211lab11GameModel

```
package model;

import GameIO.*;
import java.awt.Point;
import java.util.*;
import model.board.*;
import model.nextMove.*;

/**
 * A concrete model of a game.  For descriptions of the methods, see the IModel
 * and IModelAdmin interface documentation. Except for the setPlayers(),
 * getPlayers() and addPlayer() methods, whose descriptions are below.
 */
public final class GameModel implements IModel {
    /**
    * An abstract factory to create APlayers.  Used privately by GameModel.
    */
    private static interface IMakePlayer {
        /**
         * Instantiates an APlayer object given the player's "player number".
         * Player number 0 plays first, player number 1 plays second.
         * @param playerNo The player number for the player to be instantiated.
         * @return APlayer object
         */
        public APlayer create(int playerNo);
    }

    /**
    * The maximum amount of time a player is alloted to take its turn.
    */
    private int maxTurnTime = 15;   // default time in seconds

    /**
     * For player management.  Initalized to a null TurnControl object.
     */
    private TurnControl turnControl = TurnControl.NullObject;

    /**
     * Adapter to talk to the view to display/clear a game piece ("token") or
     * a String message.
     */
    private ICommand iCommand;

    /**
     * Adapter to talk to the view to announce winner, draw, or reset.
     */
    private IViewAdmin viewAdmin;

    /**
     * Adapter to talk to the view to tell that the human player needs to try a
     * move.
     */
    private ITurnAdmin turnAdmin;

    /**
     * The invariant, encapsulated rules and behaviors of a game.
     */
    private IBoardModel boardModel;

    /**
    * Initializes this GameModel to a given IBoradModel and a max turn time.
    * @param boardModel for example Othello or TicTacToe.
    * @param maxTurnTime maximum allowable time to make the next move.
    */
    public GameModel(IBoardModel boardModel, int maxTurnTime) {
        this.boardModel = boardModel;
```

```
    this.maxTurnTime = maxTurnTime;
}

// FOR STUDENTS TO COMPLETE IN THE LAB
private IRequestor requestor =
    new IRequestor() {
        public void setTokenAt(final int row, final int col, final int player,
                               final IRejectCommand rejectCommand) {
            boardModel.makeMove(row, col, player,
                new ICheckMoveVisitor() {
                    public void invalidMoveCase() {
                        // TO DO
                    }
                    public void validMoveCase() {
                        // TO DO
                    }
                },
                new IBoardStatusVisitor() {
                    public Object player0WonCase(IBoardModel host, Object param) {
                        // TO DO
                        return null;
                    }
                    public Object player1WonCase(IBoardModel host, Object param) {
                        // TO DO
                        return null;
                    }
                    public Object drawCase(IBoardModel host, Object param) {
                        // TO DO
                        return null;
                    }
                    public Object noWinnerCase(IBoardModel host, Object param) {
                        // TO DO
                        return null;
                    }
                });
        }
    };



/**
* Initializes the adapter to talk to the view.
* @param command Adapter to talk to the view to display/clear a game piece
* ("token") or a String message.
*/
public void setCommand(ICommand command) {
    iCommand = command;
}

public void reset() {
    System.out.println("Resetting");
    boardModel.reset();
    boardModel.redrawAll(iCommand);
    turnControl.setHalt();
}

/**
 * Adds players created by the supplied factories to the TurnControl.
 * Assumes that the players are IMakePlayer factory objects.
 * Any existing players are lost.   Player 0 starts the game.
 * @param player0 Factory for the starting player.
 * @param player1 Factory for the second player.
 */
public void setPlayers(Object player0, Object player1) {
    // Last in/first play
    turnControl = new TurnControl(((IMakePlayer) player1).create(1));
    turnControl.addPlayer(((IMakePlayer) player0).create(0));
    turnControl.setAdapters(viewAdmin, iCommand);
    turnControl.run(maxTurnTime);
}
```

```
public void setViewAdmin(IViewAdmin viewAdmin, ITurnAdmin turnAdmin) {
    this.viewAdmin = viewAdmin;
    this.turnAdmin = turnAdmin;
}

public IBoardModel getBoardModel() {
    return boardModel;
}


/**
 * Returns a Vector filled with IMakePlayer factory objects.
 * @return
 */
public Vector getPlayers() {
    Vector v = new Vector();
    v.addElement(new IMakePlayer() {
        public APlayer create(int playerNo) {
            return new HumanPlayer(requestor, playerNo, turnAdmin);
        }
        public String toString() {
            return "Human player";
        }
    });

    v.addElement(new IMakePlayer() {
        public APlayer create(int playerNo) {
            return new ComputerPlayer(requestor, playerNo, GameModel.this,
                                      new RandomMoveStrategy());
        }
        public String toString() {
            return "Computer RandomMove";
        }
    });

    v.addElement(new IMakePlayer() {
        public APlayer create(int playerNo) {
            return new ComputerPlayer(requestor, playerNo, GameModel.this,
                                      new RandomValidMove());
        }
        public String toString() {
            return "Computer RandomValidMove";
        }
    });

    /*
    // CODE TO ADD PLAYERS WITH OTHER KINDS OF NEXT MOVE STRATEGIES THAT
    // STUDENTS ARE REQUIRED TO IMPLEMENT:
    v.addElement(new IMakePlayer() {
        public APlayer create(int playerNo) {
            return new ComputerPlayer(requestor, playerNo, GameModel.this,
                                      new MinMax(new MinMaxFac()));
        }
        public String toString() {
            return "Computer w. MinMax";
        }
    });

    v.addElement(new IMakePlayer() {
        public APlayer create(int playerNo) {
            return new ComputerPlayer(requestor, playerNo, GameModel.this,
                                      new MinMax(new AlphaBetaFac()));
      }
      public String toString() {
        return "Computer w. AlphaBeta";
      }
    });

    v.addElement(new IMakePlayer() {
        public APlayer create(int playerNo) {
            return new ComputerPlayer(requestor, playerNo, GameModel.this,
```

```
                                    new MinMax(new DepthFac(new AlphaBetaFac(),2)));
            }
            public String toString() {
                return "Computer w. Depth 2";
            }
        });

        v.addElement(new IMakePlayer() {
            public APlayer create(int playerNo) {
                return new ComputerPlayer(requestor, playerNo, GameModel.this,
                                    new MinMax(new DepthFac(new AlphaBetaFac(),3)));
            }
            public String toString() {
                return "Computer w. Depth 3";
            }
        });
        */

        return v;
    }

    public void exit() {
        reset();
    }


    /**
     * Add a ComputerPlayer based using the INextMoveStrategy specified by the
     * fully qualified (i.e. included package name) String classname.
     * The strategy is assumed to have either a no-parameter constructor or a
     * constructor that takes an IModel as an input parameter.
     * An IMakePlayer factory is returned.
     * @param className Fully qualified class name for an INextMoveStrategy subclass.
     * @return IMakePlayer object that can construct a new Computer player with the supplied strategy.
     */
    public Object addPlayer(final String className) {
        final INextMoveStrategy strategy;
        try {
            java.lang.reflect.Constructor c =  Class.forName(className).getConstructors()[0];
            Object [][] args = new Object[][]{new Object[]{}, new Object[]{this}};
            strategy = (INextMoveStrategy) c.newInstance(args[c.getParameterTypes().length]);
        }
        catch(Exception ex) {
            iCommand.setMessage(ex.toString());
            return null;
        }
        iCommand.setMessage("");
        return new IMakePlayer() {
            public APlayer create(int playerNo) {
                return new ComputerPlayer(requestor, playerNo, GameModel.this, strategy);
            }
            public String toString() {
                return className;
            }
        };
    }

    /**
     * Get the time left for the current player's turn in milliseconds.
     * Return value is undefined if no player is currently taking their turn.
     * @return
     */
    public long getTimeLeft() {
        return turnControl.getTimeLeft();
    }
}
```

Access Permissions: (Please don't edit)

- Set ALLOWTOPICCHANGE = Main.TeachersComp211Group