Project: Knockabout

Maxime CHAMBREUIL
McGill ID: 260067572
maxime.chambreuil@mail.mcgill.ca

Contents

1 Introduction

In the AI course at McGill University, each student has the opportunity to implement an AI player to a chosen game. The game chosen for the Fall 2003 term is Knockabout. You can find the rules of this game at http://www.pair-of-dice.com/games/knockabout.html.

2 Algorithm

2.1 Principle

The main idea of Knockabout is based on space layout: further a die is from the middle of the board, more chance it has to be in the gutter. This is the heuristic I used inside my implementation of the Monte Carlo algorithm.

2.2 Algorithm

It consists in generating all the possible that I can do. For each of this move, I clone the current board to apply it. I obtain the first step board, called ”firstBoard”.

Then, I generate all the possible move of my opponent to apply each of them to a clone of ”firstBoard”. I finally obtain the second step board, called ”secondBoard”.

On ”secondBoard”, I evaluate the board : I compute the distance between my dies and the middle (”myTotalDistance”), and the distance between opponent dies and the middle of the board (”oppTotalDistance”). I add those distance at each move of the opponent and average their difference to compute the evaluation of my move.

I finally choose the move with the minimum.

2.3 UML Class diagram

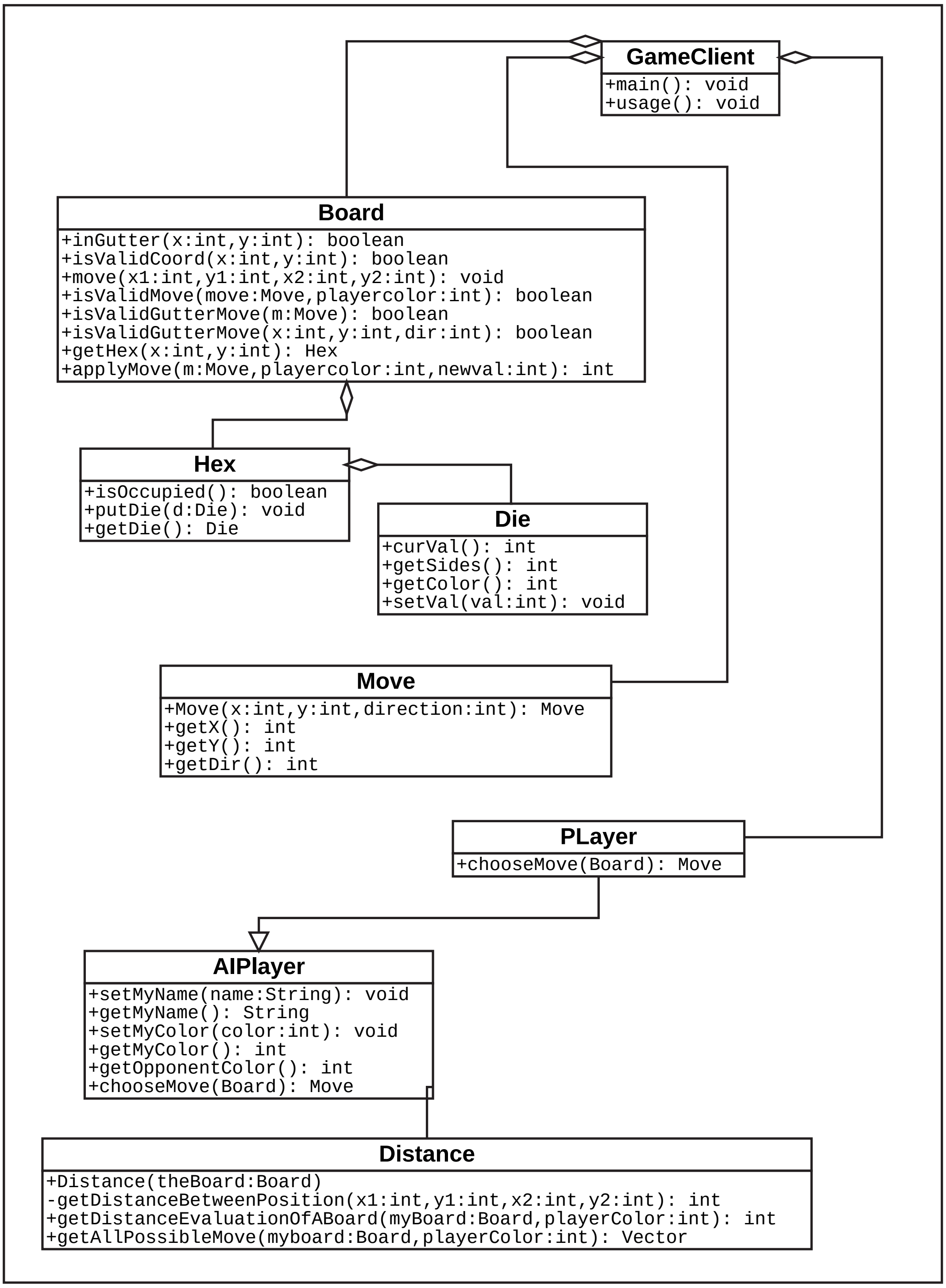
GameClient
+main(): void
+usage(): void
Board
+inGutter(x:int,y:int): boolean
+isValidCoord(x:int,y:int): boolean
+move(x1:int,y1:int,x2:int,y2:int): void
+isValidMove(move:Move,playercolor:int): boolean
+isValidGutterMove(m:Move): boolean
+isValidGutterMove(x:int,y:int,dir:int): boolean
+getHex(x:int,y:int): Hex
+applyMove(m:Move,playercolor:int,newval:int): int
Hex
+isOccupied(): boolean
+putDie(d:Die): void
+getDie(): Die
Die
+curVal(): int
+getSides(): int
+getColor(): int
+setVal(val:int): void
Move
+Move(x:int,y:int,direction:int): Move
+getX(): int
+getY(): int
+getDir(): int
PLayer
+chooseMove(Board): Move
AIPlayer
+setMyName(name:String): void
+getMyName(): String
+setMyColor(color:int): void
+getMyColor(): int
+getOpponentColor(): int
+chooseMove(Board): Move
Distance
+Distance(theBoard:Board)
-getDistanceBetweenPosition(x1:int,y1:int,x2:int,y2:int): int
+getDistanceEvaluationOfABoard(myBoard:Board,playerColor:int): int
+getAllPossibleMove(myboard:Board,playerColor:int): Vector

Figure 1: UML Class diagram of my client

2.4 Java Code - AIPlayer.chooseMove(Board)

```
public Move chooseMove(Board board){
  Distance myDistance = new Distance();
  int myTotalDistance = 0, oppTotalDistance = 0;
  int maxDistance = 0, int minDistance = 100000;
  int newval = 0;
  Move myMove = new Move(0,0,0);
  Vector moveList = myDistance.getAllPossibleMove(board,getMyColor());
  Vector oppMoveList = new Vector();
  Board firstBoard,secondBoard;

  // For all my possible move
  for (int i=0;i<moveList.size();i++){

    // Clone the board to apply the move
    firstBoard = board.deepclone();

    // Apply the current move to the board
    firstBoard.applyMove( (Move) moveList.get(i),getMyColor(),newval);

    // Get the list of the opponent possible move
    oppMoveList = myDistance.getAllPossibleMove(firstBoard,getOpponentColor());

    // (Re)-Initialization
    oppTotalDistance = myTotalDistance = 0;

    // For all opponent possible move
    for (int j=0;j<oppMoveList.size();j++){

      // Clone the board to apply the move
      secondBoard = firstBoard.deepclone();

      // Apply the current move to the board
      secondBoard.applyMove( (Move) oppMoveList.get(j),getOpponentColor(),newval);

      // Evaluate the board and add the value to the total distance
      myTotalDistance = myTotalDistance + myDistance.
                     getDistanceEvaluationOfABoard(secondBoard,getMyColor());
      oppTotalDistance = oppTotalDistance + myDistance.
                 getDistanceEvaluationOfABoard(secondBoard,getOpponentColor());
    }

    // We substract and we average
    myTotalDistance = (myTotalDistance - oppTotalDistance)/oppMoveList.size();

    if (myTotalDistance<minDistance){
      minDistance = myTotalDistance;
      myMove = (Move) moveList.get(i);
    }
  }
  return myMove;
};
```

3 Download

You can download

- the server
- the random player
- my Minimax player
- my Monte Carlo player (described in this report)

on my website at :

http://www.maxime-chambreuil.fr.st/education/mcgill/5.1/424/

4 Behaviour and results

This algorithm is a good compromise between increasing the total distance of the opponent and decreasing mine : I can then decide the agressivity of my player.

When I compute the distance between the die and the middle, I set the distance to 1 000 if the die is in the gutter. Thus the program knows the objective of the game: putting the opponent dies in the gutter and avoid mine to be there.

Concerning the result, my AI player is better than me, and all the player I have tested with.

5 Conclusion

This project was a good and funny application of the class, even if I did not implement a complicated AI player. As it was said in class, implementing AI player is dangerous because we always want to find a new AI player, which is even better. This can be funny, but mostly time-consuming.

6 References

- www.pair-of-dice.com/games/knockabout.html
- www.cs.mcgill.ca/˜dprecup/courses/ai.html
- www.eclipse.org
- www.cvshome.org