GRADER

Name of student running submit: Mark Thornburg
Login of student running submit: -yn

Second team member's name:Declan Shener
Second team member's login: -qv

Third team member's name: Alexander Lindsay
Third team member's login: -mf

IMPORTANT: Once you've submitted Project 2 once, the same team member should submit always. If a different teammate must submit, inform cs61b@cory.eecs of all the details. Include a complete list of team members, and let us know which submission you want graded.

If you've submitted your project once, or even written a substantial amount of code together, you may not change partners without the permission of the instructor.

==========

Does your program compile without errors?

Have you tested your program on the 61B lab machines?

Did you successfully implement game tree search? Did you successfully implement alpha-beta pruning? Are there any limitations on it? What is the default number of search levels set by the one-parameter MachinePlayer constructor (or is it a variable-depth search)?

Describe your board evaluation function in some detail.
Evaluate determines an integer value of how good a given move is by checking to see if the move causes the machine player to win, if it blocks the opponent from forming a winning network, if it forms connections with other squares, if it places a square in an endzone when there are no other squares there, if the move changes the number of player's opponent's valid moves significantly, and if move breaks their opponent's connections.

Does your MachinePlayer use any special method of choosing the first few moves?
No, our MachinePlayer does not use any special method of choosing the first few moves?

Is there anything else the graders should know to help them read your project?

Describe the classes, modules, and interfaces you designed before and while you implemented the project. Your description should include:

- A list of the classes your program uses.
- A list of each of the "modules" used in or by MachinePlayer, similar to the list in the "Teamwork" section of the README (but hopefully more detailed). (If you're using a list class, that should probably count as a separate module.)
- For each module, list the class(es) the module is implemented in.
- For each module, say which of your team members implemented it.
- For each module, describe its interface--specifically, the prototype and behavior of each method that is available for external callers (outside the module) to call. Don't include methods that are only meant to be called from within the module.

  For each method, provide (1) a method prototype and (2) a complete, unambiguous description of the behavior of the method/module. This description should also appear before the method in your code's comments.

You will probably need to change some of your design decisions as you go; be sure to modify this file to reflect these changes before you submit your project. Your design of classes and interfaces will be worth 10% of your grade.

//Creates an object that stores a best move and the score of the move
//Implemented by Declan Shener
Best.java

public Move move
public int Score
public Best(Move move, int score)
public Best()

//A doubly-linked non-circular implementation of a list
//Implemented by Mark Thornburg
DList.java

protected DListNode head;
protected DListNode tail;
protected int size;
public DList()
public DListNode getHead() - returns the DListNode at the head of this list

public DListNode getTail() - returns the DListNode at the end of this list
public int getSize() - returns the size of this list
public void insertFront(Square i)- inserts a Square at the front of the list
public void InsertFront(Object i) - inserts an Object at the front of the list
public void removeFront() - removes the front DListNode from the list
public void removeBack() - removes the tail DListNode from the list

//A one-item DListNode with next and prev fields
//Implemented by Mark Thornburg

DListNode.java

public Object item;
protected DListNode prev;
protected DListNode next;
public DListNode()
public DListNode(int i)
public DListNode(Square i)
public DListNode(Object i)
public Object getItem() - returns the item of this DListNode
public DListNode getNext() - returns the next DListNode
public DListNode getPrev() - returns the prev DListNode
public void setItem(Object object) - sets this DListNode's item to object
public void setItem(Square square) - sets this DListNode's item to square
public DListNode findNth(long n) - finds the nth DListNode from this node indexed at 1

//Our representation of a Network Gameboard
//Implemented by Alexander Lindsay and Mark Thornburg

Gameboard.java

private Square[][] board;
private int numMoves;
public Gameboard()
protected Square[][] copyBoard() - returns a copy of the internal Square[][] of this board
protected void forceMove(Move move,int color) - Sets a square on to the board with given "color" and coordinates given by "move". Only use if you're going to remove right after!
protected void forceRemove(Move move, int color) - Removes a square on to the board with given "color" and coordinates given by "move". Used to remove a forced move
protected void SetSquare(Move move) - sets a square of this Gameboard determined by a move
protected Square getSquare(Move move) - gets a square based on the directions of a move
protected Square getSquare(int x, int y) - gets a square at coordinate (x,y)
protected void removeSquare(Move move) - removes a square based on a move
protected Square[] blackLocation() - returns an array of all black's squares on the

board

protected Square[] whiteLocation() - returns an array of all white's squares on the board

private Square[] vertConnect(Square square) - Returns an array of the squares directly above or below the inputted square if they form a connection

private Square[] horizConnect(Square square) - Returns an array of the squares directly right or left the inputted square if they form a connection

private Square[] diagConnect(Square square) - Returns an array of the squares located diagonally from the inputted square if they form a connection

protected Square[] formConnection(Square square) - returns an array of all the that form an immediate connection with the square

protected boolean isValidMove(Move move) - returns true if the move is valid according to the rules of network

protected boolean isCorner(Square square) - returns true if the square is a corner square

protected boolean isOccupied(Square square) - returns true if the square is occupied

protected boolean createsConnectedGroup(Square square) - Returns false if move creates a group of more than 2 chips with the same color (Rule #4)

protected Square[] getSurroundingSquares(Square square) - returns an array of the squares surrounding this square

protected DList allValidMoves(int clr) - generates a list of all valid moves based on this gameboard

protected boolean hasWin() - returns true if this gameboard has a win

protected boolean formsNetwork() - returns true if there is a network in this board

protected boolean formsNetwork2(int color) - returns true if their is a network for the player who has the color int

protected boolean hasNetwork(Square mainsquare, int color) -

protected boolean recurseNetwork(Square currentsquare, DList plist,int depth, int color, int direction)

protected Square[] removeMatching(Square[] masterarray, DList keylist)

protected Square[] strongFormsConnection(Square mainsquare, int direction, int color)

protected Square[] removeStartingSquares(Square[] sarray, int color)

protected boolean isStartingSquare(Square square, int color)

protected int generateDirection(Square mainsquare, Square subsquare)

protected boolean endSquare(Square square)

protected Square[] findFSquares(int color)

protected int convertColor(String color)

protected void setTest(Square square)

protected int numPieces()

// Our representation of a Machine Player
//Implemented by Declan Shener and Alexander Lindsay

MachinePlayer.java

private int color;
private int searchDepth;
private Gameboard gameBoard;
private int numMoves;
private Move best;
private int opponentColor;
public MachinePlayer(int color)
public MachinePlayer(int color, int searchDepth)
public Move chooseMove() - makes move returned by minimax method
private Best minimax(boolean maximizingPlayer, int alpha, int beta, int depth) - uses alpha-beta pruning to find optimal move and returns Best object with the best move and score within it
protected int evaluate(Move move) - Determines the value of a given “move” based off of the current gameBoard
public boolean opponentMove(Move m) -If the Move m is legal, records the move as a move by the opponent (updates the internal game board) and returns true. If the move is illegal, returns false without modifying the internal state of "this" player. This method allows your opponents to inform you of their moves.
public boolean forceMove(Move m) - If the Move m is legal, records the move as a move by "this" player (updates the internal game board) and returns true. If the move is illegal, returns false without modifying the internal state of "this" player. This method is used to help set up "Network problems" for your player to solve.

Square.java

private int color;
private int x;
private int y;
public Square(int x, int y)
protected void setColor(int color) - populates square with piece of certain color
protected int getXCoordinate() - returns x-coordinate of square
protected int getYCoordinate() - returns y-coordinate of square
protected void setX(int x) - sets piece to new x-coordinate
protected void setY(int y) - sets piece to new y-coordinate
protected int getColor() - returns -1 if space is empty, otherwise returns color(int) of piece inhabiting space