玉当てゲーム説明書

2012年 6月 30日
TA 知的システムデザイン研究室 M2 宮地 正大

目次

はじめに

本プログラムはJavaプログラミング1 吉見クラスの講義課題として作成されたプロジェクトです.

本プロジェクトでは講義で学んだことを生かして，対戦ゲームのAI（人工知能）を作成します．単純に与えられたことをこなす課題ではなく，制約条件の中でどうすればAIが強くなるかを考え，それを自ら実装することでプログラミング力だけでなく自由な発想を養って欲しいと考えています．講義が進むに連れて，ゲームの制約条件を増やしていき，より高度な環境でAIの思考パターンを作成してもらう予定です.

玉当てゲーム

概要

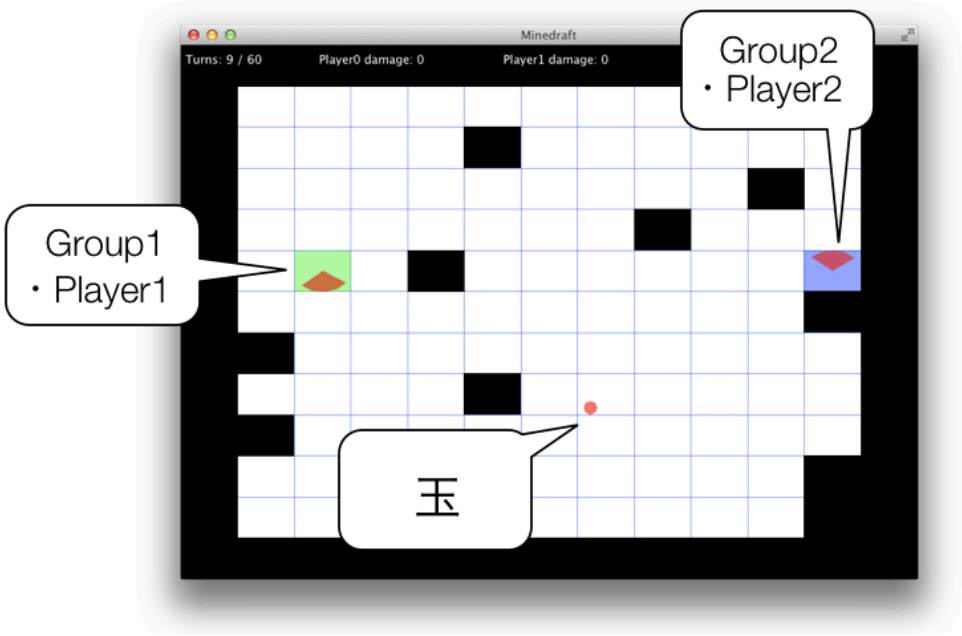
障害物のあるフィールド上で，キャラクター同士が玉を投げ合い，当てた玉の数を競う単純なゲームです．フィールドや障害物などのAI部分以外の実装はTAが行っています．そのため，皆さんにはAIファイルだけを変更することでプレーヤーの振る舞いをプログラミングしてもらいます．その際に必要なフィールドなどのデータを取得するAPIは用意してあります（詳細は後述）.

ルール

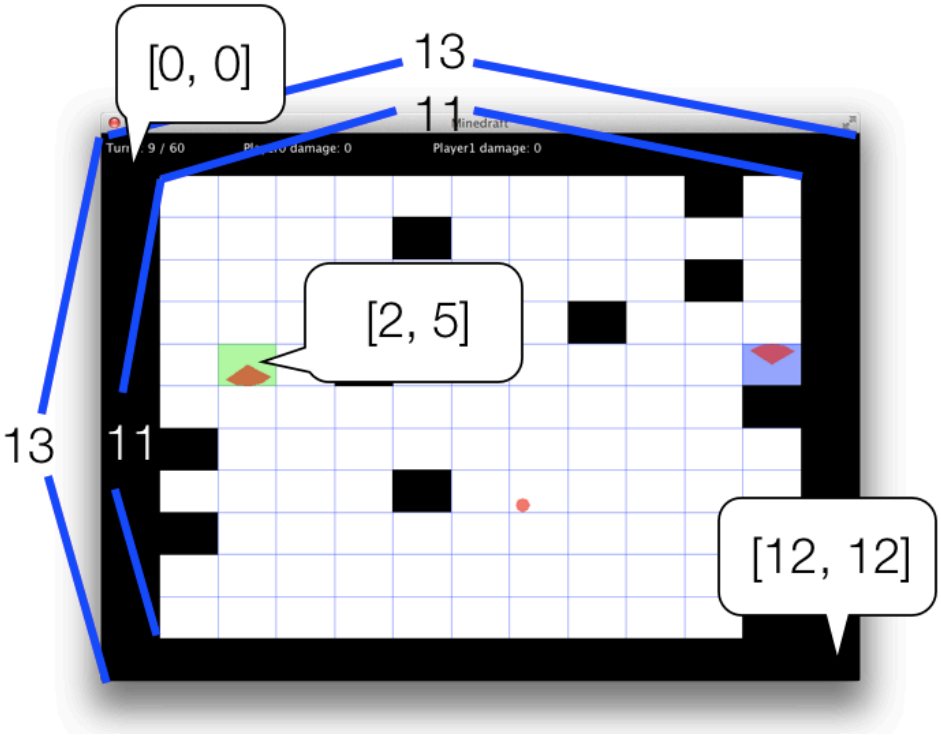
- 長方形のマス上のフィールドに障害物・プレーヤーを配置
- 外周には必ず壁（障害物）が配置，一列に一つランダムで障害物が配置される
- すべてのプレーヤーが交代で行動する
- 一回の行動でプレーヤーは 移動・方向転換・玉を投げる 動作をすることができる
- 一回の行動中で玉を投げる動作は一回まで．その他は好きに使える
- 玉を投げると何かに当たるまで玉は飛び続ける（自然消滅しない）．自分には当たらないが，味方（同じグループに所属しているプレーヤー）には当たる
- 一人のプレーヤーが行動を終了して，次のプレーヤーに行動権利が移るまでを1ターンとする（行動中に玉を投げた場合，玉が着弾するまで次のターンに移らない）
- 終了条件は指定したターン数を経過したとき
- 玉を他プレーヤーに当てた場合，加点．逆に被弾した場合，減点
- プレーヤーはグループで分けられており，グループ間での総得点で勝敗を決める

フィールド

　可変長の二次元の盤（フィールド）上に壁（障害物）・プレーヤーが定義してあります.



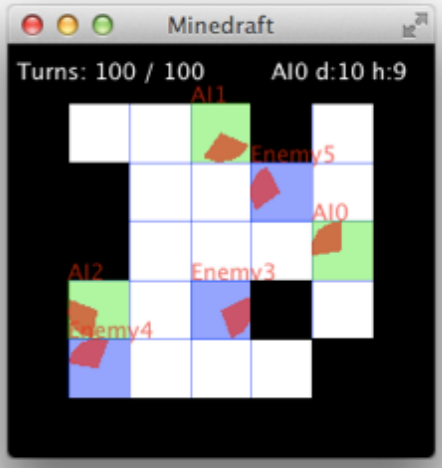
　プレーヤーは何れかのグループに所属しており，グラフィック上では同一グループのプレーヤーは同じ色として描画されます．玉は赤色の◯として描かれています．プレーヤー上の扇形は現在のプレーヤーの向いている方向を表しています.



　フィールドは左上が原点となっており，正方形のマスで構成されています．フィールドは二次元の int型配列で構成されており，マスが空白（Piece.EMPTY：-1）壁/障害物

（ Piece.WALL：-2）プレーヤー（プレーヤーの所属するグループID：0~）で表現されています．プレーヤーIDとは異なるので注意してください．直接プレーヤーIDを用いずにこのような実装になっているのは，今後敵のプレーヤーIDを取得不能にして敵味方の判別だけの情報量に制限する予定があったためです．そのため，Piece.ME および Piece.ENEMY は非推奨の定義となりました．

```
5 public abstract class Piece {
6     public static final int EMPTY = -1;
7     public static final int WALL = -2;
```



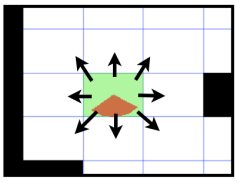

```
1 -2-2-2-2-2-2-2
2 -2-1-1 0-2-1-2
3 -2-2-1-1 1-1-2
4 -2-2-1-1-1 0-2
5 -2 0-1-1-2-1-2
6 -2 1-1-1 1-2-2
7 -2-2-2-2-2-2-2
```

上記の図の例では，全てのマスと数字が対応しています．グループIDが0の緑色のチームに所属するプレーヤーが存在するマスには0が，グループIDが1の青色の場合は1が入力されています．空マス / 壁かどうかの判定式には Piece.EMPTY / Piece.WALLを用いてください．番号の割り振りが変化する可能性があるので，数値決め打ちのハードコードしないでください．

行動

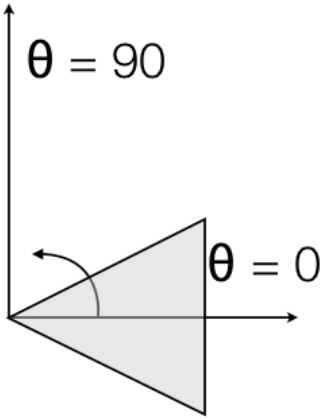
- 移動

現在位置から周囲8マスのうち，空マスの場所に移動可能です．
移動用関数 move は引数にPoint型をとっており，移動可能なPointを与えるとプレーヤー位置が移動します．



- 方向転換

現在のプレーヤーの向いている向きを変更します．角度の扱いは極座標系で右が原点で反時計回りに回転します．角度変更用の関数 angle には int型で引数を与えた場合には度数(0 ~ 360)で扱い，float型で与えた場合にはラジアン(0 ~ 2 π)として扱います．



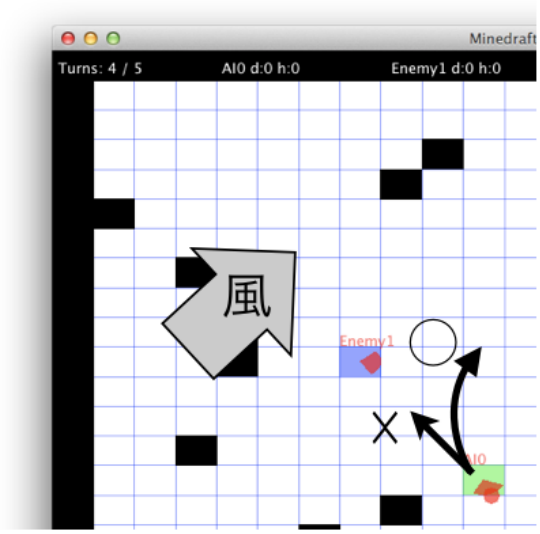
例えば，angle(45) と angle((float)(Math.PI / 4f))は同じ挙動をします.
ここで一つ注意する必要があるのが，ゲーム中で定義しているフィールドは左上原点なので，感覚的には上下反転した形でグラフィックが描画されます．つまり，angle(45) の状態で玉を投げても，右上には飛ばず，フィールド上での扱いと同じく上方向が下に向かって表現されるので，右下に向かって飛びます.

- 玉を投げる

指定した方向 or 現在向いている方向に玉を投げます.
角度の扱いは上記「方向転換」と同様です．投げる関数 throwing は angle と同じく，int型で引数を与えた場合には度数で扱い，float型で与えた場合にはラジアンとして扱います．引数を与えない場合には，そのままの向きに玉を投げます．引数を与えた場合には与えられた向きで angle関数を呼び出して向きを変えた上で玉を投げます.
その際の当たり判定方法は，t 秒 * Piece.DYNAMICS を加算した値で得られる玉の位置と，物体の位置が同一マス上にある場合は当たりとします．なので，直線軌道上ではあたってるはずでも，運良くカスる形で当たらない可能性があります．1マスの大きさが 1.0f，玉の移動に関する物理ダイナミクス（増加量）が 0.05f で定義されているため，ほとんどありえませんが.
6/22配布版の状態では玉は何かに当たるまで直進し続けましたが，6/29版より，風の概念を追加し玉の軌道が曲るようになっています.

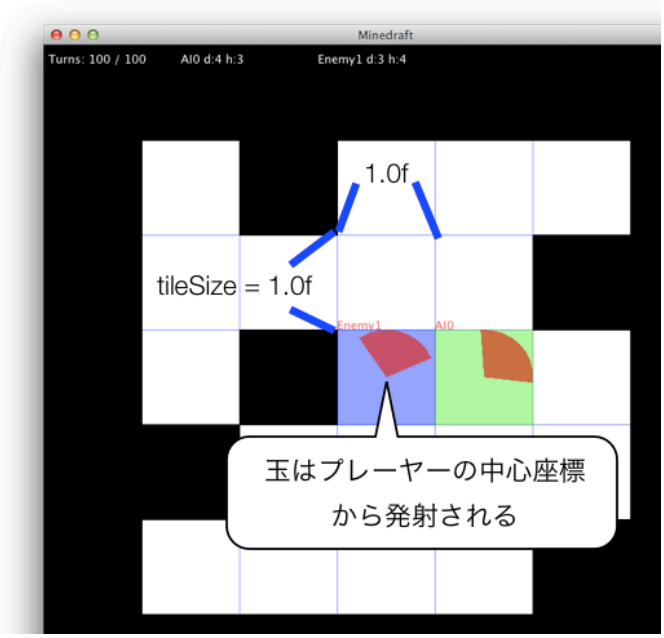
風について

　前回までは玉は投げると，まっすぐ進むことが保証されていたが，特定の向き，強さで風が吹いて玉が曲がりながら飛ぶようになりました．

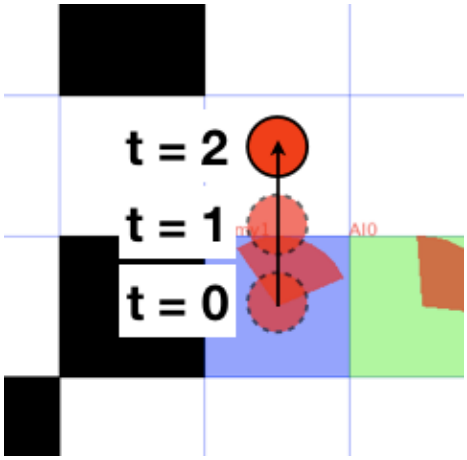


　上記図のように，風によって本来の玉の軌道とは違う場所に着弾する可能性があります．また，風の情報は取得できず，玉を投げた角度，着弾位置から自分で計算しましょう．

　この風の概念によって，今まで意識することがなかったフィールド内部で扱っていた座標系も考慮する必要があります．まず，大事なのが1マスの大きさをどのように定義しているかで，風が軌道に与える影響が変わってきます．ゲームで用いるマスの大きさはBoardクラスの tileSize変数で管理されており，1.0fで設定されています．



また，玉は投げた人の中心座標から発射されています．



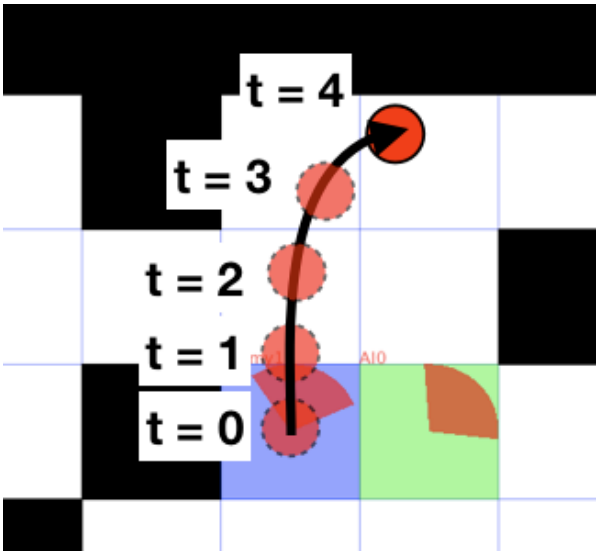
この図のように，投げた人の中心から玉が発射(t = 0) してから，玉の持つベクトルに従って n 秒後の玉の位置が決定します．

風を考慮していない場合の玉の位置計算式は以下の通りです (t = 1)．

```
// 自分のインスタンス取得
Player me = (Player)board.getME();
// 自分の位置取得
Point me_point =  board.getPosition(me.getID());
// 玉の位置初期化 投げた人の中心座標
float arrow[] = {
    // arrow[0] -> 玉(弓矢)のx座標，arrow[1] -> 玉のy座標
    me_point.x*board.tileSize+board.tileSize/2,
    me_point.y*board.tileSize+board.tileSize/2
};
// 玉の移動に関する物理ダイナミクス分だけ移動する
arrow[0] += Math.cos(angle)*Piece.DYNAMICS;
arrow[1] += Math.sin(angle)*Piece.DYNAMICS;
// 盤上の座標からフィールドにおけるポイントに変換
Point bpoint = board.convertRealToBoard(arrow[0], arrow[1]);
```

arrowは玉の盤上における位置（Pointではありません）を表現しているfloat型の2次元配列です．玉を投げた人の中心座標に初期化して，玉の移動に関する物理ダイナミクス分だけ移動します．そして，玉の盤上の座標からフィールドにおけるポイントに変換します．以上の手順で t = 1 のときの玉の位置を予測することができます．t = n のときに玉の位置がどうなるかは各自考えてください．

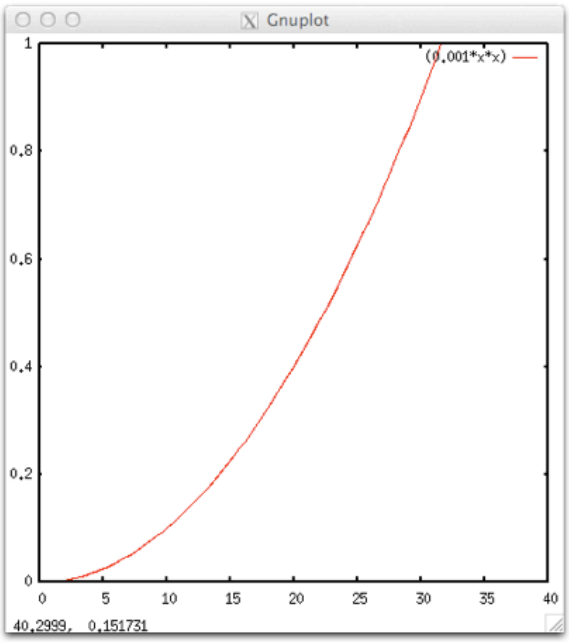
次は風の影響を考慮した場合を考えます．風は玉に対し，その方向に応じて移動ベクトルを加算します．ただし，常に一定量を与えると仮定すると，玉の軌道は単純に斜めに移動することになります．それでは玉の軌道が曲るようなものを表現できていません．なので時間に応じて加算量が変化するようなモデルを仮定します．さらに線形な単純加算では円弧を描くだけになるので，指数関数的に増加するモデルを考えます．また，現実では空気抵抗などの影響により最大速度が決まっているのでそれを表現する必要があります．



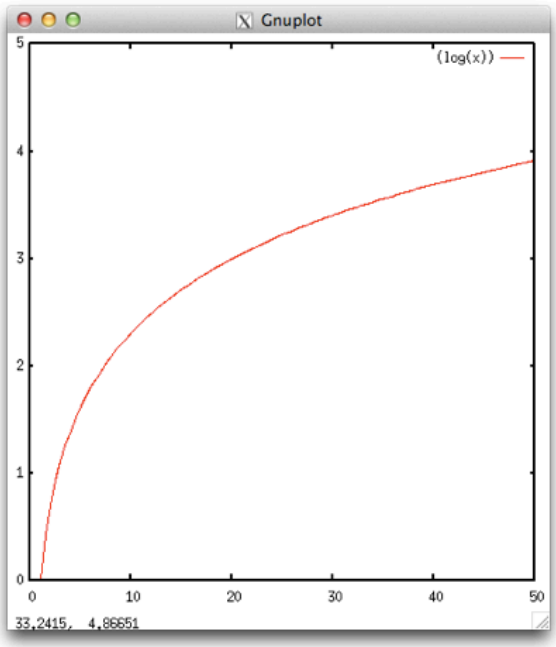
この通り，t = 1 ではあまり影響がないが，時間が進むに連れて影響が大きくなる状況を目指します.

以上から以下の通り，モデル化を行いました.

$$y = \begin{cases} \log t & (y > 1) \\ 0.001 * t * t & (otherwise) \end{cases}$$



y = 0.001*t*t



y =log(t)

縦軸 y は 横軸 t 秒のときどれだけ風による移動量の加算を行うかの係数を示しています．つまり，y > 1 のとき，t = 0 から t = 32 あたりまでは左のグラフのような加算を行

い，それ以降は右のグラフに従って移動量の加算が行われます．これはTAの読み違いですが，右のグラフに移行した途端，加算量が大きく変わります．なので感覚的に風の影響を模倣してるかどうかはわかりませんが，今回はこのモデルに従って計算を行います．

実際にプログラムに落とし込んだ，風を考慮した玉の軌道計算式は以下の通りです．

```
float arrow[] = { // 玉の位置初期化 投げた人の中心座標
    getPosition(player.getID()).x*tileSize+tileSize/2,
    getPosition(player.getID()).y*tileSize+tileSize/2
};
// 玉(弓矢)オブジェクト生成、今は同時に飛ぶことがないからいらない
Arrows.add(arrow);
int t = 1;
double dynamics = 0;
while(true){
    dynamics = (0.001)* t*t;
    t++;
    if (dynamics > 1) {
        // dynamics = 1;
        dynamics = Math.log(t);
    }
    // arrow[0] -> 玉(弓矢)のx座標，arrow[1] -> 玉のy座標 初期値は投げた人の中心座標
    arrow[0] += Math.cos(angle)*Piece.DYNAMICS
        + Math.cos(Math.PI*((float)WIND_DIRECTION/180.0))*WIND_DYNAMICS * dynamics;
    arrow[1] += Math.sin(angle)*Piece.DYNAMICS
        + Math.sin(Math.PI*((float)WIND_DIRECTION/180.0))*WIND_DYNAMICS * dynamics;
    // 盤上でのポイントに変換
    Point bpoint = convertRealToBoard(arrow[0], arrow[1]);
    if(getPoint(bpoint.x, bpoint.y) != Piece.EMPTY){
        // 壁かどうか判定
        if(getPoint(bpoint.x, bpoint.y) != Piece.WALL){
            Player p = getPointPlayerReal(bpoint.x, bpoint.y);
            // 玉を投げた人とあたった人が同一だった場合、無視
            if(p.getID() != player.getID()){
                hit.x = bpoint.x;
                hit.y = bpoint.y;
                p.damage();
                player.hit();
                break;
            }
        }else{
            hit.x = bpoint.x;
            hit.y = bpoint.y;
            break;
        }
    }
```

このプログラムはAIクラス用のサンプルプログラムではなく，実際にBoardクラスで定義されている玉の軌道計算プログラムの抜粋です．

風の向きおよび強さを定義している WIND_DIRECTION および WIND_DYNAMICS は Boardクラスのプライベート変数として定義されています．なので，AIクラスからはこの変数の値にアクセスすることができません．なので，発射角度と着弾位置（ throwing関数は返り値に着弾位置のポイントを返す ）からそれらを予測するシステムが必要になります．AIクラスでどのようにして風の影響を受けた玉の軌道計算をするべきかは各自考えてください．

プレーヤー / フィールド情報の取得

AIクラスからある程度，Boardクラスで管理されているフィールドの情報にアクセス可能です．（※ 授業中に配布したスライド中の説明が途中からずれていました．正しくはこちらです）

- Vector<Point> getMovablePos()
自分が移動することができるすべての位置を返す
- Vector<Point> getPlayers()
すべてのプレーヤーの位置を返す
- Vector<Player> getPlayersObject()
すべてのプレーヤー インスタンスを返す
- Vector<Point> getPlayersPosition()
すべてのプレーヤーの位置を返す
- Vector<Point> getEnemies()
敵(自分以外のグループに所属しているすべてのプレーヤー)の位置を返す
- Vector<Player> getEnemiesObject()
敵(自分以外のグループに所属しているすべてのプレーヤー)のインスタンスを返す
- Vector<Point> getEnemiesPositon()
敵(自分以外のグループに所属しているすべてのプレーヤー)の位置を返す
- Vector<Point> getMembers()
自分(と同じグループに所属しているプレーヤー)の位置を返す
- Vector<Player> getMembersObject()
自分(と同じグループに所属しているプレーヤー)のインスタンスを返す
- Vector<Point> getMembersPositon()
自分(と同じグループに所属しているプレーヤー)の位置を返す
- Player getME()
自分のプレーヤーインスタンスを返す
- Vector<Point> getHazard()
すべての障害物の位置を返す
- void showBoard()
フィールド情報を出力
- int[][] getBoard()
フィールドを返す(コピーなのでこれを直接弄ってもゲームに影響はない)
- int getTurn()
今 何ターン目か返す
- int getMaxTurn()
最大ターン数を返す
- Point getPosition(int id)
指定されたID（プレイヤーID）のオブジェクトがどこにいるのか返す
- int getPoint(int x, int y)
指定したフィールド上に何がいるのか返す(空白 壁 グループID のどれか)
- Player getPointPlayer(int x, int y)
指定した盤上の位置いるプレーヤーインスタンスを返す

- Point convertRealToBoard(float x, float y)
  数値的な盤上の位置からフィールドにおけるポイントを返す

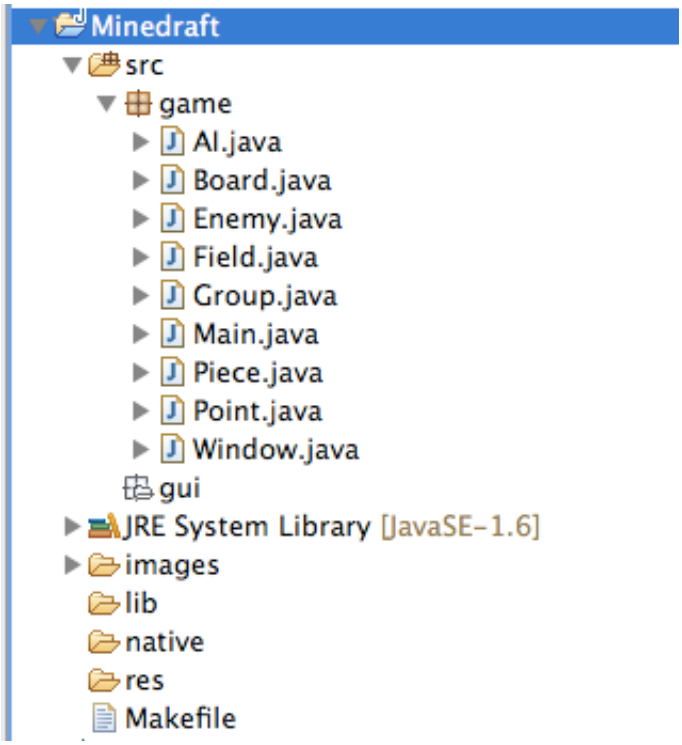
また，取得したプレイヤーインスタンスから，以下の情報も取得することができます.

```
interface Player{
    public static final int MAX_ENERGY = 1000;
    public static final int THROW_VAL = 300;
    public static final int MOVE_VAL = 200;
    public static final int REFRESH_VAL = 1000;
    public Object clone();
    public int getType();
    public int getID();
    public int getGroupID();
    public int getDamage();
    public int getHitCount();
    public float getAngle();
    public float setAngle(float angle);
    public String getName();
    public int getEnergy();
    public int damage();
    public int hit();
    public boolean refresh();
    public boolean spendEnergy(int energy);
    public void onTurn(Board board) throws Exception;
}
```

- int getID()
  このプレーヤーIDを返す
- int getGroupID()
  このプレーヤーが所属するグループIDを返す
- int getDamage()
  このプレーヤーが玉を受けた数を返す
- int getHitCount()
  このプレーヤーが玉を当てた数を返す
- float getAngle()
  このプレーヤーの向きを返す（0~360度）
- String getName()
  このプレーヤーの名前を返す

プロジェクト構成

講義中に配布したJavaプロジェクト(6/29配布版)は以下の構成になっています.



基本的に src ディレクトリにある game パッケージを変更します.

- AI.java
  プレイヤーがどういう振る舞いをするか決定するためのAIが記述されたクラス

- Board.java
  フィールドを定義しているクラス

- Enemy.java
  敵プレイヤーがどういう振る舞いをするか決定するためのAIが記述されたクラス

- Field.java
  画面描画用クラス

- Group.java
  プレイヤーを管理するグループを定義したクラス

- Main.java
  本プロジェクトのメインクラス

  ゲームのメインスレッドとAIの抽象クラスが記述されている

- Piece.java
  フィールド上に配置しているオブジェクトのインデックスや静的な変数などを定義して

いるユーティリティクラス

- Point.java
  ゲーム中のフィールド表現でよく使う二次元のインデックスを保持したクラス

- Window.java
  画面描画用のフレームを生成するクラス

FAQ ~よくある質問~

1. Q：angleに45とか入力しても，右上に玉を投げてくれずに右下に投げています．反時計回りの座標系になってませんか
   A：ウィンドウ上で表現されているフィールドは左上原点です．なので縦のY軸方向は加算すると下に向かって増加します．内部的には反時計回りで扱っていますがこの理由から上下反転しているように見えます． これの関しては混乱を避けるために左下原点に変換することも考えましたが，すでにAIを作ってくれている人のことも考えて変更しないことにしました．ご了承ください．
   また，プレーヤーの向きに関してはこの通り扱っていなかったので向いている方向と上下反対方向に玉が発射されていました．これに関しては最新版で修正済みです．

2. Q：showBoard関数（現在の盤面をコンソールに出力する関数）で出力される結果とGUIで表現されている結果が，間違っていませんか
   A：間違っていました．出力時の縦横のインクリメント変数が入れ替わっていました．最新版では修正済みです．

3. Q：getPosition関数の挙動がおかしい
   A：間違ってました．一致するプレーヤーIDを持つ位置を返すつもりがオブジェクトID（障害物・空白・グループID）で一致する位置を返してました．修正済み．

4. Q：ゲームスピード変えられないのか
   A：Boardクラスで定義しているint SLEEP_TIME変数で玉の進むスピードをコントロールしています．この値を小さくすると描画のためにあえてゆっくりにしていた部分が早くなり，劇的にゲームスピードが早くなります．ただ，早くしすぎると玉が見えなくなります．
   さらに早くする場合にはMainクラスで定義されているboolean iswindow変数をfalseにしてください．ゲームスタート時にウィンドウを生成しないコンソールモードで起動するので結果が早く出ます．ただ何やってるのかわけわかりません．

5. Q：変更するのはAIクラスを継承したAiAlgorithmクラスのmoveメソッドだけですか
   A：提出するのがAI.javaファイルで，こちらが用意しているメイン文でそのファイルが動作すれば基本的に問題ありません．なのでAiAlgorithmクラスのクラス変数や内部クラスなどを駆使して高度なAIを記述してください．今までの行動履歴などはAPI側で用意してないので，自分でクラスを作って管理するとよいでしょう．