**プログラミングコンテスト2020**

**ガイスターゲーム**

**内容**

[**１．ゲームの流れ** 2](#_Toc5642567)

[**２．オバケの種類** 2](#_Toc5642568)

[**３．オバケの配置** 3](#_Toc5642569)

[**４．プレイヤーの行動** 3](#_Toc5642570)

[**５．画面の説明** 4](#_Toc5642571)

[**６．終了条件** 4](#_Toc5642572)

[**７．AI作成時に利用するクラス・関数・変数など** 5](#_Toc5642573)

[**８．備考** 7](#_Toc5642574)

# **１．ゲームの流れ**

1. **初期化**
2. **P1のターン**
3. **P2のターン**
4. **②，③を繰り返す**
5. **勝利条件を満たすか，P1,P2合わせて200ターン経過した時点で終了**

# **２．オバケの種類**

**両プレイヤーは以下のような2種類のオバケそれぞれ3体ずつを持つ**

**オバケのサイズはすべて1×1マス分**

**ただし,相手のお化けの種類は分からない**

****

* **青のオバケ  
  ID：0~2**

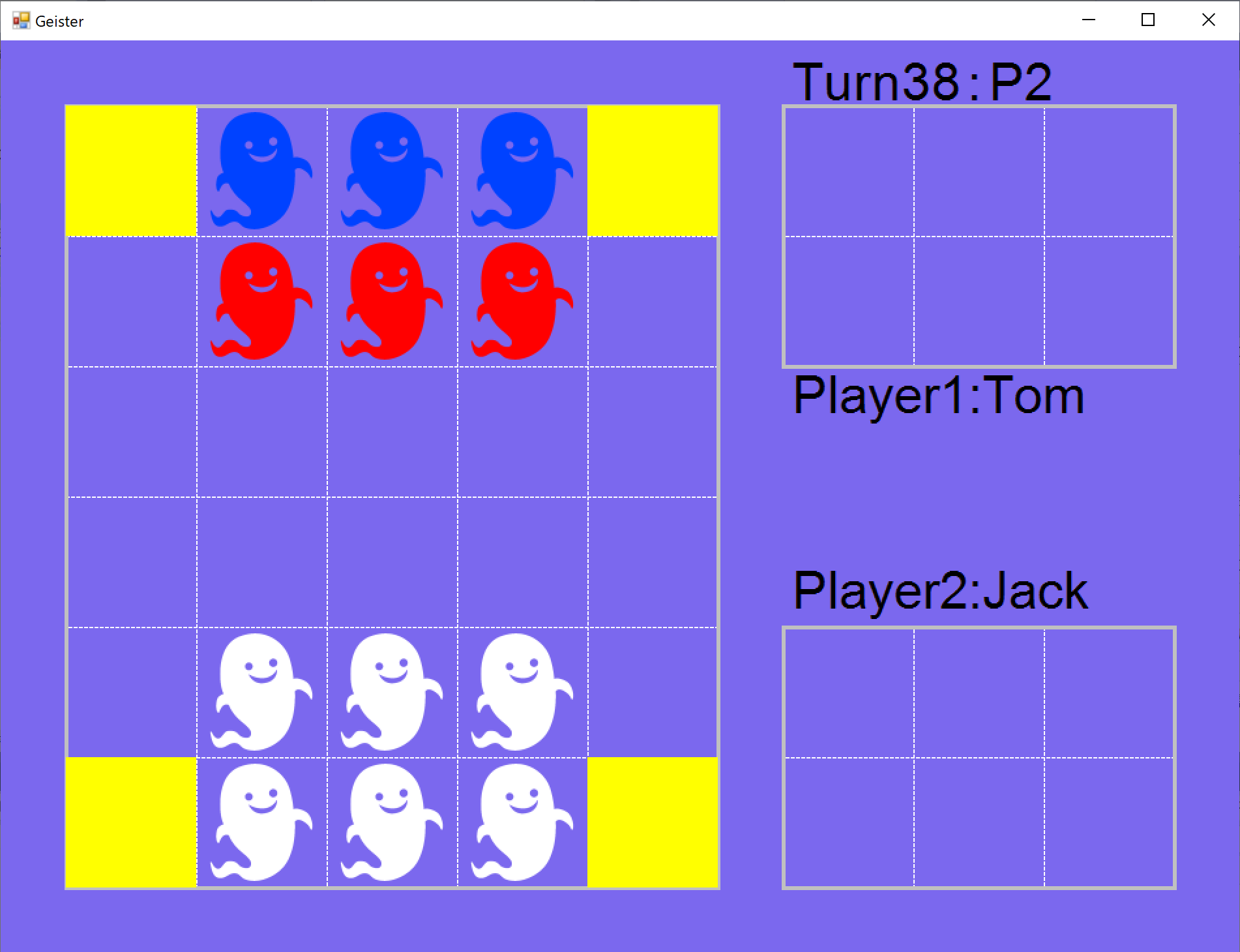
****

* **赤のオバケ  
  ID：3~5**

# **３．オバケの配置**

**エリアは5×6マス，座標は(0,0)が左上，(4,5)が右下となる**

**ゲーム開始時に両プレイヤーのオバケを以下の指定マス内に自由に配置**

**(1 <= X <= 3, 0 <= Y <= 1)**

プレイエリア

指定マス

# **４．プレイヤーの行動**

**プレイヤーは1ターンにつき1回，味方のオバケを1つだけ移動させることができる**

* **移動のルール：上下左右のいずれかに移動**
  + **敵オバケと重なった場合，そのオバケを取れる**

**※画面外，および味方のオバケがいる場所への移動および攻撃は不可能  
※不可能な行動を選択した場合や各手に250ミリ秒以上かけた場合パスされる**

# **５．画面の説明**

③→

③→

1. →
2. **現在のターン数と行動プレイヤー**
3. **プレイヤー名とそのプレイヤーが取った敵のオバケ**
4. **脱出マス**

# **６．終了条件**

**(1) 青のオバケをすべて取る  
敵の青のオバケをすべて取った場合，自身が勝利する**

**(2) 赤のオバケをすべて取る**

**敵の赤のオバケをすべて取った場合，自身が敗北する**

**(3) 青のオバケが脱出**

**自身の青のオバケのいずれかが脱出マスに到着してからとられずに**

**1ターン経過した場合，自身が勝利する**

**(4) 合計200ターン経過後  
P1,P2合わせて200ターン経過した時点でゲームを終了  
(盤面の自身の青のオバケの数)-(盤面の自身の赤のオバケの数)が多い方が勝利**

**上記が同数だった場合，盤面の自身の青のオバケの，脱出マスからの距離の合計が小さい方が勝利**

# **７．AI作成時に利用するクラス・関数・変数など**

**クラス**

* **abstract class AbstPlayer{  
   abstract Dicision Dicision()  
  }**

**AIを設定するための抽象クラス  
このクラスを継承したクラスを用意し，Dicision()関数をオーバーライドしてDicisionクラスのインスタンスを返すAIを実装する**

* **class Dicision{  
   int GI  
   Action\_Label AL  
   Direction DR  
  }  
  決定した行動を格納するクラス  
  このクラスのインスタンスを作成し，各変数に選択した行動の値を設定する**
  + **int GI：行動させるオバケのID (0~2：青のオバケ，3~5：赤のオバケ)**
  + **Action\_Label AL：行動の種類 (移動)**
  + **Direction DR：移動を行う方向**
* **class GhostState{  
   int GetId()  
   Point GetPos()  
   bool GetDoA()  
   }  
  オバケの状態が格納されるクラス**
* **int GetId()：オバケのIDを返す関数**
* **Point GetPos()：オバケの位置を返す関数**
* **bool GetDoA()：オバケが生存しているかどうかを返す関数(true：生存，false：死亡)**
* **class Log{  
   Player player  
   Action\_Label al  
   Action\_Result ar  
  }**

**ゲームログが格納されるクラス**

* **Player player：そのターンの行動プレイヤーID**
* **Action\_Label al：そのターンの行動の種類 (移動)**
* **Action\_Result ar：そのターンの行動の結果**

**関数**

* **int Function.NowTurn()  
  現在のターン数を返す関数**
* **Log Function.Log(int turn)  
  turnで指定したターン目のゲームログを返す関数  
  turnに0未満，または現在のターン数以上を指定した場合ダミーデータが返ってくる**
* **Player Function.MyPlayer()  
  自身のプレイヤーIDを返す関数**
* **GhostState Function.MyGhostState(int id)  
  idで指定した味方のオバケの状態を返す関数  
  idに0未満，または7以上を指定した場合ダミーデータが返ってくる**

**＊2020/06/08修正**

* **List<Point> Function.EnemyGhostState()**

**生存している敵のオバケの座標値を返す関数**

**座標値を要素に持つリストが返ってくる**

**返り値：[Point(x0,y0), Point(x1,y1), …, Point(x5, y5) ]**

**\*生存しているオバケの座標のみ返ってくる**

**＊2020/06/08修正**

* **List<int> Function.GetDeleteEnemyNum()**

**死亡させた敵のオバケの数を返す関数**

**青のオバケの数と赤のオバケの数を格納したリストが返ってくる**

**返り値：[int(取った青オバケ数), int(取った赤オバケ個数)]**

* **bool Function.CanDR(Point pos, Direction dir)  
  pos の位置からdirの方向への移動および攻撃が可能かどうかを返す関数**

**行動が可能ならtrue，不可能ならfalseが返ってくる**

**変数**

* **const int global.Range\_X = 5  
  フィールドの横幅が格納されている変数**
* **const int global.Range\_Y = 6  
  フィールドの縦幅が格納されている変数**
* **const int global.GhostNum = 6  
  各プレイヤーのオバケの最大数が格納されている変数**
* **const int global.FinalTurn = 200  
  最大ターン数が格納されている変数**

**列挙型**

* **enum Player**
  + **P1：先攻のプレイヤー**
  + **P2：後攻のプレイヤー**
  + **Dummy：ダミーデータ**
* **enum Direction**
  + **Up：上**
  + **Right：右**
  + **Down：下**
  + **Left：左**
  + **Dummy：ダミーデータ**
* **enum Action\_Label**
  + **Move：移動**
  + **Dummy：ダミーデータ**
* **enum Action\_Result**
  + **Miss：移動による衝突なし，攻撃が当たらなかった**
  + **P1Death：衝突や攻撃によりP1側のオバケのみが倒れた**
  + **P2Death：衝突や攻撃によりP2側のオバケのみが倒れた**
  + **Dummy：ダミーデータ**

# **８．備考**

**AIは自身が1P側として作成してください**