれか 1 つの方向へ他の駒、または盤端に当たるまで移動できる。

- ルーク

ルークを立方体の中心部に置いた場合、各面の中央に向かう 6 方向に移動できる。すなわち、ルークは初期位置からベクトル$(0, 0, \pm 1), (0, \pm 1, 0), (\pm 1, 0, 0)$のいずれか 1 つの方向へ他の駒、または盤端に当たるまで移動できる。

● ナイト

ナイトは各面に向かって 2 マス進み、そこから 90° 曲がった方向(上下左右 4 方向)に 1 マスだけ進んだ系 24 箇所に移動できる。つまり初期位置(x, y, z)にいるナイトは($x \pm 2, y \pm 1, z$), ($x \pm 2, y, z \pm 1$), ($x \pm 1, y, z \pm 2$), ($x, y \pm 2, z \pm 1$)のいずれか 1 つのマスへ移動できる。

● ユニコーン

ラオムシャッハ独自のユニコーンは立方体の頂点方向 8 方向に無限大に動くことができる。すなわち、ユニコーンは初期位置からベクトル(±1, ± 1, ±1)のいずれか 1 つの方向へ、他の駒、または盤端に当たるまで移動できる。

● クイーン

クイーンはビショップ、ルーク、ユニコーンの全ての動きを兼ね揃えた駒である。すなわち、クイーンは 27 個のベクトル(x, y, z) ($x, y, z = \{-1, 0, 1\}$)のうち 0 ベクトル(0, 0, 0)を除く 26 個のいずれか一つの方向へ他の駒、または盤端に当たるまで移動できる。

図 2、図 3、図 4 に各駒の動き方を示す。図中の色付きマスは各駒が移動できるマスであり、そこに敵駒がいればその駒を攻撃することができる。なお、ナイト以外の駒は他の駒を飛び越えることが出来ない。

● ポーン

本来のポーンの動きとは 1 マス前へ進むか、もしくは相手陣地に進む方向へ階層を 1 層移動できる。初期位置(x, y, z)にいる白駒のポーンは($x, y+1, z$), ($x, y, z+1$)のどちらかに移動でき、($x \pm 1, y \pm 1, z$), ($x \pm 1, y, z+1$), ($x, y+1, z+1$)のいずれかに黒駒がいればそれを取って移動できる。(黒駒を動かす場合はこれの逆の動きをする。)

しかし、今回作成したポーンにはこの動きは搭載されていない。

今回作成したポーンは($x, y+1, z$), ($x, y, z+1$)のどちらかに移動でき($x, y+1, z$), ($x, y, z+1$)に敵駒があれば取って移動することができる。(味方駒のあるマスには移動できない。)

また、通常のチェスでは、初期位置から動いていないポーンは 2 マス前進する 2 段跳ね(2-step initial move)を行えるが、ラオムシャッハでは 2 段跳ねは行えない。

図 5 にポーンの白駒、黒駒それぞれの動きを示す。これも図 2～図 4 と同様に、図中の色付きマスは駒が移動できるマスであり、そこに敵駒がいればその駒を攻撃することができる。

2.3 その他のルール

その他ルールとして、ラオムシャッハにはステールメートおよびプロモーションがある。

● ステールメート

先に記述したとおりキングはチェックされる位置に移動することはできない。よって残りの駒が極端に少なくなった場合、どの駒も動かすことの出来ない状態が発生することがある。

これをステールメイト(Stalemate)といい、この時ゲームは引き分けとなる。本研究で作成したアプリケーションには搭載されていない。

● プロモーション

ポーンがそれ以上進めない位置(相手陣地の最奥)に到達したとき、そのポーンをキング以外の月の駒に変化させることができる。これをプロモーション(昇格)(Promotion)という。こちらも本研究で作成したアプリケーションには搭載されていない。

また通常のチェスに存在するアンパッサン(En passant)(ポーンの特殊な動き)、キャスリング(Castling)(キングとルークの特殊な動き)は、ラオムシャッハには存在しない。

表 1 ラオムシャッハの駒

略称	駒	個数
K	キング(King)	1
Q	クイーン(Queen)	1
N	ナイト(Knight)	2
B	ビショップ(Bishop)	2
R	ルーク(Rook)	2
U	ユニコーン(Unicorn)	2
P	ポーン(Pawn)	10

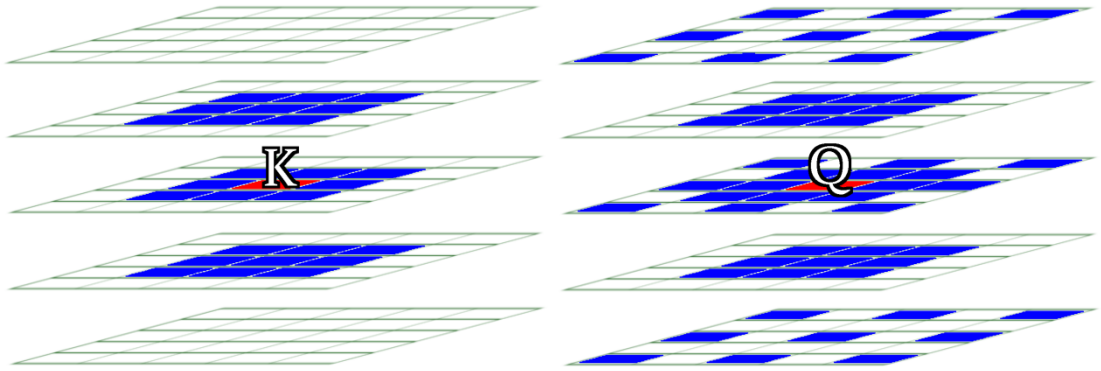

図 2 キング・クイーンの動き方

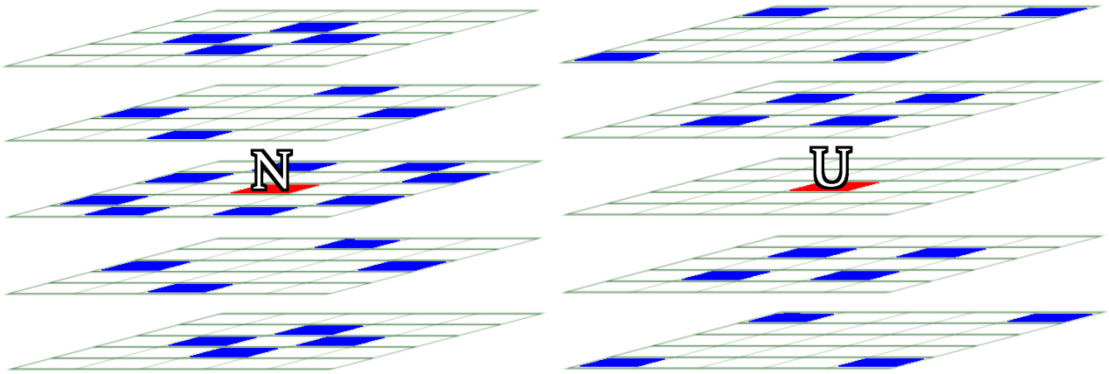

図 3 ナイト・ユニコーンの動き方

他にラオムシャッハでは、兵力不足(どちらのプレイヤーも残りの駒がキングのみになった場合など決着をつけることが不可能になった場合)、千日手(同じ局面を繰り返す場合)、50 手ルール(50 手に渡って一度もポーンが動かず取られもしなかった場合)なども、通常のチェスと同様で引き分けであるが、本研究で開発したアプリケーションにはそれらの機能は搭載されていない。

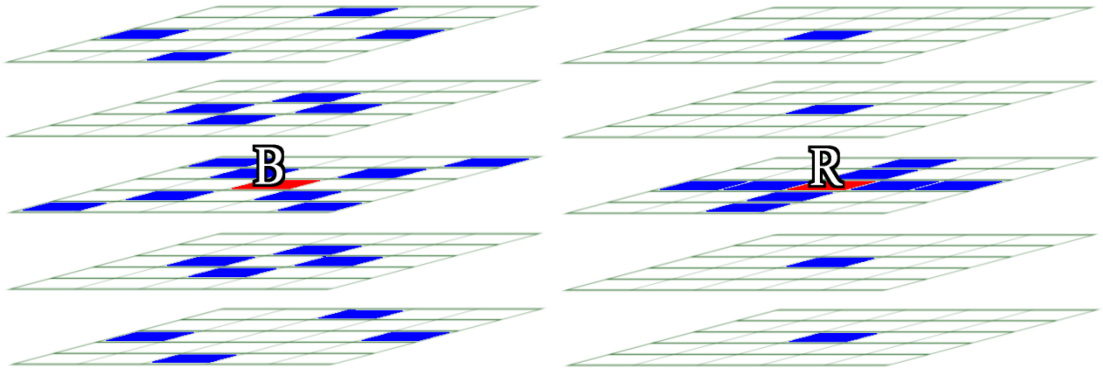

図 4 ビショップ・ルークの動き方

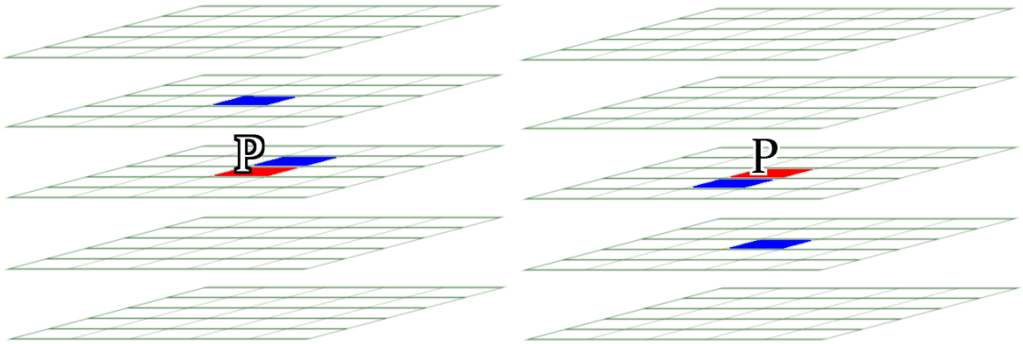

図 5 白駒のポーン・黒駒のポーンの動き方

また、キングの自殺手についても禁止であるが本研究で開発したアプリケーションには搭載されていない。

3 アプリケーションの仕様

本章では、本研究で作成したラオムシャッハアプリケーションについて述べる。Windows 上で実行できるアプリケーションには、Windows アプリケーションとコンソールアプリケーションの 2 種類がある。

Windows アプリケーションとは GUI(グラフィカルユーザインタフェース)で動作するウィンドウアプリケーションのことで、操作や実行結果などをグラフィカルに表現できるアプリケーションである。もう一方のコンソールアプリケーションとは CUI(コマンドラインインタフェース)で動作するアプリケーションのことで、実行結果は文字列での出力となる。

本研究ではアプリケーションを用いるにあたり快適な 3D ゲーム開発エンジンとして Unity3D[7]を利用して開発する。本研究では Unity3D の Game 画面内での動作として実行される。

3.1 Unity3D について

本研究で使用した Unity3D は高度なプログラミング技術力が無くても手軽に開発することができる事で近年注目されている総合開発エンジンである。3D ゲームの開発に適しているが、ソーシャルゲームなどの実績もある。また 3D ゲームに限らず 2D ゲームを作成するための機能も強化されている。(バージョン 4.3 以降)

本研究では三次元チェスのアプリケーションを開発するので、3D ゲームの開発エンジンとして以前より噂されていた Unity3D に注目し開発を行った。

3.2 ゲームオブジェクトについて

Unity3D の大きな特徴として、ゲームオブジェクトのスクリプトに直接プログラミングを行い、すぐに反映させ視覚的に動作テストが行える。ゲームオブジェクトの作成、また物理エンジンの追加などは Unity3D 内にて直接追加が行える。Inspector と呼ばれるオブジェクト情報の集合を変更することにより駒のゲームを始める前の初期情報などを書き換えることができる。チェスの駒の初期配置、またチェスの盤などについても同様に作成した。本研究で作成したゲームオブジェクトを表 2 に示す。また付録に各オブジェクトの inspector の中身を記述する。

表 2　作成したゲームオブジェクトの名前とスクリプト名

オブジェクト名	スクリプト名
Kingw	King
Queenw	Queen
Knightw	Knight
Unicornw	Unicorn
Bishopw	Bishop
Rookw	Rook
Pawnw	Pawn
GameText	Game
Plane	無し

以下、各オブジェクトについて説明する。

- Kingw
  Kingw は白のキングを表すオブジェクトである。各面に白で“K”と書かれた立方体で表示される。
- Queenw
  Queenw は白のクイーンを表すオブジェクトである。各面に白で“Q”と書かれた立方体で表示される。

- Knightw
  Knightw は白のナイトを表すオブジェクトである。各面に白で“N”と書かれた立方体で表示される。
- Unicornw
  Unicornw は白のユニコーンを表すオブジェクトである。各面に白で“U”と書かれた立方体で表示される。
- Bishopw
  Bishopw は白のビショップを表すオブジェクトである。各面に白で“B”と書かれた立方体で表示される。
- Rookw
  Rookw は白のルークを表すオブジェクトである。各面に白で“R”と書かれた立方体で表示される。
- Pawnw
  Pawnw は白のポーンを表すオブジェクトである。各面に白で“P”と書かれた立方体で表示される。

列挙したのは白駒だけであるが同様に黒駒もオブジェクト名“Kingb”スクリプト名“Kingb”としてそれぞれ作成した。また複数個ある駒のオブジェクト名に関しては“Pawn1w,Pawn2w,……”のようにそれぞれ別のオブジェクト名が付けられている。スクリプト名に関しては白駒、黒駒の違いしか無く“Pawn1w,Pawn2w,……”といった同じ種類の駒は同一の“Pawn”スクリプトで動作している。

- GameText
  GameText はテキスト表示を行うオブジェクトである。ゲーム開始時には表示されていないがゲーム判定を行い終了条件が満たされていれば画面上に表示される。
- Plane
  Plane はチェス盤のそれぞれのマスを表すオブジェクトである。各オブジェクトは白と黒に色分けされている。

3.3 ゲームの流れ

本研究で作成したゲームの流れを説明する。アプリケーションの実行は Unity3D 内の実行ボタンをクリックすることで起動する。

本アプリケーションは実行された時点では動かず白駒のプレイヤーの入力待ち状態になっている。手番になったプレイヤーは以下の状態を繰り返し、ゲームを進めていく。図 6 にゲーム実行時の初期状態の画面を示す。

1. 移動する駒を選択
   手番になったプレイヤー側の駒は最初に入力待ち状態になっている。
   手番になっている側の駒の選択をクリックにて行う。駒がクリックされたら、その駒の色を黄色に変更し状態 2.へ移行する。
2. 選択した駒を移動する先を選択
   選択した駒の移動先マスをクリックにて選択できる。選択されたマスが移動できるマスの場合移動をして状態 3.へ移行する。また、クリックされたマスに攻撃できる駒があればゲームから取り除き移動をして状態 3.へ移行する。この時動けないマスを選択しても

駒は反応せず、状態 2.から移行しない。また状態 1.の時点で選択された駒を、この時選択すると黄色に変更されていた色は元の色に戻され、また駒の選択状態も解除される。

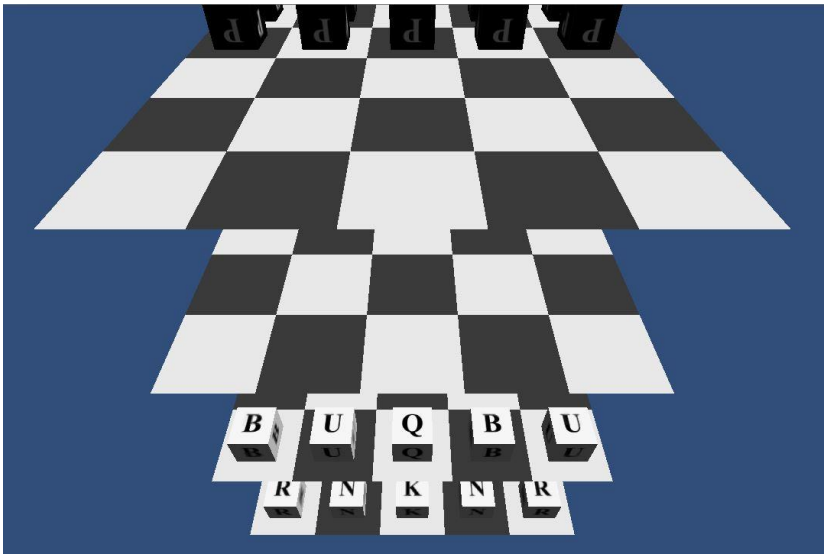

図 6 ゲーム実行時の初期状態

3. 移動した後
   この状態ではプレイヤーは操作を行わない。
   ゲームの終了条件を満たしていればゲーム上に“GAME OVER”の文字とともに“勝利した方のプレイヤーカラー(WHITE or BLACK)　WIN”が表示される。
   ゲームの終了条件を満たしていなければ、選択されていた駒の色が元の色に戻り選択状態も解除される。またプレイヤーが相手に移り状態 1.へ移行する。

また状態 1.～状態 3.に関わらず視点を自由に動かすことができる。W、S、A、D キーがそれぞれ視点の上、下、左、右に、また矢印キーの↑、↓、←、→がそれぞれの方向に視点を回転できるようになっている。図 7 に状態 2.のゲーム実行画面を示す。また駒やチェス盤の配置、プログラムの内容については付録にて解説する。

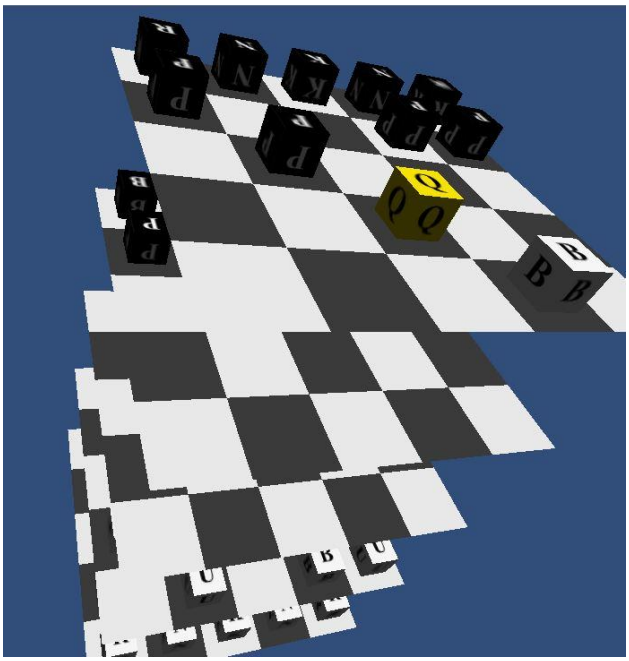

図 7 状態 2.の一例

4 考察

本研究の大きな目的の一つでもある 3D チェスアプリケーションの UI の機能向上については、マウス操作への対応や視点の自由な切り替え機能の搭載など、本来の目的は満足していると考える。また既存の 3D チェスアプリケーションである Three-dimensional Chess: Raumschach[3]等と比較した所、操作性において既存のアプリケーションには駒の動き、また視点の変更も全てマウスのみで行っており、駒を操作しようとして誤ってチェス盤の視点を回転させてしまう事があった。本研究で作成したアプリケーションは、駒はマウス操作、視点変更はキーボード、といったように分離させて操作することで快適な操作性を得た。こうした点から操作性については本研究が既存のアプリケーションよりも優れていると評価した。また視認性において、本研究で作成したアプリケーションでは実際の駒の形を再現したわけではないが、駒の種類の頭文字を立体のオブジェクトに貼り付けることによりシンプルで認識しやすいものとなっている。こうした点から視認性については既存のアプリケーションと同等であると評価した。

しかしながら今回作成したアプリケーションは UI の機能向上といった点に拘り過ぎてしまい本来の 3D チェスで搭載せねばならなかったポーンの移動後の攻撃、またプロモーションなどの一部のルールが実装されていない。まだこれでは完璧な 3D チェスアプリケーションとは呼べず更なる改善が必要である。このような点が反省課題に挙げられる。

5 結論・今後の課題

本研究では、3D グラフィックを用いたラオムシャッハアプリケーションを開発した。本アプリケーションでは 2 人のプレイヤーがラオムシャッハで対戦が行える。ままた既存の 3D チェスアプリケーションである Three-dimensional Chess: Raumschach[3]等と比較した所、操作性において既存のアプリケーションには駒の動き、また視点の変更も全てマウスのみで行っており、駒を操作しようとして誤ってチェス盤の視点を回転させてしまう事があった。本研究で作成した

アプリケーションは、駒はマウス操作、視点変更はキーボード、といったように分離させて操作することで快適な操作性を得た。こうした点から操作性については本研究が既存のアプリケーションよりも優れていると評価した。また視認性において、本研究で作成したアプリケーションでは実際の駒の形を再現したわけではないが、駒の種類の頭文字を立体のオブジェクトに貼り付けることによりシンプルで認識しやすいものとなっている。こうした点から視認性については既存のアプリケーションと同等であると評価した。しかしながら本研究で作成したアプリケーションでは本来の 3D チェスで搭載せねばならなかった一部のルールが搭載されておらず更なる改善が必要である。そして本研究で作成したアプリケーションは Unity3D 内での操作に留まっている。このままではゲームの内容を改善してもゲームで遊べるユーザが限られてしまう。今後の課題として、より多くの人に使ってもらえるアプリケーションにするためには、ゲーム開発エンジン内だけでなく実際に Windows アプリケーションとして操作できるように変更することが考えられる。また対人戦だけでなく CPU との対戦できるように変更することも考えられる。例を挙げると既存の将棋やチェスの AI には 1.2 節で述べた評価値計算や探索が利用されている。[8] 通常のチェスでなく 3 次元のチェスであるためプログラムを流用することはできないが、ゲーム木探索を深く進めることによって人間とまともに戦える CPU が作成できるのではないかと考える。

謝辞

本研究の完成、また本報告書の制作にあたり、数々の御指導、御支援などを頂き石水先生には大変お世話になりました。誠に感謝申し上げます。ありがとうございました。

参考文献

[1] A.S.M.Dickins, “Guide to Fairy Chess,” Dover Publications Inc (1971)
[2] Dan Beyer, 3D CHESS (2006), http://thehinge.net/3dchess
[3] Jcfrog, Three-dimensional Chess: Raumschach (2014),
https://fr.jocly.com/raumschach
[4] L. Lynn Smith, Game: Emperor Raumschach (2005),
http://www.zillions-of-games.com/cgi-bin/zilligames/submissions.cgi?do=show;id=1116
[5] Robert Price, Game: Raumschach (2000),
http://www.zillions-of-games.com/cgi-bin/zilligames/submissions.cgi?do=show;id=708
[6] 美添一樹、山下宏、松原仁、コンピュータ囲碁-モンテカルロ法の理論と実践-, 共立出版 (2012)
[7] Unity3D, http://japan.unity3d.com/
[8] 竹内聖悟, コンピュータ将棋の技術とGPS将棋について, 第19回ビジュアリゼーションカンファレンス, 一般社団法人 可視化情報学会 (2013).
http://www.cybernet.co.jp/avs/documents/pdf/seminar_event/conf/19/1-3.pdf

付録 A 各ゲームオブジェクトの inspector

- Kingw

Tag = piece1 //相手の駒と味方の駒を識別するためのタグ(piece1=白駒)
Layer = Default
Transform
 Position //初期位置
  X = 0, Y = -3.77, Z = 2
 Scale //オブジェクトの大きさ
  X = 0.5, Y = 0.5, Z = 0.5
Script = King //読み込んだスクリプト
RigidBody
 Mass = 1 //質量
 Drag = 0 //空気抵抗
 Angular Drag = 0.05 //回転に対する抵抗
 Use Gravity //重力を使用
Constraints
 Freeze Position X, Z //X,Z 軸方向に動かない
 Freeze Rotation X, Y, Z //X,Y,Z 軸方向に回転しない

以下各白駒については相違点のみ記述する

- Queenw

Tag = piece1
Layer = Default
Transform
 Position
  X = 0, Y = -1.77, Z = 2
 Scale
  X = 0.5, Y = 0.5, Z = 0.5
Script = Queen

- Knight1w, Knight2w

Tag = piece1
Layer = Default
Transform
 Position
  X = 1, -1, Y = -3.77, Z = 2
 Scale
  X = 0.5, Y = 0.5, Z = 0.5
Script = Knight

- Unicorn1w, Unicorn2w

Tag = piece1
Layer = Default
Transform
 Position
  X = 1, -2, Y = -1.77, Z = 2
 Scale
  X = 0.5, Y = 0.5, Z = 0.5
Script = Unicorn

- Knight1w, Knight2w

Tag = piece1
Layer = Default
Transform
 Position
  X = 1, -1, Y = -3.77, Z = 2
 Scale
  X = 0.5, Y = 0.5, Z = 0.5
Script = Knight

- Bishop1w, Bishop2w

Tag = piece1
Layer = Default
Transform
 Position
  X = 2, -1, Y = -1.77, Z = 2
 Scale
  X = 0.5, Y = 0.5, Z = 0.5
Script = Bishop

- Rook1w, Rook2w

Tag = piece1
Layer = Default
Transform
 Position
  X = 2, -2, Y = -3.77, Z = 2
 Scale
  X = 0.5, Y = 0.5, Z = 0.5
Script = Rook

- Pawn1w ～ Pawn5w

Tag = piece1
Layer = Default
Transform
 Position
  X = 2, 1, 0, -1, -2, Y = -3.77, Z = 1
 Scale
  X = 0.5, Y = 0.5, Z = 0.5
Script = Pawn

- Pawn6w ～ Pawn10w

Tag = piece1
Layer = Default
Transform
 Position
  X = 2, 1, 0, -1, -2, Y = -1.77, Z = 1
 Scale
  X = 0.5, Y = 0.5, Z = 0.5
Script = Pawn

- Kingb

Tag = piece2//相手の駒と味方の駒を識別するためのタグ(piece2=黒駒)
Layer = Default
Transform
 Position
  X = 0, Y = 4.23, Z = -2
 Scale
  X = 0.5, Y = 0.5, Z = 0.5
Script = Kingb
RigidBody
 Mass = 1
 Drag = 0
 Angular Drag = 0.05
 Use Gravity
Constraints
 Freeze Position X, Z
 Freeze Rotation X, Y, Z

以下各黒駒については相違点のみ記述する

- Queenb

Tag = piece2
Layer = Default
Transform
 Position
  X = 0, Y = 2.23, Z = -2
 Scale
  X = 0.5, Y = 0.5, Z = 0.5
Script = Queenb

- Knight1b, Knight2b

Tag = piece2
Layer = Default
Transform
 Position
  X = -1, 1, Y = 4.23, Z = -2
 Scale
  X = 0.5, Y = 0.5, Z = 0.5
Script = Knightb

- Unicorn1b, Unicorn2b

Tag = piece2
Layer = Default
Transform
 Position
  X = 1, -2, Y = 2.23, Z = -2
 Scale
  X = 0.5, Y = 0.5, Z = 0.5
Script = Unicornb

- Bishop1b, Bishop2b

Tag = piece2
Layer = Default
Transform
 Position
  X = 2, -1, Y = 2.23, Z = -2
 Scale
  X = 0.5, Y = 0.5, Z = 0.5
Script = Knightb

- Rook1b, Rook2b

Tag = piece2
Layer = Default
Transform
 Position
  X = -2, 2, 1, Y = 4.23, Z = -2
 Scale
  X = 0.5, Y = 0.5, Z = 0.5
Script = Rookb

- Pawn1b ～ Pawn5b

Tag = piece2
Layer = Default
Transform
 Position
  X = -2, -1, 0, 1, 2, Y = 4.23, Z = -1
 Scale
  X = 0.5, Y = 0.5, Z = 0.5
Script = Pawnb

- Pawn6b ～ Pawn10b

Tag = piece2
Layer = Default
Transform
 Position
  X = -2, -1, 0, 1, 2, Y = 2.23, Z = -1
 Scale
  X = 0.5, Y = 0.5, Z = 0.5
Script = Pawnb

付録 B　各ゲームオブジェクトのスクリプト

- King

```
public class King : MonoBehaviour{
 RaycastHit hit, hit2, hit9, hit10; //オブジェクトの存在の判定を行う。
 RaycastHit cellhit; //オブジェクトの存在の判定を行う。
 bool isSelect = false; //駒の選択状態判定用の変数である。
 Color orgColor,selColor = Color.yellow; //駒が選択された時の色を保管しておく変数である。
 //string t = "a"; //テスト用のテキストである

 void Start(){ //初期条件
  orgColor = gameObject.renderer.material.color; //元々のオブジェクトの色に設定しておく。
 }

void Update(){ //常に実行され続けている。
  GameJudge (); //ゲーム終了条件を満たしているかの判定メソッド
  /*  プレイヤーのターンかどうかの判定をしている。
  *Global.flg とはグローバル変数(bool 型)であり
  * 'Global.flg == true'で白のターンである。
  */
  if (!Global.flg ) { //敵のターンだった場合
   return; //Update()を終了させている。
  }
  if(Input.GetMouseButtonDown(0)){ //マウスの左クリックが押された時
   Vector3 pos = Input.mousePosition; //押された時のマウスの位置を保管する。
   pos.z = 10.0f; //しかしマウスでは奥行きが認識できないため z 座標を設定する。
/*カメラワークの位置から pos 座標へ向けて
*直線に伸びる光線を作成する。
*/
   Ray ray = Camera.main.ScreenPointToRay(pos);

/*直線に 100 の距離の
*光線を飛ばし当たったゲームオブジェクトがあるか。
*あるなら hit に代入
*/
   if(Physics.Raycast(ray,out hit,100)){

/*hit に代入されたゲームオブジェクトの名前が
*/このスクリプトのオブジェクトであるか
     if(hit.collider.gameObject.name == gameObject.name){
      isSelect = !isSelect;//f -> t //このオブジェクトに当たったなら選択状態へ移行
     }
// クリックした位置がチェス盤であった場合
```

```
      if(isSelect && hit.collider.gameObject.tag == "cell"){
       if(Control()){ //クリックされたマスが移動可能マスか判定
        Vector3 newpos = hit.collider.gameObject.transform.position; //クリック位置を保管
        /*クリック位置を保管して y 座標だけ 1f 大きい値に設定する。
        *
        */
        Vector3 newpos2 = new Vector3(newpos.x ,newpos.y + 1.0f ,newpos.z);
/*クリックされた位置の 1f 高い位置から真下に 3f の長さの光線を放ち当たったオブジェクトを
*hit2 に保管する。
*/
       if(Physics.Raycast(newpos2,Vector3.down ,out hit2,3f)){
        if(hit2.collider.gameObject.tag == "piece2"){//光線の当たったオブジェクトが敵駒である場合
         Destroy(hit2.collider.gameObject);//当たった駒の名前を取得しゲームから取り除く。
        }
       }
       //クリックしたマスへ移動(y 座標の+0.23f はマスの厚みを配慮)
        transform.position = new Vector3(newpos.x, newpos.y + 0.23f, newpos.z);
        //collision Time
        isSelect = false; //選択状態の解除
        Flg fchange = new Flg(); //Flg クラスのインスタンスの作成
        /*グローバル変数である Global.flg の T or F の入れ替え
        *これによって白駒が動かせなくなり黒駒が動かせるようになる。
        */
        Global.flg = fchange.FlgSetter(Global.flg);
       }
      }

      if(isSelect){ //選択状態の時
       gameObject.renderer.material.color = selColor; //駒の色を変える。(黄色)
      }else{ //選択状態でない場合
       gameObject.renderer.material.color = orgColor; //駒の色を元の色に戻す。
      }
     }
    }
   }

//King の動けるマスの判定を行う
  public bool Control(){//King piece control
    float xPosition = transform.position.x ;
    float yPosition = transform.position.y - 0.23f;
    float zPosition = transform.position.z ;

    bool a = false;
    for(float X = -1f; X <= 1f; X++ ){
     for(float Y = -2f; Y <= 2f; Y = Y + 2f){
```

```
      for(float Z = -1f; Z <= 1f; Z++ ){
       //King の現在の座標からの絶対位置を表す
       Vector3 onCell = new Vector3(xPosition + X, yPosition + Y, zPosition + Z);//true position
       Vector3 hitposition = hit.collider.gameObject.transform.position;//King の移動できる座標の保管

       if(hit.collider.gameObject.transform.position.x >= X -0.5f&&
hit.collider.gameObject.transform.position.x < X + 0.5f&&
hit.collider.gameObject.transform.position.y >= Y -0.5f &&
hit.collider.gameObject.transform.position.y < Y + 0.5f&& hit.collider.gameObject.transform.position.z
>= Z -0.5f && hit.collider.gameObject.transform.position.z < Z + 0.5f){//Ray hit position
        hitposition = new Vector3(X ,Y ,Z); //光線がマスに当たった座標は整数値でないため整数値に変換
       }

//光線が当たった座標が King の動ける位置に収まっているか
       if(onCell.x == hitposition.x && onCell.z == hitposition.z && onCell.y == hitposition.y){

        a = true; //bool の書き換え
        /**annallyJudge**/
        Vector3 anally0 = hit.collider.gameObject.transform.position; //クリックした座標を代入
        //クリックした座標の y 座標が+1f の位置を代入
        Vector3 anally = new Vector3 (anally0.x, anally0.y + 1.0f, anally0.z);
        /*クリックした座標の y 座標が+1f の位置から下方向に 3f の長さの光線を発射する。
        *ゲームオブジェクトにぶつかったら hit10 に情報が記憶される。
        */
        if(Physics.Raycast(anally,Vector3.down ,out hit10,3f)){

         if(hit10.collider.gameObject.tag == "piece1"){//hit10 は味方の駒であるか
          a = false;//味方の駒のあるマスであると移動できないため false に書き換え
         }
        }

        if(X == 0f && Y == 0f && Z == 0f){//今選択されている駒のある位置(移動しない)
         a = false;//移動しないのは false
        }
         break;
       }
      }
     }
    }
   return a;//移動できるマスなら T 不可なら F
  }


//ゲーム終了判定メソッド
  public void GameJudge(){
```

```
  bool whiteKing = false;//白キングがあるかの判定
  bool blackKing = false;//黒キングがあるかの判定
  for (float X = -2f; X <= 2f; X++) {//全てのマスの探索
   for (float Y = -4f; Y <= 4f; Y = Y + 2f) {
    for (float Z = -2f; Z <= 2f; Z++) {
     Vector3 onCell = new Vector3 (X, Y + 1f, Z);//全てのマスの y 座標の 1f 大きい位置を保管
     if (Physics.Raycast (onCell, Vector3.down, out hit, 3f)) {//全てのマスのオブジェクトを調べる
      if (hit.collider.gameObject.name == "Kingw") {//白のキングの有無
       whiteKing = true;//白のキングがあれば true
      } else if (hit.collider.gameObject.name == "Kingb") {//黒のキングの有無
       blackKing = true;//黒のキングがあれば true
      }
     }
    }
   }
  }
  for (float X = -2f; X <= 2f; X++) {//全てのマスの探索
   for (float Y = -4f; Y <= 4f; Y = Y + 2f) {
    for (float Z = -2f; Z <= 2f; Z++) {
     Vector3 onCell = new Vector3 (X, Y + 1f, Z); //全てのマスの y 座標の 1f 大きい位置を保管
     if (Physics.Raycast (onCell, Vector3.down, out hit9, 3f)) {//全てのマスのオブジェクトを調べる
     //白キングのある状態で黒キングの有無の判定
      if (hit.collider.gameObject.name == "Kingb" && whiteKing) {
       blackKing = true;//黒のキングがあれば true
       //黒キングのある状態で白キングの有無の判定
      } else if (hit.collider.gameObject.name == "Kingw" && blackKing) {
       whiteKing = true;//白のキングがあれば true
      }
     }
    }
   }
  }

  if (whiteKing && blackKing) {//白黒のキングがあるか
   } else {//片一方もしくは両方が存在しない場合
   GameOver (whiteKing, blackKing); //ゲーム終了判定
  }
 }

//ゲーム終了判定メソッド
  public void GameOver(bool whiteKing,bool blackKing){
   if(whiteKing && !blackKing){//白キング T 黒キング F
    WhiteWin();//白勝利メソッド呼び出し
   }else if(!whiteKing && blackKing){//白キング F 黒キング T
    BlackWin();//黒勝利メソッド呼び出し
```

```
  }else{//白キング F 黒キング F
    DrawGame();//ドローメソッド呼び出し
  }
 }
//白勝利メソッド
 public void WhiteWin(){
  Game g = new Game();//Game クラスのインスタンス作成
  Global.tex = "GAME OVER¥n" +
  "WHITE WIN!";//ゲーム終了時のテキストをグローバル変数である Global.tex に代入
  gameov();//テキスト表示用のメソッド呼び出し
 }

 public void BlackWin(){//Game クラスのインスタンス作成
  Game g = new Game();
  Global.tex = "GAME OVER¥n" +
  "BLACK WIN!"; //ゲーム終了時のテキストをグローバル変数である Global.tex に代入
  gameov();//テキスト表示用のメソッド呼び出し
 }

 public void DrawGame(){//Game クラスのインスタンス作成
  Game g = new Game();
  Global.tex = "GAME OVER¥n" +
   "DRAW GAME";//ゲーム終了時のテキストをグローバル変数である Global.tex に代入
  gameov();//テキスト表示用のメソッド呼び出し
 }

 public void gameov(){//テキスト表示を行う
  GameObject.Find ("GameText").guiText.text = "" + Global.tex;
 }

 /*
 void OnGUI () {//テスト用のテキスト表示メソッド
  GUI.TextField (new Rect (10, 10, 100, 200), t);
 }
 */
}
```

- Queen

```
public class Queen: MonoBehaviour{
   RaycastHit hit, hit2, hit9, hit10;//オブジェクトの存在判定を行う。
   bool isSelect = false;//駒の選択状態判定用の変数である。
   Color orgColor,selColor = Color.yellow;//駒が選択された時の色を保管しておく変数である。
   //string t = "a";//テスト用のテキストである。
```

```
    void Start(){//初期条件
        orgColor = gameObject.renderer.material.color;//元々のオブジェクトの色に設定しておく
    }

    void Update(){//常に実行され続けている
/*プレイヤーのターンかどうかの判定をしている。
*Glpbal.flg とはグローバル変数(bool 型)であり
*'Global.flg == true'で白のターンである。
*/
        if (!Global.flg) {
            return;//Update()を終了させている。
        }
        if(Input.GetMouseButtonDown(0)){//マウスの左クリックが押された時
            Vector3 pos = Input.mousePosition;//押された時のマウスの位置を保管する。
            pos.z = 10.0f;//しかしマウスは奥行きが認識できないため z 座標を設定する。

            Ray ray = Camera.main.ScreenPointToRay(pos);//光線の作成

            if(Physics.Raycast(ray,out hit,100)){//当たったオブジェクトの判定
                //当たったオブジェクトがこのオブジェクトの場合
                if(hit.collider.gameObject.name == gameObject.name){
                    isSelect = !isSelect;//f -> t//選択状態に移行
                }

                if(isSelect && hit.collider.gameObject.tag == "cell"){//マスがクリックされた時
                    if(Control()){//移動可能か判定
                        Vector3 newpos = hit.collider.gameObject.transform.position;クリック位置保管
                        //敵駒判定用の光線作成
                        Vector3 newpos2 = new Vector3(newpos.x ,newpos.y + 1.0f ,newpos.z);
                        if(Physics.Raycast(newpos2,Vector3.down ,out hit2,3f)){
                            if(hit2.collider.gameObject.tag == "piece2"){
                                Destroy(hit2.collider.gameObject);//敵駒破壊
                            }
                        }
                        //クリックしたマスへ移動
                        transform.position = new Vector3(newpos.x, newpos.y + 0.23f, newpos.z);
                        //collision Time
                        isSelect = false;//選択状態解除
                        Flg fchange = new Flg();//白黒ターン入れ替えのフラグ変更
                        Global.flg = fchange.FlgSetter(Global.flg);
                    }
                }

                if(isSelect){//選択駒の色変え
```

```
                gameObject.renderer.material.color = selColor;
            }else{
                gameObject.renderer.material.color = orgColor;
            }
        }
    }
}

    public bool Control(){//Queenpiece control
        float xPosition = transform.position.x ;//now position
        float yPosition = transform.position.y - 0.23f;
        float zPosition = transform.position.z ;
//今現在のオブジェクトの位置から y 座標を調整して保管
        Vector3 np = new Vector3 (xPosition, yPosition + 0.3f, zPosition);
        bool a = false;//移動できる位置か判定
        for(float X = -4f; X <= 4f; X++ ){
            for(float Y = -8f; Y <= 8f; Y = Y + 2f ){
                for(float Z = -4f; Z <= 4f; Z++ ){
                    Vector3 onCell = new Vector3(xPosition + X, yPosition + Y, zPosition + Z);//true position
                    //クリックした位置を保管
                    Vector3 hitposition = hit.collider.gameObject.transform.position;

                    //x.0000~ -> xf
                    if(hit.collider.gameObject.transform.position.x >= X -0.5f
                        && hit.collider.gameObject.transform.position.x < X + 0.5f
                        && hit.collider.gameObject.transform.position.y >= Y -0.5f
                        && hit.collider.gameObject.transform.position.y < Y + 0.5f
                        && hit.collider.gameObject.transform.position.z >= Z -0.5f
                        && hit.collider.gameObject.transform.position.z < Z + 0.5f){//Ray hit position
                       hitposition = new Vector3(X ,Y ,Z);//整数調整
                    }

                    //移動できる範囲に収まっているかの判定
                    if(onCell.x == hitposition.x && onCell.z == hitposition.z && onCell.y == hitposition.y){
                    //実際に移動できるマスか判定
                       if (QueenJudge (X, Y, Z)){
                          a = true;//移動できるよう書き換え
                          if (X == 0f && Y == 0f && Z == 0f){//移動してない位置をクリックしていた場合
                             a = false;//移動できないよう書き換え
                          }
                          /*今の位置とクリックした位置の間のベクトルの大きさを保管する。
                          *しかし後に間にオブジェクトがあるか判定するためクリックした位置の駒と
                          *区別するため 1f 短くしてクリックマスにある駒に当たらないようにしている。
                          */
                          float dis = (hitposition - np).magnitude - 1.0f;//hitp <--> nowp
```

```
                    //全てのマスオブジェクトの情報を取得
                    GameObject[] cells = GameObject.FindGameObjectsWithTag("cell");


                    //今あるオブジェクトの位置とクリックした地点の距離を保管
                    Vector3 vecp = new Vector3 (hitposition.x - np.x, hitposition.y - np.y + 0.3f,
hitposition.z - np.z);

                    /*hits に現在のオブジェクト位置と移動先(クリック位置)の間の全てのオブジェクトの
                    *情報を格納していく。(格納される順番はランダム)
                    */
                    RaycastHit[] hits = Physics.RaycastAll (np, vecp, dis);
                    for (int i = 0; i < hits.Length; i++) {//何個間にあったか判定
                       //当たったオブジェクトのタグを保管する
                       string hitTag = hits [(int)i].collider.gameObject.tag;
                       //当たったオブジェクトが駒だった場合移動できないよう書き換え
                       if (hitTag == "piece2" || hitTag == "piece1") {
                           a = false;
                       }
                    }
                    break;
                }
            }
        }
    }
    return a;
}


public bool QueenJudge(float X, float Y, float Z){
    bool tf = false;

//クイーンの現在の位置からの相対的な移動可能判定
    if (X == Z && X == Y/2 ||
        -X == Z && -X == Y/2 ||
        X == -Z && X == Y/2 ||
        -X == -Z && -X == Y/2 ||
        X == Z && X == -Y/2 ||
        -X == Z && -X == -Y/2 ||
        X == -Z && X == -Y/2 ||
        -X == -Z && -X == -Y/2) {
       tf = true;
    } else if (Y == 0f && X == Z ||
                Y == 0f && X == -Z ||
```

```
                Y == 0f && -X == Z ||
                Y == 0f && -X == -Z) {
            tf = true;
        }else if(X == 0 && Z == Y / 2||
                 X == 0 && -Z == Y / 2||
                 Z == 0 && X == Y / 2||
                 Z == 0 && -X == Y / 2){
            tf = true;
        }else if (Y == 0f && X == 0f ||
                  Y == 0f && Z == 0f) {
            tf = true;
        }else if(Y != 0f && X == 0f && Z == 0f){
            tf = true;
        }

        /**annallyJudge**/
        Vector3 anally0 = hit.collider.gameObject.transform.position;//クリックした座標を代入
        //クリックした座標の y 座標が+1f の位置を代入
        Vector3 anally = new Vector3 (anally0.x, anally0.y + 1.0f, anally0.z);
        //下方向に光線を発射しぶつかったゲームオブジェクトの情報を記憶。
        if(Physics.Raycast(anally,Vector3.down ,out hit10,3f)){
            //ぶつかったオブジェクトが味方の駒の時
            if(hit10.collider.gameObject.tag == "piece1"){
                tf = false;//飛び越え禁止
            }
        }

        return tf;
    }

    /*
    void OnGUI () {//テスト用テキスト表示メソッド
        GUI.TextField (new Rect (10, 10, 100, 200), t);
    }*/

}
```

- Knight

```
public class Knight: MonoBehaviour{
    RaycastHit hit, hit2, hit9, hit10;//オブジェクトの存在判定を行う。
    bool isSelect = false;//駒の選択状態判定用の変数である。
    Color orgColor,selColor = Color.yellow;//駒が選択された時の色を保管しておく変数である。
    //string t = "a";//テスト用のテキストである。
```

```
    void Start(){//初期条件
        orgColor = gameObject.renderer.material.color;//元々のオブジェクトの色に設定しておく
    }

    void Update(){//常に実行され続けている
/*プレイヤーのターンかどうかの判定をしている。
*Glpbal.flg とはグローバル変数(bool 型)であり
*'Global.flg == true'で白のターンである。
*/
        if (!Global.flg) {
            return;//Update()を終了させている。
        }
        if(Input.GetMouseButtonDown(0)){//マウスの左クリックが押された時
            Vector3 pos = Input.mousePosition;//押された時のマウスの位置を保管する。
            pos.z = 10.0f;//しかしマウスは奥行きが認識できないため z 座標を設定する。

            Ray ray = Camera.main.ScreenPointToRay(pos);//光線の作成

            if(Physics.Raycast(ray,out hit,100)){//当たったオブジェクトの判定
                //当たったオブジェクトがこのオブジェクトの場合
                if(hit.collider.gameObject.name == gameObject.name){
                    isSelect = !isSelect;//f -> t//選択状態に移行
                }

                if(isSelect && hit.collider.gameObject.tag == "cell"){//マスがクリックされた時
                    if(Control()){//移動可能か判定
                        Vector3 newpos = hit.collider.gameObject.transform.position;クリック位置保管
                        //敵駒判定用の光線作成
                        Vector3 newpos2 = new Vector3(newpos.x ,newpos.y + 1.0f ,newpos.z);
                        if(Physics.Raycast(newpos2,Vector3.down ,out hit2,3f)){
                            if(hit2.collider.gameObject.tag == "piece2"){
                                Destroy(hit2.collider.gameObject);//敵駒破壊
                            }
                        }
                        //クリックしたマスへ移動
                        transform.position = new Vector3(newpos.x, newpos.y + 0.23f, newpos.z);
                        //collision Time
                        isSelect = false;//選択状態解除
                        Flg fchange = new Flg();//白黒ターン入れ替えのフラグ変更
                        Global.flg = fchange.FlgSetter(Global.flg);
                    }
                }

                if(isSelect){//選択駒の色変え
```

```
                gameObject.renderer.material.color = selColor;
            }else{
                gameObject.renderer.material.color = orgColor;
            }
        }
    }
}

    public bool Control(){//Knightpiece control
        float xPosition = transform.position.x ;//now position
        float yPosition = transform.position.y - 0.23f;
        float zPosition = transform.position.z ;
//今現在のオブジェクトの位置から y 座標を調整して保管
        Vector3 np = new Vector3 (xPosition, yPosition + 0.3f, zPosition);
        bool a = false;//移動できる位置か判定
        for(float X = -4f; X <= 4f; X++ ){
            for(float Y = -8f; Y <= 8f; Y = Y + 2f ){
                for(float Z = -4f; Z <= 4f; Z++ ){
                    Vector3 onCell = new Vector3(xPosition + X, yPosition + Y, zPosition + Z);//true position
                    //クリックした位置を保管
                    Vector3 hitposition = hit.collider.gameObject.transform.position;

                    //x.0000~ -> xf
                    if(hit.collider.gameObject.transform.position.x >= X -0.5f
                        && hit.collider.gameObject.transform.position.x < X + 0.5f
                        && hit.collider.gameObject.transform.position.y >= Y -0.5f
                        && hit.collider.gameObject.transform.position.y < Y + 0.5f
                        && hit.collider.gameObject.transform.position.z >= Z -0.5f
                        && hit.collider.gameObject.transform.position.z < Z + 0.5f){//Ray hit position
                       hitposition = new Vector3(X ,Y ,Z);//整数調整
                    }

                    //移動できる範囲に収まっているかの判定
                    if(onCell.x == hitposition.x && onCell.z == hitposition.z && onCell.y == hitposition.y){
                    //実際に移動できるマスか判定
                       if (KnightJudge (X, Y, Z)){
                          a = true;//移動できるよう書き換え
                          if (X == 0f && Y == 0f && Z == 0f){//移動してない位置をクリックしていた場合
                             a = false;//移動できないよう書き換え
                          }
                          /*今の位置とクリックした位置の間のベクトルの大きさを保管する。
                          *しかし後に間にオブジェクトがあるか判定するためクリックした位置の駒と
                          *区別するため 1f 短くしてクリックマスにある駒に当たらないようにしている。
                          */
                          float dis = (hitposition - np).magnitude - 1.0f;//hitp <--> nowp
```

```
                    //全てのマスオブジェクトの情報を取得
                    GameObject[] cells = GameObject.FindGameObjectsWithTag("cell");


                    //今あるオブジェクトの位置とクリックした地点の距離を保管
                    Vector3 vecp = new Vector3 (hitposition.x - np.x, hitposition.y - np.y + 0.3f,
hitposition.z - np.z);

                    /*hits に現在のオブジェクト位置と移動先(クリック位置)の間の全てのオブジェクトの
                    *情報を格納していく。(格納される順番はランダム)
                    */
                    RaycastHit[] hits = Physics.RaycastAll (np, vecp, dis);
                    for (int i = 0; i < hits.Length; i++) {//何個間にあったか判定
                       //当たったオブジェクトのタグを保管する
                       string hitTag = hits [(int)i].collider.gameObject.tag;
                       //当たったオブジェクトが駒だった場合移動できないよう書き換え
                       if (hitTag == "piece2" || hitTag == "piece1") {
                           a = false;
                       }
                    }
                    break;
                }
            }
        }
    }
    return a;
  }

//ナイトからの相対的な位置判定
   public bool KnightJudge(float X, float Y, float Z){

      bool tf = false;

      if (Y == 0f && X == -1f && Z == -2f||
          Y == 0f && X == 1f && Z == -2f||
          Y == 0f && X == -2f && Z == -1f||
          Y == 0f && X == 2f && Z == -1f||
          Y == 0f && X == -2f && Z == 1f||
          Y == 0f && X == 2f && Z == 1f||
          Y == 0f && X == -1f && Z == 2f||
          Y == 0f && X == 1f && Z == 2f
          ){
        tf = true;
      } else if(Y == 2f && X == 0f && Z == -2f||
```

```
                Y == 2f && X == -2f && Z == 0f||
                Y == 2f && X == 2f && Z == 0f||
                Y == 2f && X == 0f && Z == 2f){
            tf = true;
        }else if(Y == 4f && X == 0f && Z == -1f||
                 Y == 4f && X == -1f && Z == 0f||
                 Y == 4f && X == 1f && Z == 0f||
                 Y == 4f && X == 0f && Z == 1f){
            tf = true;
        }   else if(Y == -2f && X == 0f && Z == -2f||
                    Y == -2f && X == -2f && Z == 0f||
                    Y == -2f && X == 2f && Z == 0f||
                    Y == -2f && X == 0f && Z == 2f){
            tf = true;
        }else if(Y == -4f && X == 0f && Z == -1f||
                 Y == -4f && X == -1f && Z == 0f||
                 Y == -4f && X == 1f && Z == 0f||
                 Y == -4f && X == 0f && Z == 1f){
            tf = true;
        }

        /**annallyJudge**/
        Vector3 anally0 = hit.collider.gameObject.transform.position;//クリックした座標を代入
        //クリックした座標の y 座標が+1f の位置を代入
        Vector3 anally = new Vector3 (anally0.x, anally0.y + 1.0f, anally0.z);
        //下方向に光線を発射しぶつかったゲームオブジェクトの情報を記憶。
        if(Physics.Raycast(anally,Vector3.down ,out hit10,3f)){
            //ぶつかったオブジェクトが味方の駒の時
            if(hit10.collider.gameObject.tag == "piece1"){
                tf = false;//飛び越え禁止
            }
        }

        return tf;
    }

    /*
    void OnGUI () {//テスト用テキスト表示メソッド
        GUI.TextField (new Rect (10, 10, 100, 200), t);
    }*/

}
```

- Unicorn

```
public class Unicorn: MonoBehaviour{
   RaycastHit hit, hit2, hit9, hit10;//オブジェクトの存在判定を行う。
   bool isSelect = false;//駒の選択状態判定用の変数である。
   Color orgColor,selColor = Color.yellow;//駒が選択された時の色を保管しておく変数である。
   //string t = "a";//テスト用のテキストである。

   void Start(){//初期条件
      orgColor = gameObject.renderer.material.color;//元々のオブジェクトの色に設定しておく
   }

   void Update(){//常に実行され続けている
/*プレイヤーのターンかどうかの判定をしている。
*Glpbal.flg とはグローバル変数(bool 型)であり
*'Global.flg == true'で白のターンである。
*/
      if (!Global.flg) {
         return;//Update()を終了させている。
      }
      if(Input.GetMouseButtonDown(0)){//マウスの左クリックが押された時
         Vector3 pos = Input.mousePosition;//押された時のマウスの位置を保管する。
         pos.z = 10.0f;//しかしマウスは奥行きが認識できないため z 座標を設定する。

         Ray ray = Camera.main.ScreenPointToRay(pos);//光線の作成

         if(Physics.Raycast(ray,out hit,100)){//当たったオブジェクトの判定
            //当たったオブジェクトがこのオブジェクトの場合
            if(hit.collider.gameObject.name == gameObject.name){
               isSelect = !isSelect;//f -> t//選択状態に移行
            }

            if(isSelect && hit.collider.gameObject.tag == "cell"){//マスがクリックされた時
               if(Control()){//移動可能か判定
                  Vector3 newpos = hit.collider.gameObject.transform.position;クリック位置保管
                  //敵駒判定用の光線作成
                  Vector3 newpos2 = new Vector3(newpos.x ,newpos.y + 1.0f ,newpos.z);
                  if(Physics.Raycast(newpos2,Vector3.down ,out hit2,3f)){
                     if(hit2.collider.gameObject.tag == "piece2"){
                        Destroy(hit2.collider.gameObject);//敵駒破壊
                     }
                  }
                  //クリックしたマスへ移動
                  transform.position = new Vector3(newpos.x, newpos.y + 0.23f, newpos.z);
                  //collision Time
```

```
                isSelect = false;//選択状態解除
                Flg fchange = new Flg();//白黒ターン入れ替えのフラグ変更
                Global.flg = fchange.FlgSetter(Global.flg);
            }
        }

        if(isSelect){//選択駒の色変え
            gameObject.renderer.material.color = selColor;
        }else{
            gameObject.renderer.material.color = orgColor;
        }
    }
  }


  public bool Control(){//Unicornpiece control
    float xPosition = transform.position.x ;//now position
    float yPosition = transform.position.y - 0.23f;
    float zPosition = transform.position.z ;
//今現在のオブジェクトの位置から y 座標を調整して保管
    Vector3 np = new Vector3 (xPosition, yPosition + 0.3f, zPosition);
    bool a = false;//移動できる位置か判定
    for(float X = -4f; X <= 4f; X++ ){
      for(float Y = -8f; Y <= 8f; Y = Y + 2f ){
        for(float Z = -4f; Z <= 4f; Z++ ){
          Vector3 onCell = new Vector3(xPosition + X, yPosition + Y, zPosition + Z);//true position
          //クリックした位置を保管
          Vector3 hitposition = hit.collider.gameObject.transform.position;

          //x.0000~ -> xf
          if(hit.collider.gameObject.transform.position.x >= X -0.5f
             && hit.collider.gameObject.transform.position.x < X + 0.5f
             && hit.collider.gameObject.transform.position.y >= Y -0.5f
             && hit.collider.gameObject.transform.position.y < Y + 0.5f
             && hit.collider.gameObject.transform.position.z >= Z -0.5f
             && hit.collider.gameObject.transform.position.z < Z + 0.5f){//Ray hit position
            hitposition = new Vector3(X ,Y ,Z);//整数調整
          }

          //移動できる範囲に収まっているかの判定
          if(onCell.x == hitposition.x && onCell.z == hitposition.z && onCell.y == hitposition.y){
          //実際に移動できるマスか判定
            if (UnicornJudge (X, Y, Z)){
              a = true;//移動できるよう書き換え
```

```
                    if (X == 0f && Y == 0f && Z == 0f){//移動してない位置をクリックしていた場合
                        a = false;//移動できないよう書き換え
                    }
                    /*今の位置とクリックした位置の間のベクトルの大きさを保管する。
                    *しかし後に間にオブジェクトがあるか判定するためクリックした位置の駒と
                    *区別するため 1f 短くしてクリックマスにある駒に当たらないようにしている。
                    */
                    float dis = (hitposition - np).magnitude - 1.0f;//hitp <--> nowp
                    //全てのマスオブジェクトの情報を取得
                    GameObject[] cells = GameObject.FindGameObjectsWithTag("cell");


                    //今あるオブジェクトの位置とクリックした地点の距離を保管
                    Vector3 vecp = new Vector3 (hitposition.x - np.x, hitposition.y - np.y + 0.3f,
hitposition.z - np.z);

                    /*hits に現在のオブジェクト位置と移動先(クリック位置)の間の全てのオブジェクトの
                    *情報を格納していく。(格納される順番はランダム)
                    */
                    RaycastHit[] hits = Physics.RaycastAll (np, vecp, dis);
                    for (int i = 0; i < hits.Length; i++) {//何個間にあったか判定
                        //当たったオブジェクトのタグを保管する
                        string hitTag = hits [(int)i].collider.gameObject.tag;
                        //当たったオブジェクトが駒だった場合移動できないよう書き換え
                        if (hitTag == "piece2" || hitTag == "piece1") {
                            a = false;
                        }
                    }
                    break;
                }
            }
        }
    }
    return a;
}

//ユニコーンの相対的な移動可能判定
public bool UnicornJudge(float X, float Y, float Z){

    bool tf = false;

    if (X == Z && X == Y/2 ||
        -X == Z && -X == Y/2 ||
        X == -Z && X == Y/2 ||
```

```
            -X == -Z && -X == Y/2 ||
            X == Z && X == -Y/2 ||
            -X == Z && -X == -Y/2 ||
            X == -Z && X == -Y/2 ||
            -X == -Z && -X == -Y/2) {
            tf = true;
            if(Y == 0f){
                tf = false;
            }

            /**annallyJudge**/
        Vector3 anally0 = hit.collider.gameObject.transform.position;//クリックした座標を代入
        //クリックした座標の y 座標が+1f の位置を代入
        Vector3 anally = new Vector3 (anally0.x, anally0.y + 1.0f, anally0.z);
        //下方向に光線を発射しぶつかったゲームオブジェクトの情報を記憶。
        if(Physics.Raycast(anally,Vector3.down ,out hit10,3f)){
            //ぶつかったオブジェクトが味方の駒の時
            if(hit10.collider.gameObject.tag == "piece1"){
                tf = false;//飛び越え禁止
            }
        }

        return tf;
    }

    /*
    void OnGUI () {//テスト用テキスト表示メソッド
        GUI.TextField (new Rect (10, 10, 100, 200), t);
    }*/

}
```

- Bishop

```
public class Bishop : MonoBehaviour{
    RaycastHit hit, hit2, hit9, hit10;//オブジェクトの存在判定を行う。
    bool isSelect = false;//駒の選択状態判定用の変数である。
    Color orgColor,selColor = Color.yellow;//駒が選択された時の色を保管しておく変数である。
    //string t = "a";//テスト用のテキストである。

    void Start(){//初期条件
        orgColor = gameObject.renderer.material.color;//元々のオブジェクトの色に設定しておく
    }
```

```
   void Update(){//常に実行され続けている
/*プレイヤーのターンかどうかの判定をしている。
*Glpbal.flg とはグローバル変数(bool 型)であり
*'Global.flg == true'で白のターンである。
*/
      if (!Global.flg) {
         return;//Update()を終了させている。
      }
      if(Input.GetMouseButtonDown(0)){//マウスの左クリックが押された時
         Vector3 pos = Input.mousePosition;//押された時のマウスの位置を保管する。
         pos.z = 10.0f;//しかしマウスは奥行きが認識できないため z 座標を設定する。

         Ray ray = Camera.main.ScreenPointToRay(pos);//光線の作成

         if(Physics.Raycast(ray,out hit,100)){//当たったオブジェクトの判定
            //当たったオブジェクトがこのオブジェクトの場合
            if(hit.collider.gameObject.name == gameObject.name){
               isSelect = !isSelect;//f -> t//選択状態に移行
            }

            if(isSelect && hit.collider.gameObject.tag == "cell"){//マスがクリックされた時
               if(Control()){//移動可能か判定
                  Vector3 newpos = hit.collider.gameObject.transform.position;クリック位置保管
                  //敵駒判定用の光線作成
                  Vector3 newpos2 = new Vector3(newpos.x ,newpos.y + 1.0f ,newpos.z);
                  if(Physics.Raycast(newpos2,Vector3.down ,out hit2,3f)){
                     if(hit2.collider.gameObject.tag == "piece2"){
                        Destroy(hit2.collider.gameObject);//敵駒破壊
                     }
                  }
                  //クリックしたマスへ移動
                  transform.position = new Vector3(newpos.x, newpos.y + 0.23f, newpos.z);
                  //collision Time
                  isSelect = false;//選択状態解除
                  Flg fchange = new Flg();//白黒ターン入れ替えのフラグ変更
                  Global.flg = fchange.FlgSetter(Global.flg);
               }
            }

            if(isSelect){//選択駒の色変え
               gameObject.renderer.material.color = selColor;
            }else{
               gameObject.renderer.material.color = orgColor;
            }
         }
```

```
    }
  }

  public bool Control(){//Bishoppiece control
    float xPosition = transform.position.x ;//now position
    float yPosition = transform.position.y - 0.23f;
    float zPosition = transform.position.z ;
//今現在のオブジェクトの位置から y 座標を調整して保管
    Vector3 np = new Vector3 (xPosition, yPosition + 0.3f, zPosition);
    bool a = false;//移動できる位置か判定
    for(float X = -4f; X <= 4f; X++ ){
      for(float Y = -8f; Y <= 8f; Y = Y + 2f ){
        for(float Z = -4f; Z <= 4f; Z++ ){
          Vector3 onCell = new Vector3(xPosition + X, yPosition + Y, zPosition + Z);//true position
          //クリックした位置を保管
          Vector3 hitposition = hit.collider.gameObject.transform.position;

          //x.0000~ -> xf
          if(hit.collider.gameObject.transform.position.x >= X -0.5f
            && hit.collider.gameObject.transform.position.x < X + 0.5f
            && hit.collider.gameObject.transform.position.y >= Y -0.5f
            && hit.collider.gameObject.transform.position.y < Y + 0.5f
            && hit.collider.gameObject.transform.position.z >= Z -0.5f
            && hit.collider.gameObject.transform.position.z < Z + 0.5f){//Ray hit position
           hitposition = new Vector3(X ,Y ,Z);//整数調整
          }

          //移動できる範囲に収まっているかの判定
          if(onCell.x == hitposition.x && onCell.z == hitposition.z && onCell.y == hitposition.y){
          //実際に移動できるマスか判定
            if (BishopJudge (X, Y, Z)){
              a = true;//移動できるよう書き換え
              if (X == 0f && Y == 0f && Z == 0f){//移動してない位置をクリックしていた場合
                a = false;//移動できないよう書き換え
              }
              /*今の位置とクリックした位置の間のベクトルの大きさを保管する。
              *しかし後に間にオブジェクトがあるか判定するためクリックした位置の駒と
              *区別するため 1f 短くしてクリックマスにある駒に当たらないようにしている。
              */
              float dis = (hitposition - np).magnitude - 1.0f;//hitp <--> nowp
              //全てのマスオブジェクトの情報を取得
              GameObject[] cells = GameObject.FindGameObjectsWithTag("cell");


              //今あるオブジェクトの位置とクリックした地点の距離を保管
```

```
                Vector3 vecp = new Vector3 (hitposition.x - np.x, hitposition.y - np.y + 0.3f,
hitposition.z - np.z);

                /*hits に現在のオブジェクト位置と移動先(クリック位置)の間の全てのオブジェクトの
                *情報を格納していく。(格納される順番はランダム)
                */
                RaycastHit[] hits = Physics.RaycastAll (np, vecp, dis);
                for (int i = 0; i < hits.Length; i++) {//何個間にあったか判定
                  //当たったオブジェクトのタグを保管する
                  string hitTag = hits [(int)i].collider.gameObject.tag;
                  //当たったオブジェクトが駒だった場合移動できないよう書き換え
                  if (hitTag == "piece2" || hitTag == "piece1") {
                    a = false;
                  }
                }
                break;
              }
            }
          }
        }
      }
      return a;
    }

    //ビショップからの相対的な位置判定
    public bool BishopJudge(float X, float Y, float Z){

      bool tf = false;

      if (Y == 0f && X == Z ||
      Y == 0f && X == -Z ||
      Y == 0f && -X == Z ||
      Y == 0f && -X == -Z) {
        tf = true;
      }else if(X == 0 && Z == Y / 2||
      X == 0 && -Z == Y / 2||
      Z == 0 && X == Y / 2||
      Z == 0 && -X == Y / 2){
        tf = true;
      }
        /**annallyJudge**/
      Vector3 anally0 = hit.collider.gameObject.transform.position;//クリックした座標を代入
      //クリックした座標の y 座標が+1f の位置を代入
      Vector3 anally = new Vector3 (anally0.x, anally0.y + 1.0f, anally0.z);
      //下方向に光線を発射しぶつかったゲームオブジェクトの情報を記憶。
```

```
        if(Physics.Raycast(anally,Vector3.down ,out hit10,3f)){
            //ぶつかったオブジェクトが味方の駒の時
            if(hit10.collider.gameObject.tag == "piece1"){
                tf = false;//飛び越え禁止
            }
        }

        return tf;
    }

    /*
    void OnGUI () {//テスト用テキスト表示メソッド
        GUI.TextField (new Rect (10, 10, 100, 200), t);
    }*/

}
```

- Rook

```
public class Rook : MonoBehaviour{
    RaycastHit hit, hit2, hit9, hit10;//オブジェクトの存在判定を行う。
    bool isSelect = false;//駒の選択状態判定用の変数である。
    Color orgColor,selColor = Color.yellow;//駒が選択された時の色を保管しておく変数である。
    //string t = "a";//テスト用のテキストである。

    void Start(){//初期条件
        orgColor = gameObject.renderer.material.color;//元々のオブジェクトの色に設定しておく
    }

    void Update(){//常に実行され続けている
/*プレイヤーのターンかどうかの判定をしている。
*Glpbal.flg とはグローバル変数(bool 型)であり
*'Global.flg == true'で白のターンである。
*/
        if (!Global.flg) {
            return;//Update()を終了させている。
        }
        if(Input.GetMouseButtonDown(0)){//マウスの左クリックが押された時
            Vector3 pos = Input.mousePosition;//押された時のマウスの位置を保管する。
            pos.z = 10.0f;//しかしマウスは奥行きが認識できないため z 座標を設定する。

            Ray ray = Camera.main.ScreenPointToRay(pos);//光線の作成

            if(Physics.Raycast(ray,out hit,100)){//当たったオブジェクトの判定
```

```
            //当たったオブジェクトがこのオブジェクトの場合
            if(hit.collider.gameObject.name == gameObject.name){
               isSelect = !isSelect;//f -> t//選択状態に移行
            }

            if(isSelect && hit.collider.gameObject.tag == "cell"){//マスがクリックされた時
               if(Control()){//移動可能か判定
                  Vector3 newpos = hit.collider.gameObject.transform.position;クリック位置保管
                  //敵駒判定用の光線作成
                  Vector3 newpos2 = new Vector3(newpos.x ,newpos.y + 1.0f ,newpos.z);
                  if(Physics.Raycast(newpos2,Vector3.down ,out hit2,3f)){
                     if(hit2.collider.gameObject.tag == "piece2"){
                        Destroy(hit2.collider.gameObject);//敵駒破壊
                     }
                  }
                  //クリックしたマスへ移動
                  transform.position = new Vector3(newpos.x, newpos.y + 0.23f, newpos.z);
                  //collision Time
                  isSelect = false;//選択状態解除
                  Flg fchange = new Flg();//白黒ターン入れ替えのフラグ変更
                  Global.flg = fchange.FlgSetter(Global.flg);
               }
            }

            if(isSelect){//選択駒の色変え
               gameObject.renderer.material.color = selColor;
            }else{
               gameObject.renderer.material.color = orgColor;
            }
         }
      }
   }



   public bool Control(){//Rookpiece control
      float xPosition = transform.position.x ;//now position
      float yPosition = transform.position.y - 0.23f;
      float zPosition = transform.position.z ;
//今現在のオブジェクトの位置から y 座標を調整して保管
      Vector3 np = new Vector3 (xPosition, yPosition + 0.3f, zPosition);
      bool a = false;//移動できる位置か判定
      for(float X = -4f; X <= 4f; X++ ){
         for(float Y = -8f; Y <= 8f; Y = Y + 2f ){
            for(float Z = -4f; Z <= 4f; Z++ ){
```

```
Vector3 onCell = new Vector3(xPosition + X, yPosition + Y, zPosition + Z);//true position
//クリックした位置を保管
Vector3 hitposition = hit.collider.gameObject.transform.position;

//x.0000~ -> xf
if(hit.collider.gameObject.transform.position.x >= X -0.5f
    && hit.collider.gameObject.transform.position.x < X + 0.5f
    && hit.collider.gameObject.transform.position.y >= Y -0.5f
    && hit.collider.gameObject.transform.position.y < Y + 0.5f
    && hit.collider.gameObject.transform.position.z >= Z -0.5f
    && hit.collider.gameObject.transform.position.z < Z + 0.5f){//Ray hit position
   hitposition = new Vector3(X ,Y ,Z);//整数調整
}

//移動できる範囲に収まっているかの判定
if(onCell.x == hitposition.x && onCell.z == hitposition.z && onCell.y == hitposition.y){
//実際に移動できるマスか判定
   if (RookJudge (X, Y, Z)){
      a = true;//移動できるよう書き換え
      if (X == 0f && Y == 0f && Z == 0f){//移動してない位置をクリックしていた場合
         a = false;//移動できないよう書き換え
      }
      /*今の位置とクリックした位置の間のベクトルの大きさを保管する。
      *しかし後に間にオブジェクトがあるか判定するためクリックした位置の駒と
      *区別するため 1f 短くしてクリックマスにある駒に当たらないようにしている。
      */
      float dis = (hitposition - np).magnitude - 1.0f;//hitp <--> nowp
      //全てのマスオブジェクトの情報を取得
      GameObject[] cells = GameObject.FindGameObjectsWithTag("cell");


      //今あるオブジェクトの位置とクリックした地点の距離を保管
      Vector3 vecp = new Vector3 (hitposition.x - np.x, hitposition.y - np.y + 0.3f,
hitposition.z - np.z);

      /*hits に現在のオブジェクト位置と移動先(クリック位置)の間の全てのオブジェクトの
      *情報を格納していく。(格納される順番はランダム)
      */
      RaycastHit[] hits = Physics.RaycastAll (np, vecp, dis);
      for (int i = 0; i < hits.Length; i++) {//何個間にあったか判定
         //当たったオブジェクトのタグを保管する
         string hitTag = hits [(int)i].collider.gameObject.tag;
         //当たったオブジェクトが駒だった場合移動できないよう書き換え
         if (hitTag == "piece2" || hitTag == "piece1") {
            a = false;
```

```
                    }
                }
                break;
            }
        }
    }
  }
  return a;
}

//ルークからの相対的な位置判定
  public bool RookJudge(float X, float Y, float Z){

    bool tf = false;

    if (Y == 0f && X == 0f ||
        Y == 0f && Z == 0f) {
      tf = true;
    }else if(Y != 0f && X == 0f && Z == 0f){
      tf = true;
    }

      /**annallyJudge**/
    Vector3 anally0 = hit.collider.gameObject.transform.position;//クリックした座標を代入
    //クリックした座標の y 座標が+1f の位置を代入
    Vector3 anally = new Vector3 (anally0.x, anally0.y + 1.0f, anally0.z);
    //下方向に光線を発射しぶつかったゲームオブジェクトの情報を記憶。
    if(Physics.Raycast(anally,Vector3.down ,out hit10,3f)){
      //ぶつかったオブジェクトが味方の駒の時
      if(hit10.collider.gameObject.tag == "piece1"){
        tf = false;//飛び越え禁止
      }
    }

    return tf;
  }

  /*
  void OnGUI () {//テスト用テキスト表示メソッド
    GUI.TextField (new Rect (10, 10, 100, 200), t);
  }*/

}
```

- Pawn

```
public class Pawn : MonoBehaviour{
   public class Rook : MonoBehaviour{
   RaycastHit hit, hit2, hit9, hit10;//オブジェクトの存在判定を行う。
   bool isSelect = false;//駒の選択状態判定用の変数である。
   Color orgColor,selColor = Color.yellow;//駒が選択された時の色を保管しておく変数である。
   //string t = "a";//テスト用のテキストである。

   void Start(){//初期条件
      orgColor = gameObject.renderer.material.color;//元々のオブジェクトの色に設定しておく
   }

   void Update(){//常に実行され続けている
/*プレイヤーのターンかどうかの判定をしている。
*Glpbal.flg とはグローバル変数(bool 型)であり
*'Global.flg == true'で白のターンである。
*/
      if (!Global.flg) {
         return;//Update()を終了させている。
      }
      if(Input.GetMouseButtonDown(0)){//マウスの左クリックが押された時
         Vector3 pos = Input.mousePosition;//押された時のマウスの位置を保管する。
         pos.z = 10.0f;//しかしマウスは奥行きが認識できないため z 座標を設定する。

         Ray ray = Camera.main.ScreenPointToRay(pos);//光線の作成

         if(Physics.Raycast(ray,out hit,100)){//当たったオブジェクトの判定
            //当たったオブジェクトがこのオブジェクトの場合
            if(hit.collider.gameObject.name == gameObject.name){
               isSelect = !isSelect;//f -> t//選択状態に移行
            }

            if(isSelect && hit.collider.gameObject.tag == "cell"){//マスがクリックされた時
               if(Control()){//移動可能か判定
                  Vector3 newpos = hit.collider.gameObject.transform.position;クリック位置保管
                  //敵駒判定用の光線作成
                  Vector3 newpos2 = new Vector3(newpos.x ,newpos.y + 1.0f ,newpos.z);
                  if(Physics.Raycast(newpos2,Vector3.down ,out hit2,3f)){
                     if(hit2.collider.gameObject.tag == "piece2"){
                        Destroy(hit2.collider.gameObject);//敵駒破壊
                     }
                  }
                  //クリックしたマスへ移動
```

```
                transform.position = new Vector3(newpos.x, newpos.y + 0.23f, newpos.z);
                //collision Time
                isSelect = false;//選択状態解除
                Flg fchange = new Flg();//白黒ターン入れ替えのフラグ変更
                Global.flg = fchange.FlgSetter(Global.flg);
            }
        }

        if(isSelect){//選択駒の色変え
            gameObject.renderer.material.color = selColor;
        }else{
            gameObject.renderer.material.color = orgColor;
        }
    }
  }

  public bool Control(){//Pawnpiece control
    float xPosition = transform.position.x ;//now position
    float yPosition = transform.position.y - 0.23f;
    float zPosition = transform.position.z ;
//今現在のオブジェクトの位置から y 座標を調整して保管
    Vector3 np = new Vector3 (xPosition, yPosition + 0.3f, zPosition);
    bool a = false;//移動できる位置か判定
    for(float X = -4f; X <= 4f; X++ ){
      for(float Y = -8f; Y <= 8f; Y = Y + 2f ){
        for(float Z = -4f; Z <= 4f; Z++ ){
          Vector3 onCell = new Vector3(xPosition + X, yPosition + Y, zPosition + Z);//true position
          //クリックした位置を保管
          Vector3 hitposition = hit.collider.gameObject.transform.position;

          //x.0000~ -> xf
          if(hit.collider.gameObject.transform.position.x >= X -0.5f
             && hit.collider.gameObject.transform.position.x < X + 0.5f
             && hit.collider.gameObject.transform.position.y >= Y -0.5f
             && hit.collider.gameObject.transform.position.y < Y + 0.5f
             && hit.collider.gameObject.transform.position.z >= Z -0.5f
             && hit.collider.gameObject.transform.position.z < Z + 0.5f){//Ray hit position
            hitposition = new Vector3(X ,Y ,Z);//整数調整
          }

          //移動できる範囲に収まっているかの判定
          if(onCell.x == hitposition.x && onCell.z == hitposition.z && onCell.y == hitposition.y){
          //実際に移動できるマスか判定
            if (PawnJudge (X, Y, Z)){
```

```
                    a = true;//移動できるよう書き換え
                    if (X == 0f && Y == 0f && Z == 0f){//移動してない位置をクリックしていた場合
                        a = false;//移動できないよう書き換え
                    }
                    /*今の位置とクリックした位置の間のベクトルの大きさを保管する。
                    *しかし後に間にオブジェクトがあるか判定するためクリックした位置の駒と
                    *区別するため 1f 短くしてクリックマスにある駒に当たらないようにしている。
                    */
                    float dis = (hitposition - np).magnitude - 1.0f;//hitp <--> nowp
                    //全てのマスオブジェクトの情報を取得
                    GameObject[] cells = GameObject.FindGameObjectsWithTag("cell");


                    //今あるオブジェクトの位置とクリックした地点の距離を保管
                    Vector3 vecp = new Vector3 (hitposition.x - np.x, hitposition.y - np.y + 0.3f,
hitposition.z - np.z);

                    /*hits に現在のオブジェクト位置と移動先(クリック位置)の間の全てのオブジェクトの
                    *情報を格納していく。(格納される順番はランダム)
                    */
                    RaycastHit[] hits = Physics.RaycastAll (np, vecp, dis);
                    for (int i = 0; i < hits.Length; i++) {//何個間にあったか判定
                        //当たったオブジェクトのタグを保管する
                        string hitTag = hits [(int)i].collider.gameObject.tag;
                        //当たったオブジェクトが駒だった場合移動できないよう書き換え
                        if (hitTag == "piece2" || hitTag == "piece1") {
                            a = false;
                        }
                    }
                    break;
                }
            }
        }
    }
    return a;
}

//ポーンの相対的な位置判定
public bool PawnJudge(float X, float Y, float Z){

    bool tf = false;

    if (Y == 0f && X == 0f && Z == -1f||
        Y == 2f && X == 0f && Z == 0f) {
```

```
            tf = true;
        }

            /**annallyJudge**/
        Vector3 anally0 = hit.collider.gameObject.transform.position;//クリックした座標を代入
        //クリックした座標の y 座標が+1f の位置を代入
        Vector3 anally = new Vector3 (anally0.x, anally0.y + 1.0f, anally0.z);
        //下方向に光線を発射しぶつかったゲームオブジェクトの情報を記憶。
        if(Physics.Raycast(anally,Vector3.down ,out hit10,3f)){
            //ぶつかったオブジェクトが味方の駒の時
            if(hit10.collider.gameObject.tag == "piece1"){
                tf = false;//飛び越え禁止
            }
        }

        return tf;
    }

    /*
    void OnGUI () {//テスト用テキスト表示メソッド
        GUI.TextField (new Rect (10, 10, 100, 200), t);
    }*/

}
```

- Kingb

```
public class Kingb : MonoBehaviour{
    RaycastHit hit, hit2, hit9, hit10; //オブジェクトの存在の判定を行う。
  RaycastHit cellhit; //オブジェクトの存在の判定を行う。
  bool isSelect = false; //駒の選択状態判定用の変数である。
  Color orgColor,selColor = Color.yellow; //駒が選択された時の色を保管しておく変数である。
  //string t = "a"; //テスト用のテキストである

  void Start(){ //初期条件
    orgColor = gameObject.renderer.material.color; //元々のオブジェクトの色に設定しておく。
  }

void Update(){ //常に実行され続けている。
    GameJudge (); //ゲーム終了条件を満たしているかの判定メソッド
    /*  プレイヤーのターンかどうかの判定をしている。
    *Global.flg とはグローバル変数(bool 型)であり
    * 'Global.flg == false'で黒のターンである。
    */
```

```
   if (Global.flg ) { //敵のターンだった場合
    return; //Update()を終了させている。
   }
   if(Input.GetMouseButtonDown(0)){ //マウスの左クリックが押された時
    Vector3 pos = Input.mousePosition; //押された時のマウスの位置を保管する。
    pos.z = 10.0f; //しかしマウスでは奥行きが認識できないため z 座標を設定する。
/*カメラワークの位置から pos 座標へ向けて
*直線に伸びる光線を作成する。
*/
    Ray ray = Camera.main.ScreenPointToRay(pos);

/*直線に 100 の距離の
*光線を飛ばし当たったゲームオブジェクトがあるか。
*あるなら hit に代入
*/
    if(Physics.Raycast(ray,out hit,100)){

/*hit に代入されたゲームオブジェクトの名前が
*/このスクリプトのオブジェクトであるか
      if(hit.collider.gameObject.name == gameObject.name){
       isSelect = !isSelect;//f -> t //このオブジェクトに当たったなら選択状態へ移行
      }
// クリックした位置がチェス盤であった場合
      if(isSelect && hit.collider.gameObject.tag == "cell"){
       if(Control()){ //クリックされたマスが移動可能マスか判定
        Vector3 newpos = hit.collider.gameObject.transform.position; //クリック位置を保管
        /*クリック位置を保管して y 座標だけ 1f 大きい値に設定する。
        *
        */
        Vector3 newpos2 = new Vector3(newpos.x ,newpos.y + 1.0f ,newpos.z);
/*クリックされた位置の 1f 高い位置から真下に 3f の長さの光線を放ち当たったオブジェクトを
*hit2 に保管する。
*/
       if(Physics.Raycast(newpos2,Vector3.down ,out hit2,3f)){
        if(hit2.collider.gameObject.tag == "piece1"){//光線の当たったオブジェクトが敵駒である場合
         Destroy(hit2.collider.gameObject);//当たった駒の名前を取得しゲームから取り除く。
        }
       }
       //クリックしたマスへ移動(y 座標の+0.23f はマスの厚みを配慮)
        transform.position = new Vector3(newpos.x, newpos.y + 0.23f, newpos.z);
        //collision Time
        isSelect = false; //選択状態の解除
        Flg fchange = new Flg(); //Flg クラスのインスタンスの作成
        /*グローバル変数である Global.flg の T or F の入れ替え
        *これによって白駒が動かせなくなり黒駒が動かせるようになる。
```

```
        */
        Global.flg = fchange.FlgSetter(Global.flg);
       }
      }

      if(isSelect){ //選択状態の時
       gameObject.renderer.material.color = selColor; //駒の色を変える。(黄色)
      }else{ //選択状態でない場合
       gameObject.renderer.material.color = orgColor; //駒の色を元の色に戻す。
      }
     }
    }
   }

//King の動けるマスの判定を行う
  public bool Control(){//King piece control
    float xPosition = transform.position.x ;
    float yPosition = transform.position.y - 0.23f;
    float zPosition = transform.position.z ;

    bool a = false;
    for(float X = -1f; X <= 1f; X++ ){
     for(float Y = -2f; Y <= 2f; Y = Y + 2f){
      for(float Z = -1f; Z <= 1f; Z++ ){
       //King の現在の座標からの絶対位置を表す
       Vector3 onCell = new Vector3(xPosition + X, yPosition + Y, zPosition + Z);//true position
       Vector3 hitposition = hit.collider.gameObject.transform.position;//King の移動できる座標の保管

       if(hit.collider.gameObject.transform.position.x >= X -0.5f&&
hit.collider.gameObject.transform.position.x < X + 0.5f&&
hit.collider.gameObject.transform.position.y >= Y -0.5f &&
hit.collider.gameObject.transform.position.y < Y + 0.5f&& hit.collider.gameObject.transform.position.z
>= Z -0.5f && hit.collider.gameObject.transform.position.z < Z + 0.5f){//Ray hit position
        hitposition = new Vector3(X ,Y ,Z); //光線がマスに当たった座標は整数値でないため整数値に変換
       }

//光線が当たった座標が King の動ける位置に収まっているか
       if(onCell.x == hitposition.x && onCell.z == hitposition.z && onCell.y == hitposition.y){

        a = true; //bool の書き換え
        /**annallyJudge**/
        Vector3 anally0 = hit.collider.gameObject.transform.position; //クリックした座標を代入
        //クリックした座標の y 座標が+1f の位置を代入
        Vector3 anally = new Vector3 (anally0.x, anally0.y + 1.0f, anally0.z);
        /*クリックした座標の y 座標が+1f の位置から下方向に 3f の長さの光線を発射する。
```

```
        *ゲームオブジェクトにぶつかったら hit10 に情報が記憶される。
        */
        if(Physics.Raycast(anally,Vector3.down ,out hit10,3f)){

          if(hit10.collider.gameObject.tag == "piece2"){//hit10 は味方の駒であるか
            a = false;//味方の駒のあるマスであると移動できないため false に書き換え
          }
        }

        if(X == 0f && Y == 0f && Z == 0f){//今選択されている駒のある位置(移動しない)
          a = false;//移動しないのは false
        }
          break;
       }
      }
     }
    }
   return a;//移動できるマスなら T 不可なら F
  }


//ゲーム終了判定メソッド
  public void GameJudge(){
   bool whiteKing = false;//白キングがあるかの判定
   bool blackKing = false;//黒キングがあるかの判定
   for (float X = -2f; X <= 2f; X++) {//全てのマスの探索
    for (float Y = -4f; Y <= 4f; Y = Y + 2f) {
     for (float Z = -2f; Z <= 2f; Z++) {
      Vector3 onCell = new Vector3 (X, Y + 1f, Z);//全てのマスの y 座標の 1f 大きい位置を保管
      if (Physics.Raycast (onCell, Vector3.down, out hit, 3f)) {//全てのマスのオブジェクトを調べる
       if (hit.collider.gameObject.name == "Kingw") {//白のキングの有無
        whiteKing = true;//白のキングがあれば true
       } else if (hit.collider.gameObject.name == "Kingb") {//黒のキングの有無
        blackKing = true;//黒のキングがあれば true
       }
      }
     }
    }
   }
   for (float X = -2f; X <= 2f; X++) {//全てのマスの探索
    for (float Y = -4f; Y <= 4f; Y = Y + 2f) {
     for (float Z = -2f; Z <= 2f; Z++) {
      Vector3 onCell = new Vector3 (X, Y + 1f, Z); //全てのマスの y 座標の 1f 大きい位置を保管
      if (Physics.Raycast (onCell, Vector3.down, out hit9, 3f)) {//全てのマスのオブジェクトを調べる
      //白キングのある状態で黒キングの有無の判定
```

```
        if (hit.collider.gameObject.name == "Kingb" && whiteKing) {
          blackKing = true;//黒のキングがあれば true
          //黒キングのある状態で白キングの有無の判定
        } else if (hit.collider.gameObject.name == "Kingw" && blackKing) {
          whiteKing = true;//白のキングがあれば true
        }
       }
      }
     }
    }

    if (whiteKing && blackKing) {//白黒のキングがあるか
     } else {//片一方もしくは両方が存在しない場合
     GameOver (whiteKing, blackKing); //ゲーム終了判定
    }
   }

//ゲーム終了判定メソッド
  public void GameOver(bool whiteKing,bool blackKing){
    if(whiteKing && !blackKing){//白キング T 黒キング F
      WhiteWin();//白勝利メソッド呼び出し
    }else if(!whiteKing && blackKing){//白キング F 黒キング T
      BlackWin();//黒勝利メソッド呼び出し
    }else{//白キング F 黒キング F
      DrawGame();//ドローメソッド呼び出し
    }
  }
//白勝利メソッド
  public void WhiteWin(){
    Game g = new Game();//Game クラスのインスタンス作成
    Global.tex = "GAME OVER¥n" +
    "WHITE WIN!";//ゲーム終了時のテキストをグローバル変数である Global.tex に代入
    gameov();//テキスト表示用のメソッド呼び出し
  }

  public void BlackWin(){//Game クラスのインスタンス作成
    Game g = new Game();
    Global.tex = "GAME OVER¥n" +
    "BLACK WIN!"; //ゲーム終了時のテキストをグローバル変数である Global.tex に代入
    gameov();//テキスト表示用のメソッド呼び出し
  }

  public void DrawGame(){//Game クラスのインスタンス作成
    Game g = new Game();
    Global.tex = "GAME OVER¥n" +
```

```
    "DRAW GAME";//ゲーム終了時のテキストをグローバル変数である Global.tex に代入
   gameov();//テキスト表示用のメソッド呼び出し
  }

  public void gameov(){//テキスト表示を行う
   GameObject.Find ("GameText").guiText.text = "" + Global.tex;
  }

  /*
  void OnGUI () {//テスト用のテキスト表示メソッド
   GUI.TextField (new Rect (10, 10, 100, 200), t);
  }
  */
}
```

- Queenb

```
public class Queenb: MonoBehaviour{
    RaycastHit hit, hit2, hit9, hit10;//オブジェクトの存在判定を行う。
    bool isSelect = false;//駒の選択状態判定用の変数である。
    Color orgColor,selColor = Color.yellow;//駒が選択された時の色を保管しておく変数である。
    //string t = "a";//テスト用のテキストである。

    void Start(){//初期条件
        orgColor = gameObject.renderer.material.color;//元々のオブジェクトの色に設定しておく
    }

    void Update(){//常に実行され続けている
/*プレイヤーのターンかどうかの判定をしている。
*Glpbal.flg とはグローバル変数(bool 型)であり
*'Global.flg == false'で黒のターンである。
*/
        if (Global.flg) {
            return;//Update()を終了させている。
        }
        if(Input.GetMouseButtonDown(0)){//マウスの左クリックが押された時
            Vector3 pos = Input.mousePosition;//押された時のマウスの位置を保管する。
            pos.z = 10.0f;//しかしマウスは奥行きが認識できないため z 座標を設定する。

            Ray ray = Camera.main.ScreenPointToRay(pos);//光線の作成

            if(Physics.Raycast(ray,out hit,100)){//当たったオブジェクトの判定
                //当たったオブジェクトがこのオブジェクトの場合
                if(hit.collider.gameObject.name == gameObject.name){
```

```
            isSelect = !isSelect;//f -> t//選択状態に移行
        }

        if(isSelect && hit.collider.gameObject.tag == "cell"){//マスがクリックされた時
            if(Control()){//移動可能か判定
                Vector3 newpos = hit.collider.gameObject.transform.position;クリック位置保管
                //敵駒判定用の光線作成
                Vector3 newpos2 = new Vector3(newpos.x ,newpos.y + 1.0f ,newpos.z);
                if(Physics.Raycast(newpos2,Vector3.down ,out hit2,3f)){
                    if(hit2.collider.gameObject.tag == "piece1"){
                        Destroy(hit2.collider.gameObject);//敵駒破壊
                    }
                }
                //クリックしたマスへ移動
                transform.position = new Vector3(newpos.x, newpos.y + 0.23f, newpos.z);
                //collision Time
                isSelect = false;//選択状態解除
                Flg fchange = new Flg();//白黒ターン入れ替えのフラグ変更
                Global.flg = fchange.FlgSetter(Global.flg);
            }
        }

        if(isSelect){//選択駒の色変え
            gameObject.renderer.material.color = selColor;
        }else{
            gameObject.renderer.material.color = orgColor;
        }
    }
  }

  public bool Control(){//Queenpiece control
    float xPosition = transform.position.x ;//now position
    float yPosition = transform.position.y - 0.23f;
    float zPosition = transform.position.z ;
//今現在のオブジェクトの位置から y 座標を調整して保管
    Vector3 np = new Vector3 (xPosition, yPosition + 0.3f, zPosition);
    bool a = false;//移動できる位置か判定
    for(float X = -4f; X <= 4f; X++ ){
      for(float Y = -8f; Y <= 8f; Y = Y + 2f ){
        for(float Z = -4f; Z <= 4f; Z++ ){
          Vector3 onCell = new Vector3(xPosition + X, yPosition + Y, zPosition + Z);//true position
          //クリックした位置を保管
          Vector3 hitposition = hit.collider.gameObject.transform.position;
```

```
            //x.0000~ -> xf
            if(hit.collider.gameObject.transform.position.x >= X -0.5f
                && hit.collider.gameObject.transform.position.x < X + 0.5f
                && hit.collider.gameObject.transform.position.y >= Y -0.5f
                && hit.collider.gameObject.transform.position.y < Y + 0.5f
                && hit.collider.gameObject.transform.position.z >= Z -0.5f
                && hit.collider.gameObject.transform.position.z < Z + 0.5f){//Ray hit position
               hitposition = new Vector3(X ,Y ,Z);//整数調整
            }

            //移動できる範囲に収まっているかの判定
            if(onCell.x == hitposition.x && onCell.z == hitposition.z && onCell.y == hitposition.y){
            //実際に移動できるマスか判定
               if (QueenJudge (X, Y, Z)){
                  a = true;//移動できるよう書き換え
                  if (X == 0f && Y == 0f && Z == 0f){//移動してない位置をクリックしていた場合
                     a = false;//移動できないよう書き換え
                  }
                  /*今の位置とクリックした位置の間のベクトルの大きさを保管する。
                  *しかし後に間にオブジェクトがあるか判定するためクリックした位置の駒と
                  *区別するため 1f 短くしてクリックマスにある駒に当たらないようにしている。
                  */
                  float dis = (hitposition - np).magnitude - 1.0f;//hitp <--> nowp
                  //全てのマスオブジェクトの情報を取得
                  GameObject[] cells = GameObject.FindGameObjectsWithTag("cell");


                  //今あるオブジェクトの位置とクリックした地点の距離を保管
                  Vector3 vecp = new Vector3 (hitposition.x - np.x, hitposition.y - np.y + 0.3f,
hitposition.z - np.z);

                  /*hits に現在のオブジェクト位置と移動先(クリック位置)の間の全てのオブジェクトの
                  *情報を格納していく。(格納される順番はランダム)
                  */
                  RaycastHit[] hits = Physics.RaycastAll (np, vecp, dis);
                  for (int i = 0; i < hits.Length; i++) {//何個間にあったか判定
                     //当たったオブジェクトのタグを保管する
                     string hitTag = hits [(int)i].collider.gameObject.tag;
                     //当たったオブジェクトが駒だった場合移動できないよう書き換え
                     if (hitTag == "piece2" || hitTag == "piece1") {
                        a = false;
                     }
                  }
                  break;
               }
```

```
            }
        }
    }
  }
  return a;
}


public bool QueenJudge(float X, float Y, float Z){
  bool tf = false;

//クイーンの現在の位置からの相対的な移動可能判定
  if (X == Z && X == Y/2 | |
      -X == Z && -X == Y/2 | |
      X == -Z && X == Y/2 | |
      -X == -Z && -X == Y/2 | |
      X == Z && X == -Y/2 | |
      -X == Z && -X == -Y/2 | |
      X == -Z && X == -Y/2 | |
      -X == -Z && -X == -Y/2) {
    tf = true;
  } else if (Y == 0f && X == Z | |
               Y == 0f && X == -Z | |
               Y == 0f && -X == Z | |
               Y == 0f && -X == -Z) {
    tf = true;
  }else if(X == 0 && Z == Y / 2| |
             X == 0 && -Z == Y / 2| |
             Z == 0 && X == Y / 2| |
             Z == 0 && -X == Y / 2){
    tf = true;
  }else if (Y == 0f && X == 0f | |
              Y == 0f && Z == 0f) {
    tf = true;
  }else if(Y != 0f && X == 0f && Z == 0f){
    tf = true;
  }

  /**annallyJudge**/
  Vector3 anally0 = hit.collider.gameObject.transform.position;//クリックした座標を代入
  //クリックした座標の y 座標が+1f の位置を代入
  Vector3 anally = new Vector3 (anally0.x, anally0.y + 1.0f, anally0.z);
  //下方向に光線を発射しぶつかったゲームオブジェクトの情報を記憶。
  if(Physics.Raycast(anally,Vector3.down ,out hit10,3f)){
    //ぶつかったオブジェクトが味方の駒の時
```

```
            if(hit10.collider.gameObject.tag == "piece2"){
                tf = false;//飛び越え禁止
            }
        }

        return tf;
    }

    /*
    void OnGUI () {//テスト用テキスト表示メソッド
        GUI.TextField (new Rect (10, 10, 100, 200), t);
    }*/

}
```

- Knightb

```
public class Knightb: MonoBehaviour{
    RaycastHit hit, hit2, hit9, hit10;//オブジェクトの存在判定を行う。
    bool isSelect = false;//駒の選択状態判定用の変数である。
    Color orgColor,selColor = Color.yellow;//駒が選択された時の色を保管しておく変数である。
    //string t = "a";//テスト用のテキストである。

    void Start(){//初期条件
        orgColor = gameObject.renderer.material.color;//元々のオブジェクトの色に設定しておく
    }

    void Update(){//常に実行され続けている
/*プレイヤーのターンかどうかの判定をしている。
*Glpbal.flg とはグローバル変数(bool 型)であり
*'Global.flg == false'で黒のターンである。
*/
        if (Global.flg) {
            return;//Update()を終了させている。
        }
        if(Input.GetMouseButtonDown(0)){//マウスの左クリックが押された時
            Vector3 pos = Input.mousePosition;//押された時のマウスの位置を保管する。
            pos.z = 10.0f;//しかしマウスは奥行きが認識できないため z 座標を設定する。

            Ray ray = Camera.main.ScreenPointToRay(pos);//光線の作成

            if(Physics.Raycast(ray,out hit,100)){//当たったオブジェクトの判定
                //当たったオブジェクトがこのオブジェクトの場合
                if(hit.collider.gameObject.name == gameObject.name){
```

```
            isSelect = !isSelect;//f -> t//選択状態に移行
        }

        if(isSelect && hit.collider.gameObject.tag == "cell"){//マスがクリックされた時
            if(Control()){//移動可能か判定
                Vector3 newpos = hit.collider.gameObject.transform.position;クリック位置保管
                //敵駒判定用の光線作成
                Vector3 newpos2 = new Vector3(newpos.x ,newpos.y + 1.0f ,newpos.z);
                if(Physics.Raycast(newpos2,Vector3.down ,out hit2,3f)){
                    if(hit2.collider.gameObject.tag == "piece1"){
                        Destroy(hit2.collider.gameObject);//敵駒破壊
                    }
                }
                //クリックしたマスへ移動
                transform.position = new Vector3(newpos.x, newpos.y + 0.23f, newpos.z);
                //collision Time
                isSelect = false;//選択状態解除
                Flg fchange = new Flg();//白黒ターン入れ替えのフラグ変更
                Global.flg = fchange.FlgSetter(Global.flg);
            }
        }

        if(isSelect){//選択駒の色変え
            gameObject.renderer.material.color = selColor;
        }else{
            gameObject.renderer.material.color = orgColor;
        }
    }
  }

  public bool Control(){//Knightpiece control
    float xPosition = transform.position.x ;//now position
    float yPosition = transform.position.y - 0.23f;
    float zPosition = transform.position.z ;
//今現在のオブジェクトの位置から y 座標を調整して保管
    Vector3 np = new Vector3 (xPosition, yPosition + 0.3f, zPosition);
    bool a = false;//移動できる位置か判定
    for(float X = -4f; X <= 4f; X++ ){
      for(float Y = -8f; Y <= 8f; Y = Y + 2f ){
        for(float Z = -4f; Z <= 4f; Z++ ){
          Vector3 onCell = new Vector3(xPosition + X, yPosition + Y, zPosition + Z);//true position
          //クリックした位置を保管
          Vector3 hitposition = hit.collider.gameObject.transform.position;
```

```
            //x.0000~ -> xf
            if(hit.collider.gameObject.transform.position.x >= X -0.5f
                && hit.collider.gameObject.transform.position.x < X + 0.5f
                && hit.collider.gameObject.transform.position.y >= Y -0.5f
                && hit.collider.gameObject.transform.position.y < Y + 0.5f
                && hit.collider.gameObject.transform.position.z >= Z -0.5f
                && hit.collider.gameObject.transform.position.z < Z + 0.5f){//Ray hit position
               hitposition = new Vector3(X ,Y ,Z);//整数調整
            }

            //移動できる範囲に収まっているかの判定
            if(onCell.x == hitposition.x && onCell.z == hitposition.z && onCell.y == hitposition.y){
            //実際に移動できるマスか判定
               if (KnightJudge (X, Y, Z)){
                  a = true;//移動できるよう書き換え
                  if (X == 0f && Y == 0f && Z == 0f){//移動してない位置をクリックしていた場合
                     a = false;//移動できないよう書き換え
                  }
                  /*今の位置とクリックした位置の間のベクトルの大きさを保管する。
                  *しかし後に間にオブジェクトがあるか判定するためクリックした位置の駒と
                  *区別するため 1f 短くしてクリックマスにある駒に当たらないようにしている。
                  */
                  float dis = (hitposition - np).magnitude - 1.0f;//hitp <--> nowp
                  //全てのマスオブジェクトの情報を取得
                  GameObject[] cells = GameObject.FindGameObjectsWithTag("cell");


                  //今あるオブジェクトの位置とクリックした地点の距離を保管
                  Vector3 vecp = new Vector3 (hitposition.x - np.x, hitposition.y - np.y + 0.3f,
hitposition.z - np.z);

                  /*hits に現在のオブジェクト位置と移動先(クリック位置)の間の全てのオブジェクトの
                  *情報を格納していく。(格納される順番はランダム)
                  */
                  RaycastHit[] hits = Physics.RaycastAll (np, vecp, dis);
                  for (int i = 0; i < hits.Length; i++) {//何個間にあったか判定
                     //当たったオブジェクトのタグを保管する
                     string hitTag = hits [(int)i].collider.gameObject.tag;
                     //当たったオブジェクトが駒だった場合移動できないよう書き換え
                     if (hitTag == "piece2" || hitTag == "piece1") {
                        a = false;
                     }
                  }
                  break;
               }
```

```
                }
            }
        }
    }
    return a;
}

//ナイトからの相対的な位置判定
    public bool KnightJudge(float X, float Y, float Z){

        bool tf = false;

        if (Y == 0f && X == -1f && Z == -2f||
            Y == 0f && X == 1f && Z == -2f||
            Y == 0f && X == -2f && Z == -1f||
            Y == 0f && X == 2f && Z == -1f||
            Y == 0f && X == -2f && Z == 1f||
            Y == 0f && X == 2f && Z == 1f||
            Y == 0f && X == -1f && Z == 2f||
            Y == 0f && X == 1f && Z == 2f
            ){
            tf = true;
        } else if(Y == 2f && X == 0f && Z == -2f||
                    Y == 2f && X == -2f && Z == 0f||
                    Y == 2f && X == 2f && Z == 0f||
                    Y == 2f && X == 0f && Z == 2f){
            tf = true;
        }else if(Y == 4f && X == 0f && Z == -1f||
                   Y == 4f && X == -1f && Z == 0f||
                   Y == 4f && X == 1f && Z == 0f||
                   Y == 4f && X == 0f && Z == 1f){
            tf = true;
        }   else if(Y == -2f && X == 0f && Z == -2f||
                      Y == -2f && X == -2f && Z == 0f||
                      Y == -2f && X == 2f && Z == 0f||
                      Y == -2f && X == 0f && Z == 2f){
            tf = true;
        }else if(Y == -4f && X == 0f && Z == -1f||
                   Y == -4f && X == -1f && Z == 0f||
                   Y == -4f && X == 1f && Z == 0f||
                   Y == -4f && X == 0f && Z == 1f){
            tf = true;
        }

        /**annallyJudge**/
```

```
        Vector3 anally0 = hit.collider.gameObject.transform.position;//クリックした座標を代入
        //クリックした座標の y 座標が+1f の位置を代入
        Vector3 anally = new Vector3 (anally0.x, anally0.y + 1.0f, anally0.z);
        //下方向に光線を発射しぶつかったゲームオブジェクトの情報を記憶。
        if(Physics.Raycast(anally,Vector3.down ,out hit10,3f)){
            //ぶつかったオブジェクトが味方の駒の時
            if(hit10.collider.gameObject.tag == "piece2"){
                tf = false;//飛び越え禁止
            }
        }

        return tf;
    }

    /*
    void OnGUI () {//テスト用テキスト表示メソッド
        GUI.TextField (new Rect (10, 10, 100, 200), t);
    }*/

}
```

- Unicornb

```
public class Unicornb: MonoBehaviour{
    RaycastHit hit, hit2, hit9, hit10;//オブジェクトの存在判定を行う。
    bool isSelect = false;//駒の選択状態判定用の変数である。
    Color orgColor,selColor = Color.yellow;//駒が選択された時の色を保管しておく変数である。
    //string t = "a";//テスト用のテキストである。

    void Start(){//初期条件
        orgColor = gameObject.renderer.material.color;//元々のオブジェクトの色に設定しておく
    }

    void Update(){//常に実行され続けている
/*プレイヤーのターンかどうかの判定をしている。
*Glpbal.flg とはグローバル変数(bool 型)であり
*'Global.flg == false'で黒のターンである。
*/
        if (Global.flg) {
            return;//Update()を終了させている。
        }
        if(Input.GetMouseButtonDown(0)){//マウスの左クリックが押された時
            Vector3 pos = Input.mousePosition;//押された時のマウスの位置を保管する。
            pos.z = 10.0f;//しかしマウスは奥行きが認識できないため z 座標を設定する。
```

```
        Ray ray = Camera.main.ScreenPointToRay(pos);//光線の作成

        if(Physics.Raycast(ray,out hit,100)){//当たったオブジェクトの判定
          //当たったオブジェクトがこのオブジェクトの場合
          if(hit.collider.gameObject.name == gameObject.name){
            isSelect = !isSelect;//f -> t//選択状態に移行
          }

          if(isSelect && hit.collider.gameObject.tag == "cell"){//マスがクリックされた時
            if(Control()){//移動可能か判定
              Vector3 newpos = hit.collider.gameObject.transform.position;クリック位置保管
              //敵駒判定用の光線作成
              Vector3 newpos2 = new Vector3(newpos.x ,newpos.y + 1.0f ,newpos.z);
              if(Physics.Raycast(newpos2,Vector3.down ,out hit2,3f)){
                if(hit2.collider.gameObject.tag == "piece1"){
                  Destroy(hit2.collider.gameObject);//敵駒破壊
                }
              }
              //クリックしたマスへ移動
              transform.position = new Vector3(newpos.x, newpos.y + 0.23f, newpos.z);
              //collision Time
              isSelect = false;//選択状態解除
              Flg fchange = new Flg();//白黒ターン入れ替えのフラグ変更
              Global.flg = fchange.FlgSetter(Global.flg);
            }
          }

          if(isSelect){//選択駒の色変え
            gameObject.renderer.material.color = selColor;
          }else{
            gameObject.renderer.material.color = orgColor;
          }
        }
      }
    }


    public bool Control(){//Unicornpiece control
      float xPosition = transform.position.x ;//now position
      float yPosition = transform.position.y - 0.23f;
      float zPosition = transform.position.z ;
//今現在のオブジェクトの位置から y 座標を調整して保管
      Vector3 np = new Vector3 (xPosition, yPosition + 0.3f, zPosition);
      bool a = false;//移動できる位置か判定
```

```
for(float X = -4f; X <= 4f; X++ ){
  for(float Y = -8f; Y <= 8f; Y = Y + 2f ){
    for(float Z = -4f; Z <= 4f; Z++ ){
      Vector3 onCell = new Vector3(xPosition + X, yPosition + Y, zPosition + Z);//true position
      //クリックした位置を保管
      Vector3 hitposition = hit.collider.gameObject.transform.position;

      //x.0000~ -> xf
      if(hit.collider.gameObject.transform.position.x >= X -0.5f
        && hit.collider.gameObject.transform.position.x < X + 0.5f
        && hit.collider.gameObject.transform.position.y >= Y -0.5f
        && hit.collider.gameObject.transform.position.y < Y + 0.5f
        && hit.collider.gameObject.transform.position.z >= Z -0.5f
        && hit.collider.gameObject.transform.position.z < Z + 0.5f){//Ray hit position
        hitposition = new Vector3(X ,Y ,Z);//整数調整
      }

      //移動できる範囲に収まっているかの判定
      if(onCell.x == hitposition.x && onCell.z == hitposition.z && onCell.y == hitposition.y){
      //実際に移動できるマスか判定
        if (UnicornJudge (X, Y, Z)){
          a = true;//移動できるよう書き換え
          if (X == 0f && Y == 0f && Z == 0f){//移動してない位置をクリックしていた場合
            a = false;//移動できないよう書き換え
          }
          /*今の位置とクリックした位置の間のベクトルの大きさを保管する。
          *しかし後に間にオブジェクトがあるか判定するためクリックした位置の駒と
          *区別するため 1f 短くしてクリックマスにある駒に当たらないようにしている。
          */
          float dis = (hitposition - np).magnitude - 1.0f;//hitp <--> nowp
          //全てのマスオブジェクトの情報を取得
          GameObject[] cells = GameObject.FindGameObjectsWithTag("cell");


          //今あるオブジェクトの位置とクリックした地点の距離を保管
          Vector3 vecp = new Vector3 (hitposition.x - np.x, hitposition.y - np.y + 0.3f,
hitposition.z - np.z);

          /*hits に現在のオブジェクト位置と移動先(クリック位置)の間の全てのオブジェクトの
          *情報を格納していく。(格納される順番はランダム)
          */
          RaycastHit[] hits = Physics.RaycastAll (np, vecp, dis);
          for (int i = 0; i < hits.Length; i++) {//何個間にあったか判定
            //当たったオブジェクトのタグを保管する
            string hitTag = hits [(int)i].collider.gameObject.tag;
```

```
                    //当たったオブジェクトが駒だった場合移動できないよう書き換え
                    if (hitTag == "piece2" || hitTag == "piece1") {
                        a = false;
                    }
                }
                break;
            }
        }
    }
}
return a;
}

//ユニコーンの相対的な移動可能判定
public bool UnicornJudge(float X, float Y, float Z){

    bool tf = false;

    if (X == Z && X == Y/2 ||
        -X == Z && -X == Y/2 ||
        X == -Z && X == Y/2 ||
        -X == -Z && -X == Y/2 ||
        X == Z && X == -Y/2 ||
        -X == Z && -X == -Y/2 ||
        X == -Z && X == -Y/2 ||
        -X == -Z && -X == -Y/2) {
        tf = true;
        if(Y == 0f){
            tf = false;
        }

        /**annallyJudge**/
    Vector3 anally0 = hit.collider.gameObject.transform.position;//クリックした座標を代入
    //クリックした座標の y 座標が+1f の位置を代入
    Vector3 anally = new Vector3 (anally0.x, anally0.y + 1.0f, anally0.z);
    //下方向に光線を発射しぶつかったゲームオブジェクトの情報を記憶。
    if(Physics.Raycast(anally,Vector3.down ,out hit10,3f)){
        //ぶつかったオブジェクトが味方の駒の時
        if(hit10.collider.gameObject.tag == "piece2"){
            tf = false;//飛び越え禁止
        }
    }

    return tf;
```

```
    }

    /*
    void OnGUI () {//テスト用テキスト表示メソッド
        GUI.TextField (new Rect (10, 10, 100, 200), t);
    }*/

}
```

- Bishopb

```
public class Bishopb : MonoBehaviour{
    RaycastHit hit, hit2, hit9, hit10;//オブジェクトの存在判定を行う。
    bool isSelect = false;//駒の選択状態判定用の変数である。
    Color orgColor,selColor = Color.yellow;//駒が選択された時の色を保管しておく変数である。
    //string t = "a";//テスト用のテキストである。

    void Start(){//初期条件
        orgColor = gameObject.renderer.material.color;//元々のオブジェクトの色に設定しておく
    }

    void Update(){//常に実行され続けている
/*プレイヤーのターンかどうかの判定をしている。
*Glpbal.flg とはグローバル変数(bool 型)であり
*'Global.flg == false'で黒のターンである。
*/
        if (Global.flg) {
            return;//Update()を終了させている。
        }
        if(Input.GetMouseButtonDown(0)){//マウスの左クリックが押された時
            Vector3 pos = Input.mousePosition;//押された時のマウスの位置を保管する。
            pos.z = 10.0f;//しかしマウスは奥行きが認識できないため z 座標を設定する。

            Ray ray = Camera.main.ScreenPointToRay(pos);//光線の作成

            if(Physics.Raycast(ray,out hit,100)){//当たったオブジェクトの判定
                //当たったオブジェクトがこのオブジェクトの場合
                if(hit.collider.gameObject.name == gameObject.name){
                    isSelect = !isSelect;//f -> t//選択状態に移行
                }

                if(isSelect && hit.collider.gameObject.tag == "cell"){//マスがクリックされた時
                    if(Control()){//移動可能か判定
                        Vector3 newpos = hit.collider.gameObject.transform.position;クリック位置保管
```

```
                //敵駒判定用の光線作成
                Vector3 newpos2 = new Vector3(newpos.x ,newpos.y + 1.0f ,newpos.z);
                if(Physics.Raycast(newpos2,Vector3.down ,out hit2,3f)){
                    if(hit2.collider.gameObject.tag == "piece1"){
                        Destroy(hit2.collider.gameObject);//敵駒破壊
                    }
                }
                //クリックしたマスへ移動
                transform.position = new Vector3(newpos.x, newpos.y + 0.23f, newpos.z);
                //collision Time
                isSelect = false;//選択状態解除
                Flg fchange = new Flg();//白黒ターン入れ替えのフラグ変更
                Global.flg = fchange.FlgSetter(Global.flg);
            }
        }

        if(isSelect){//選択駒の色変え
            gameObject.renderer.material.color = selColor;
        }else{
            gameObject.renderer.material.color = orgColor;
        }
    }
  }


    public bool Control(){//Bishoppiece control
        float xPosition = transform.position.x ;//now position
        float yPosition = transform.position.y - 0.23f;
        float zPosition = transform.position.z ;
//今現在のオブジェクトの位置から y 座標を調整して保管
        Vector3 np = new Vector3 (xPosition, yPosition + 0.3f, zPosition);
        bool a = false;//移動できる位置か判定
        for(float X = -4f; X <= 4f; X++ ){
            for(float Y = -8f; Y <= 8f; Y = Y + 2f ){
                for(float Z = -4f; Z <= 4f; Z++ ){
                    Vector3 onCell = new Vector3(xPosition + X, yPosition + Y, zPosition + Z);//true position
                    //クリックした位置を保管
                    Vector3 hitposition = hit.collider.gameObject.transform.position;

                    //x.0000~ -> xf
                    if(hit.collider.gameObject.transform.position.x >= X -0.5f
                        && hit.collider.gameObject.transform.position.x < X + 0.5f
                        && hit.collider.gameObject.transform.position.y >= Y -0.5f
                        && hit.collider.gameObject.transform.position.y < Y + 0.5f
```

```
                && hit.collider.gameObject.transform.position.z >= Z -0.5f
                && hit.collider.gameObject.transform.position.z < Z + 0.5f){//Ray hit position
                hitposition = new Vector3(X ,Y ,Z);//整数調整
            }

            //移動できる範囲に収まっているかの判定
            if(onCell.x == hitposition.x && onCell.z == hitposition.z && onCell.y == hitposition.y){
            //実際に移動できるマスか判定
                if (BishopJudge (X, Y, Z)){
                    a = true;//移動できるよう書き換え
                    if (X == 0f && Y == 0f && Z == 0f){//移動してない位置をクリックしていた場合
                        a = false;//移動できないよう書き換え
                    }
                    /*今の位置とクリックした位置の間のベクトルの大きさを保管する。
                    *しかし後に間にオブジェクトがあるか判定するためクリックした位置の駒と
                    *区別するため 1f 短くしてクリックマスにある駒に当たらないようにしている。
                    */
                    float dis = (hitposition - np).magnitude - 1.0f;//hitp <--> nowp
                    //全てのマスオブジェクトの情報を取得
                    GameObject[] cells = GameObject.FindGameObjectsWithTag("cell");


                    //今あるオブジェクトの位置とクリックした地点の距離を保管
                    Vector3 vecp = new Vector3 (hitposition.x - np.x, hitposition.y - np.y + 0.3f,
hitposition.z - np.z);

                    /*hits に現在のオブジェクト位置と移動先(クリック位置)の間の全てのオブジェクトの
                    *情報を格納していく。(格納される順番はランダム)
                    */
                    RaycastHit[] hits = Physics.RaycastAll (np, vecp, dis);
                    for (int i = 0; i < hits.Length; i++) {//何個間にあったか判定
                        //当たったオブジェクトのタグを保管する
                        string hitTag = hits [(int)i].collider.gameObject.tag;
                        //当たったオブジェクトが駒だった場合移動できないよう書き換え
                        if (hitTag == "piece2" || hitTag == "piece1") {
                            a = false;
                        }
                    }
                    break;
                }
            }
        }
    }
    return a;
```

```
    }

    //ビショップからの相対的な位置判定
    public bool BishopJudge(float X, float Y, float Z){

        bool tf = false;

        if (Y == 0f && X == Z | |
        Y == 0f && X == -Z | |
        Y == 0f && -X == Z | |
        Y == 0f && -X == -Z) {
            tf = true;
        }else if(X == 0 && Z == Y / 2 | |
        X == 0 && -Z == Y / 2 | |
        Z == 0 && X == Y / 2 | |
        Z == 0 && -X == Y / 2){
            tf = true;
        }
            /**annallyJudge**/
        Vector3 anally0 = hit.collider.gameObject.transform.position;//クリックした座標を代入
        //クリックした座標の y 座標が+1f の位置を代入
        Vector3 anally = new Vector3 (anally0.x, anally0.y + 1.0f, anally0.z);
        //下方向に光線を発射しぶつかったゲームオブジェクトの情報を記憶。
        if(Physics.Raycast(anally,Vector3.down ,out hit10,3f)){
            //ぶつかったオブジェクトが味方の駒の時
            if(hit10.collider.gameObject.tag == "piece2"){
                tf = false;//飛び越え禁止
            }
        }

        return tf;
    }

    /*
    void OnGUI () {//テスト用テキスト表示メソッド
        GUI.TextField (new Rect (10, 10, 100, 200), t);
    }*/

}
```

- Rookb

```
public class Rookb : MonoBehaviour{
    RaycastHit hit, hit2, hit9, hit10;//オブジェクトの存在判定を行う。
```

```
    bool isSelect = false;//駒の選択状態判定用の変数である。
    Color orgColor,selColor = Color.yellow;//駒が選択された時の色を保管しておく変数である。
    //string t = "a";//テスト用のテキストである。

    void Start(){//初期条件
        orgColor = gameObject.renderer.material.color;//元々のオブジェクトの色に設定しておく
    }

    void Update(){//常に実行され続けている
/*プレイヤーのターンかどうかの判定をしている。
*Glpbal.flg とはグローバル変数(bool 型)であり
*'Global.flg == false'で黒のターンである。
*/
        if (Global.flg) {
            return;//Update()を終了させている。
        }
        if(Input.GetMouseButtonDown(0)){//マウスの左クリックが押された時
            Vector3 pos = Input.mousePosition;//押された時のマウスの位置を保管する。
            pos.z = 10.0f;//しかしマウスは奥行きが認識できないため z 座標を設定する。

            Ray ray = Camera.main.ScreenPointToRay(pos);//光線の作成

            if(Physics.Raycast(ray,out hit,100)){//当たったオブジェクトの判定
                //当たったオブジェクトがこのオブジェクトの場合
                if(hit.collider.gameObject.name == gameObject.name){
                    isSelect = !isSelect;//f -> t//選択状態に移行
                }

                if(isSelect && hit.collider.gameObject.tag == "cell"){//マスがクリックされた時
                    if(Control()){//移動可能か判定
                        Vector3 newpos = hit.collider.gameObject.transform.position;クリック位置保管
                        //敵駒判定用の光線作成
                        Vector3 newpos2 = new Vector3(newpos.x ,newpos.y + 1.0f ,newpos.z);
                        if(Physics.Raycast(newpos2,Vector3.down ,out hit2,3f)){
                            if(hit2.collider.gameObject.tag == "piece1"){
                                Destroy(hit2.collider.gameObject);//敵駒破壊
                            }
                        }
                        //クリックしたマスへ移動
                        transform.position = new Vector3(newpos.x, newpos.y + 0.23f, newpos.z);
                        //collision Time
                        isSelect = false;//選択状態解除
                        Flg fchange = new Flg();//白黒ターン入れ替えのフラグ変更
                        Global.flg = fchange.FlgSetter(Global.flg);
                    }
```

```
            }

            if(isSelect){//選択駒の色変え
                gameObject.renderer.material.color = selColor;
            }else{
                gameObject.renderer.material.color = orgColor;
            }
        }
    }
}


    public bool Control(){//Rookpiece control
        float xPosition = transform.position.x ;//now position
        float yPosition = transform.position.y - 0.23f;
        float zPosition = transform.position.z ;
//今現在のオブジェクトの位置から y 座標を調整して保管
        Vector3 np = new Vector3 (xPosition, yPosition + 0.3f, zPosition);
        bool a = false;//移動できる位置か判定
        for(float X = -4f; X <= 4f; X++ ){
            for(float Y = -8f; Y <= 8f; Y = Y + 2f ){
                for(float Z = -4f; Z <= 4f; Z++ ){
                    Vector3 onCell = new Vector3(xPosition + X, yPosition + Y, zPosition + Z);//true position
                    //クリックした位置を保管
                    Vector3 hitposition = hit.collider.gameObject.transform.position;

                    //x.0000~ -> xf
                    if(hit.collider.gameObject.transform.position.x >= X -0.5f
                        && hit.collider.gameObject.transform.position.x < X + 0.5f
                        && hit.collider.gameObject.transform.position.y >= Y -0.5f
                        && hit.collider.gameObject.transform.position.y < Y + 0.5f
                        && hit.collider.gameObject.transform.position.z >= Z -0.5f
                        && hit.collider.gameObject.transform.position.z < Z + 0.5f){//Ray hit position
                       hitposition = new Vector3(X ,Y ,Z);//整数調整
                    }

                    //移動できる範囲に収まっているかの判定
                    if(onCell.x == hitposition.x && onCell.z == hitposition.z && onCell.y == hitposition.y){
                    //実際に移動できるマスか判定
                       if (RookJudge (X, Y, Z)){
                          a = true;//移動できるよう書き換え
                          if (X == 0f && Y == 0f && Z == 0f){//移動してない位置をクリックしていた場合
                             a = false;//移動できないよう書き換え
                          }
                          /*今の位置とクリックした位置の間のベクトルの大きさを保管する。
```

```
                    *しかし後に間にオブジェクトがあるか判定するためクリックした位置の駒と
                    *区別するため 1f 短くしてクリックマスにある駒に当たらないようにしている。
                    */
                    float dis = (hitposition - np).magnitude - 1.0f;//hitp <--> nowp
                    //全てのマスオブジェクトの情報を取得
                    GameObject[] cells = GameObject.FindGameObjectsWithTag("cell");


                    //今あるオブジェクトの位置とクリックした地点の距離を保管
                    Vector3 vecp = new Vector3 (hitposition.x - np.x, hitposition.y - np.y + 0.3f,
hitposition.z - np.z);

                    /*hits に現在のオブジェクト位置と移動先(クリック位置)の間の全てのオブジェクトの
                    *情報を格納していく。(格納される順番はランダム)
                    */
                    RaycastHit[] hits = Physics.RaycastAll (np, vecp, dis);
                    for (int i = 0; i < hits.Length; i++) {//何個間にあったか判定
                        //当たったオブジェクトのタグを保管する
                        string hitTag = hits [(int)i].collider.gameObject.tag;
                        //当たったオブジェクトが駒だった場合移動できないよう書き換え
                        if (hitTag == "piece2" || hitTag == "piece1") {
                            a = false;
                        }
                    }
                    break;
                }
            }
        }
    }
    return a;
}

//ルークからの相対的な位置判定
    public bool RookJudge(float X, float Y, float Z){

        bool tf = false;

        if (Y == 0f && X == 0f ||
            Y == 0f && Z == 0f) {
            tf = true;
        }else if(Y != 0f && X == 0f && Z == 0f){
            tf = true;
        }
```

```
        /**annallyJudge**/
      Vector3 anally0 = hit.collider.gameObject.transform.position;//クリックした座標を代入
      //クリックした座標の y 座標が+1f の位置を代入
      Vector3 anally = new Vector3 (anally0.x, anally0.y + 1.0f, anally0.z);
      //下方向に光線を発射しぶつかったゲームオブジェクトの情報を記憶。
      if(Physics.Raycast(anally,Vector3.down ,out hit10,3f)){
        //ぶつかったオブジェクトが味方の駒の時
        if(hit10.collider.gameObject.tag == "piece2"){
          tf = false;//飛び越え禁止
        }
      }

      return tf;
    }

    /*
    void OnGUI () {//テスト用テキスト表示メソッド
      GUI.TextField (new Rect (10, 10, 100, 200), t);
    }*/

}
```

- Pawnb

```
public class Pawnb : MonoBehaviour{
    public class Rook : MonoBehaviour{
    RaycastHit hit, hit2, hit9, hit10;//オブジェクトの存在判定を行う。
    bool isSelect = false;//駒の選択状態判定用の変数である。
    Color orgColor,selColor = Color.yellow;//駒が選択された時の色を保管しておく変数である。
    //string t = "a";//テスト用のテキストである。

    void Start(){//初期条件
      orgColor = gameObject.renderer.material.color;//元々のオブジェクトの色に設定しておく
    }

    void Update(){//常に実行され続けている
/*プレイヤーのターンかどうかの判定をしている。
*Glpbal.flg とはグローバル変数(bool 型)であり
*'Global.flg == false'で黒のターンである。
*/
      if (Global.flg) {
        return;//Update()を終了させている。
      }
      if(Input.GetMouseButtonDown(0)){//マウスの左クリックが押された時
```

```
        Vector3 pos = Input.mousePosition;//押された時のマウスの位置を保管する。
        pos.z = 10.0f;//しかしマウスは奥行きが認識できないため z 座標を設定する。

        Ray ray = Camera.main.ScreenPointToRay(pos);//光線の作成

        if(Physics.Raycast(ray,out hit,100)){//当たったオブジェクトの判定
          //当たったオブジェクトがこのオブジェクトの場合
          if(hit.collider.gameObject.name == gameObject.name){
            isSelect = !isSelect;//f -> t//選択状態に移行
          }

          if(isSelect && hit.collider.gameObject.tag == "cell"){//マスがクリックされた時
            if(Control()){//移動可能か判定
              Vector3 newpos = hit.collider.gameObject.transform.position;クリック位置保管
              //敵駒判定用の光線作成
              Vector3 newpos2 = new Vector3(newpos.x ,newpos.y + 1.0f ,newpos.z);
              if(Physics.Raycast(newpos2,Vector3.down ,out hit2,3f)){
                if(hit2.collider.gameObject.tag == "piece1"){
                  Destroy(hit2.collider.gameObject);//敵駒破壊
                }
              }
              //クリックしたマスへ移動
              transform.position = new Vector3(newpos.x, newpos.y + 0.23f, newpos.z);
              //collision Time
              isSelect = false;//選択状態解除
              Flg fchange = new Flg();//白黒ターン入れ替えのフラグ変更
              Global.flg = fchange.FlgSetter(Global.flg);
            }
          }

          if(isSelect){//選択駒の色変え
            gameObject.renderer.material.color = selColor;
          }else{
            gameObject.renderer.material.color = orgColor;
          }
        }
      }
    }

    public bool Control(){//Pawnpiece control
      float xPosition = transform.position.x ;//now position
      float yPosition = transform.position.y - 0.23f;
      float zPosition = transform.position.z ;
//今現在のオブジェクトの位置から y 座標を調整して保管
      Vector3 np = new Vector3 (xPosition, yPosition + 0.3f, zPosition);
```

```
bool a = false;//移動できる位置か判定
for(float X = -4f; X <= 4f; X++ ){
   for(float Y = -8f; Y <= 8f; Y = Y + 2f ){
      for(float Z = -4f; Z <= 4f; Z++ ){
         Vector3 onCell = new Vector3(xPosition + X, yPosition + Y, zPosition + Z);//true position
         //クリックした位置を保管
         Vector3 hitposition = hit.collider.gameObject.transform.position;

         //x.0000~ -> xf
         if(hit.collider.gameObject.transform.position.x >= X -0.5f
            && hit.collider.gameObject.transform.position.x < X + 0.5f
            && hit.collider.gameObject.transform.position.y >= Y -0.5f
            && hit.collider.gameObject.transform.position.y < Y + 0.5f
            && hit.collider.gameObject.transform.position.z >= Z -0.5f
            && hit.collider.gameObject.transform.position.z < Z + 0.5f){//Ray hit position
           hitposition = new Vector3(X ,Y ,Z);//整数調整
         }

         //移動できる範囲に収まっているかの判定
         if(onCell.x == hitposition.x && onCell.z == hitposition.z && onCell.y == hitposition.y){
         //実際に移動できるマスか判定
           if (PawnJudge (X, Y, Z)){
             a = true;//移動できるよう書き換え
             if (X == 0f && Y == 0f && Z == 0f){//移動してない位置をクリックしていた場合
               a = false;//移動できないよう書き換え
             }
             /*今の位置とクリックした位置の間のベクトルの大きさを保管する。
             *しかし後に間にオブジェクトがあるか判定するためクリックした位置の駒と
             *区別するため 1f 短くしてクリックマスにある駒に当たらないようにしている。
             */
             float dis = (hitposition - np).magnitude - 1.0f;//hitp <--> nowp
             //全てのマスオブジェクトの情報を取得
             GameObject[] cells = GameObject.FindGameObjectsWithTag("cell");


             //今あるオブジェクトの位置とクリックした地点の距離を保管
             Vector3 vecp = new Vector3 (hitposition.x - np.x, hitposition.y - np.y + 0.3f,
hitposition.z - np.z);

             /*hits に現在のオブジェクト位置と移動先(クリック位置)の間の全てのオブジェクトの
             *情報を格納していく。(格納される順番はランダム)
             */
             RaycastHit[] hits = Physics.RaycastAll (np, vecp, dis);
             for (int i = 0; i < hits.Length; i++) {//何個間にあったか判定
               //当たったオブジェクトのタグを保管する
```

```
                    string hitTag = hits [(int)i].collider.gameObject.tag;
                    //当たったオブジェクトが駒だった場合移動できないよう書き換え
                    if (hitTag == "piece2" || hitTag == "piece1") {
                        a = false;
                    }
                }
                break;
            }
        }
    }
}
return a;
}

//ポーンの相対的な位置判定
public bool PawnbJudge(float X, float Y, float Z){

    bool tf = false;

    if (Y == 0f && X == 0f && Z == 1f||
        Y == -2f && X == 0f && Z == 0f) {
        tf = true;
    }

        /**annallyJudge**/
    Vector3 anally0 = hit.collider.gameObject.transform.position;//クリックした座標を代入
    //クリックした座標の y 座標が+1f の位置を代入
    Vector3 anally = new Vector3 (anally0.x, anally0.y + 1.0f, anally0.z);
    //下方向に光線を発射しぶつかったゲームオブジェクトの情報を記憶。
    if(Physics.Raycast(anally,Vector3.down ,out hit10,3f)){
        //ぶつかったオブジェクトが味方の駒の時
        if(hit10.collider.gameObject.tag == "piece2"){
            tf = false;//飛び越え禁止
        }
    }

    return tf;
}

/*
void OnGUI () {//テスト用テキスト表示メソッド
    GUI.TextField (new Rect (10, 10, 100, 200), t);
}*/
```

```
}
```

- Game //ゲーム終了時にテキストを表示するクラス

```
public class Game : MonoBehaviour {
    private GUIText gtext; //GUIText 型フィールド
    public GameObject GameText;//ゲームオブジェクト型フィールド
    public void game(){
        /*GUIText のコンポーネントを取得し GameText という名のテキスト表示オブジェクトの
        *書き換えようインスタンス作成
        */
        gtext = GameText.GetComponent<GUIText> ();
/* Global.tex というグローバル変数の中に入ったテキストを書き換えて表示させる。
*(初期状態では空欄になっている。)
*/
        gtext.text = Global.tex;
    }
}
```

- Global //グローバル変数を扱うクラス

```
public class Global : MonoBehaviour {
    public static bool flg = true;//ターン判定用グローバル変数
    public static string tex = "";//ゲーム終了時のテキスト表示曜グローバル変数
}
```

- Flg //白駒、黒駒のターンの判定クラス

```
public class Flg : MonoBehaviour {
    public bool FlgSetter(bool flg){//各駒終了時に必ず呼び出される
        flg = !flg;//ターン用のフラグ入れ替え
        return flg;
    }
}
```

- CameraMove //視点用クラス

```
public class CameraMove : MonoBehaviour {
    private GameObject m_char1;
    // Use this for initialization
    void Start () {
```

```
        m_char1 = GameObject.Find("Main Camera");   // char1 の GameObject を呼び出し.
    }

    // Update is called once per frame
    void Update () {
        Vector3 v = m_char1.transform.localPosition;

        if (Input.GetKey(KeyCode.W)) {     // W キーで前進.
            v.y += 0.05f;
        }
        if (Input.GetKey(KeyCode.S)) {     // S キーで後退.
            v.y -= 0.05f;
        }
        if (Input.GetKey(KeyCode.A)) {   // A キーで左移動.
            v.x += 0.05f;
        }
        if (Input.GetKey(KeyCode.D)) {   // D キーで右移動.
            v.x -= 0.05f;
        }
        m_char1.transform.localPosition = v;
        if(Input.GetKey(KeyCode.UpArrow) //上矢印キーで上回転
            ||Input.GetKey(KeyCode.DownArrow) //下矢印キーで下回転
            ||Input.GetKey(KeyCode.RightArrow) //右矢印キーで右回転
            ||Input.GetKey(KeyCode.LeftArrow)){ //左矢印キーで左回転
            this.transform.Rotate ( 0, ( Input.GetAxis ( "Horizontal" ) * 1 ), 0 );
            this.transform.Rotate ( ( Input.GetAxis ( "Vertical" ) * 1 ), 0, 0 );
        }
    }
}
```