CSE 4345 Requirements and Design Document

Nathan Moore, Gustavo Castillo, Peter DeNicola

Non-Functional Requirements:

Requirement 1

Description: The product shall be called ‘Halma 3.0’.

Requirement 2

Description: The product shall be a client/server web-service.

Requirement 3

Description: The product’s server-side code shall be written in Python 2.7.

Requirement 4

Description: The product will be able to accept a Json parameter as input and interpret it.

Requirement 5

Description: The product shall output a formatted Json output to be read by an AI program.

Requirement 6

Description: The product shall be able to accept user input to change the AI program

Requirement 7

Description: The product shall use HTML5 in the GUI in the front end

Requirement 8

Description: The product shall use Javascript for the front end logic

Requirement 9

Description: The product shall connect multiple AIs through HTTP POST requests

Requirement 10

Description: The product's client-side code shall be written in HTML and JavaScript.

Functional Requirements:

Requirement 11

Description: The product shall display an 18 cell by 18 cell playing board for the user.

Use Case: The user of the product will want to be able to see the progress of the two, competing AIs in order to verify their effectiveness and correct operation. Therefore, the game engine of the product shall display a visual representation of the game board. In this case, the board shall have 18 cells in each row and 18 cells in each column. The user merely needs to see the moves the AIs make.

Requirement 12

Description: The product shall display 12 circular pieces for each user's AI.

Use Case: The user of the product will want to be able to see the progress of the two, competing AIs in order to verify their effectiveness and correct operation. Therefore, the game engine of the product shall display a visual representation of the Halma game pieces. Each piece shall be represented as a circular object inside of one of the cells on the game board. Each piece shall be placed in different cells based on the moves made by its controlling AI. The user merely needs to see the moves the AIs make.

Requirement 13

Description: The product shall link two AIs to a UI that will be their interface to the game engine and compete against each other.

Use Case: The game will be played by two AIs instead of two human users. The two AIs will interact with each other through an interface that will also connect them to the primary game engine. The AIs shall connect as clients to the game engine server. The AIs must be properly programmed in order to the play the Halma game according to the rules established by the game engine, otherwise, errors will result during gameplay. Moreover, the AIs must be able to send and receive data via JSON in order to interface with the UI.

Requirement 14

Description: The product shall implement a timer to control when the AIs take their turns.

Use Case: The AIs will interface with each other and the game engine via a UI. The game engine shall implement a timer that it will use to request an AI to take its next turn at regular intervals. This is because the AIs must take turns and play the game in an orderly fashion. The user should be able to keep track of the AIs' moves using the GUI, and, therefore, the AIs will have to wait a few seconds between turns for the sake of the user. The AIs must not take turns on their own but only when the game engine requests their next turn, otherwise, there will be errors.

Requirement 15

Description: The product shall display moves as they occur.

Use Case: As the AI's make their turns a spectator will be able to see the specific jumps taking place by looking into the console. Otherwise, the board will be updated visually with each move.

Requirement 16

Description: The product shall display visual notification when an AI's piece has entered the correct goal destination. The same will be true when a piece enters an opponent's destination.

Use Case: When an AI's piece reaches the goal destination, a visual cue such as a change in color will alert the user to this occurrence. If a piece leaves the destination, the color will be removed to indicate this as well. When a piece enters an opponent's destination, it will glow a different color to indicate the occurrence.

Requirement 17

Description: The product shall enter a Game Over mode when one AI has completed their goal. This mode will allow the option to view the move list or start a new game.

Use Case: User will be able to interact with several buttons that are added to a Game Over screen allowing the user to repick AI's, restart game, or view the move list. This should only be available after the game has been completed by one AI.

Requirement 18

Description: The product shall implement error handling to handle improper responses from the AIs to prevent crashes.

Use Case: The software will attempt to handle and deal with any errors it detects in response from the AI's so that whatever response is received will not crash the overall program.

Requirement 19

Description: The product shall have 3 second intervals in between moves

Use Case: The software should have a reasonable interval in between moves. Users should not wait too long or too short for the program to make a move through a players AI.

Requirement 20

Description: The product shall validate moves to ensure that they are legal moves being made, if an illegal move is attempted then the game will display an illegal move message.

Use Case: The software should check a move received to ensure that it is a legal move as to maintain the integrity of the game. Sometimes an AI will return an invalid move and the software needs to detect that to ensure that it does not make the move and displays an error message.

Requirement 21

Description: The product shall ensure that if a piece attempts an illegal move, it will penalize the piece with a loss of move during that turn

Use Case: When an illegal move is attempted, there should be a penalty. Because every piece is given a chance to move, and all pieces are equally important, we believe that not allowing the piece to complete the turn intended is penalty enough.

Requirement 22

Description: The product shall allow an AI to move all of its pieces once per turn.

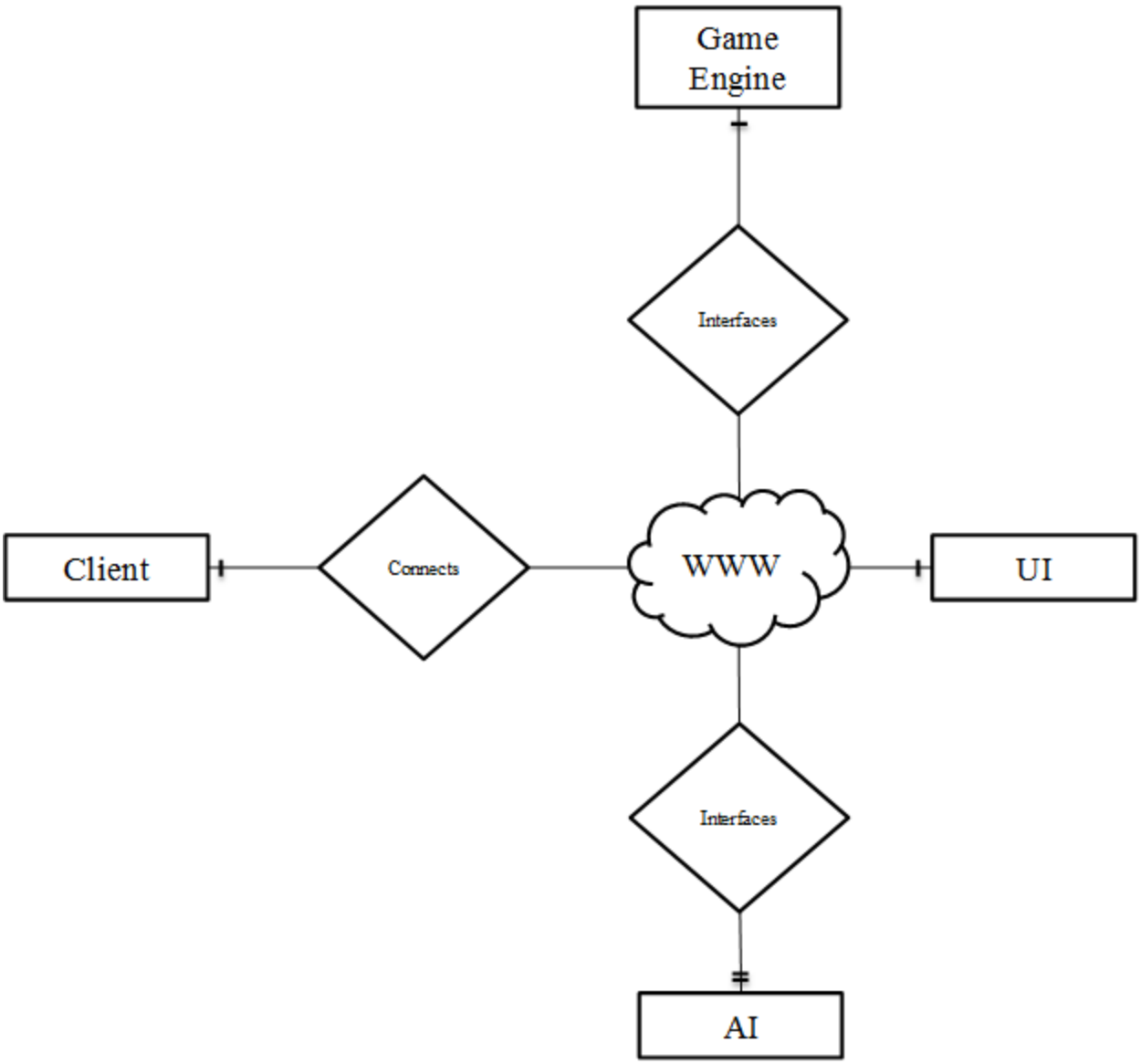
Use Case: AI games can be drawn out; thus, by allowing all pieces to be moved in a single turn, the game will speed up.

Requirement 23

Description: The product shall implement all remaining features of Halma 2.0 Multiplayer.

Use Case: The user can expect to be able to make moves, including jumps, and make use of all the other features already developed for the Halma game.

Software Architecture Diagram:



Source Code Control System

We will be using GitHub: https://github.com/nathandmoore/Halma_3_0

Source Code:

halmaAI_dumb.py

```
#!/usr/local/bin/python
# -*- coding: UTF-8 -*-

# AUTHOR: NATHAN MOORE
# URL: http://lyle.smu.edu/~ndmoore/cgi-bin/halmAI_dumb.py
# Does not take enemy pieces into account

import cgi
import cgitb
import json
import ast

cgitb.enable()


class Cell:
    """ Class: Cell
    => Description:
        Intended to represent pieces and destinations
    => Class variables:
        . int x: coordinate on game board
        . int y: coordinate on game board
        . bool arrived: true if the piece has arrived at destination
        * Upper left corner of board is 0,0
    """

    def __init__(self,x,y):
        self.x = x
        self.y = y
        self.arrived = False


    @property
    def x(self):
        return self.__x


    @property
    def y(self):
        return self.__y


    @property
    def arrived(self):
        return self.__arrived


    @x.setter
    def x(self,x):
        self.__x = x


    @y.setter
    def y(self,y):
        self.__y = y
```

```
    @arrived.setter
    def arrived(self,arrived):
        self.__arrived = arrived



def getMove(pieces,destRegion,pieceToMove):
    """ Function: getMove
        => Description:
            Calculates the next move the AI should make
        => Parameters:
            . pieces: list of cells representing the pieces on
                the board
            . destRegion: list of cells representing the
                destination area
            . pieceToMove: the piece to move next
        => Returns:
            . nextMove: list with the location of the piece to
                move and the location to move that piece to
    """

    nextMove = {}
    # Select piece to move and destination to move to:
    # Make sure that the piece isn't already in the destination
    #     region
    nextMove["destx"] = pieces[pieceToMove].x
    nextMove["desty"] = pieces[pieceToMove].y
    nextMove["piecex"] = pieces[pieceToMove].x
    nextMove["piecey"] = pieces[pieceToMove].y

    if pieceToMove > 8:
        destCell = destRegion[pieceToMove-3]
    else:
        destCell = destRegion[pieceToMove]

    if (pieces[pieceToMove].arrived and destCell.arrived):
        return nextMove
    elif (not pieces[pieceToMove].arrived) and destCell.arrived:
        numDests = len(destRegion)
        for dest in range(0,numDests):
            if not destRegion[dest].arrived:
                destCell = destRegion[dest]
                break

    # if the piece to move has not arrived at its destination
    # simulate the next move to take
    directMove = makeMove(pieces[pieceToMove],destCell)
    # Modify the move if a jump is available
    #  Check if a piece is already in the location you want to
    #     move to
    #  if yes: check next location along path
    #      if yet another piece: skip move
    #      if not: move to that second location (jump)
    #  if no: move to location
    nextMove = determineJump(pieces,pieceToMove,directMove,destCell)

    return nextMove


def makeMove(pieceToMove,destCell):
    """ Function: move
```

```
        => Description:
             Simulates the move for the selected piece to the
             desired destination directly (preferred: diagonal move)
        => Parameters:
             . Cell pieceToMove: cell representing the piece to
                 move
             . Cell destCell: cell representing the destination
        => Returns:
             . Cell directMove: cell representing the cell to move
                 to
    """

    # calculate direct move toward destination
    directMove = Cell(0,0)

    if pieceToMove.x > destCell.x:
        # move left
        directMove.x = pieceToMove.x-1
    elif pieceToMove.x < destCell.x:
        # move right
        directMove.x = pieceToMove.x+1
    else:
        # don't move
        directMove.x = pieceToMove.x

    # calculate vertical movement
    if pieceToMove.y > destCell.y:
        # move down
        directMove.y = pieceToMove.y-1
    elif pieceToMove.y < destCell.y:
        # move up
        directMove.y = pieceToMove.y+1
    else:
        # don't move
        directMove.y = pieceToMove.y

    return directMove


def determineJump(pieces,pieceToMove,move,destCell):
    """ Function: determineJump
         => Description:
             Determines if there is a piece to jump along the direct
             path to the destination and adjusts the next move to
             account for such a jump
             If there is a piece at the location you want to jump
                 to, then skip the move altogether
         => Parameters:
             . pieces: array of cells representing pieces
             . Int pieceToMove: index of the piece to move
             . move: cell representing the move to make
             . Cell destCell: cell representing the destination
         => Returns:
             . Cell newMove: the cell to move to
    """

    numPieces = len(pieces)
    newMove = {}
    newMove["destx"] = move.x
    newMove["desty"] = move.y
    newMove["piecex"] = pieces[pieceToMove].x
    newMove["piecey"] = pieces[pieceToMove].y
    newMove["jump"] = True
```

```
    for piece in range(0,numPieces):
        # check location of pieces to find if there is one to jump
        if (pieces[piece].x == move.x and pieces[piece].y == move.y):
            # calculate next potential move to simulate jump
            jumpMove = makeMove(move,destCell)

            # prevent L-shaped jumps: if x-displacement or y-displacement are
            # 2x the other, then that is an L shaped move: (up 2, over 1) or
            # (up 1, over 2)
            if (( abs(jumpMove.x - pieces[pieceToMove].x) ==
                abs(2 * (pieces[pieceToMove].y - jumpMove.y)) ) or
                ( abs(pieces[pieceToMove].y - jumpMove.y) ==
                 abs(2 * (jumpMove.x - pieces[pieceToMove].x)) )):

                newMove["destx"] = pieces[pieceToMove].x
                newMove["desty"] = pieces[pieceToMove].y
                newMove["jump"] = False

            # if there's a piece to jump
            # look at next cell to see if jump is possible
            for i in range(0,numPieces):

                if (pieces[i].x == jumpMove.x and pieces[i].y == jumpMove.y):
                    # pieces are in the way of jumping and moving so don't move
                    newMove["destx"] = pieces[pieceToMove].x
                    newMove["desty"] = pieces[pieceToMove].y
                    newMove["jump"] = False
                    break

            if (newMove["jump"]):
                newMove["destx"] = jumpMove.x
                newMove["desty"] = jumpMove.y

            break

    return newMove

def setPieces(pieces):
    """ Function: setPieces
         => Description:
             Creates cell objects for each piece x,y coordinate
         => Parameters:
             . pieces: array of piece coordinates
         => Returns:
             . piecesAsCells: array of cells with piece coords
    """

    piecesAsCells = []
    numPieces = len(pieces)

    for piece in range(0,numPieces):

        newPiece = Cell(pieces[piece]["x"],pieces[piece]["y"])
        piecesAsCells.append(newPiece)

    return piecesAsCells


def testSetPieces():


    stringJSON = json.loads(generateTestJSON(1))
```

```
    gameData = stringJSON["board"]

    pieces = gameData["pieces"]
    pieces = setPieces(pieces)

    if pieces[0].x == 1 and pieces[0].y == 1:
        return True
    else:
        return False


def setDestinations(dests):
    """ Function: setDestinations
         => Description:
             Creates cell objects for each destination x,y coord
         => Parameters:
             . dests: array of destination coords
         => Returns:
             . destsAsCells: array of cells with destination coords
    """

    destsAsCells = []
    numDests = len(dests)

    for dest in range(0,numDests):

        newDest = Cell(dests[dest]["x"],dests[dest]["y"])
        destsAsCells.append(newDest)

    return destsAsCells


def testSetDestinations():


    stringJSON = json.loads(generateTestJSON(2))
    gameData = stringJSON["board"]

    destRegion = gameData["destinations"]
    destRegion = setDestinations(destRegion)

    if destRegion[0].x == 1 and destRegion[0].y == 1:
        return True
    else:
        return False


def checkIfArrived(pieces,dests):
    """
     Function: checkIfArrived
         => Description:
             Checks if pieces are at final destinations and sets
             arrived to true for each piece at its destination
             and for each destination cell that is occupied
         => Parameters:
             . pieces: array of piece coords
             . dests: array of destination coords
         => Returns:
             . pieces: array of cells with piece coords w/updates
                 for any cell at its final destination
    """

    numDests = len(dests)
```

```
    numPieces = len(pieces)

    #check if any pieces at the back of the destination region
    for dest in range(0,numDests):

        for piece in range(0,numPieces):

            if (pieces[piece].x == dests[dest].x and
                pieces[piece].y == dests[dest].y):

                pieces[piece].arrived = True
                dests[dest].arrived = True
                break


def testCheckIfArrived():


    stringJSON = json.loads(generateTestJSON(3))
    gameData = stringJSON["board"]
    pieces = gameData["pieces"]
    pieces = setPieces(pieces)
    destRegion = gameData["destinations"]
    destRegion = setDestinations(destRegion)

    checkIfArrived(pieces,destRegion)

    if pieces[0].arrived and destRegion[0].arrived:
        # correctly identifies arrived pieces
        if pieces[1].arrived or destRegion[1].arrived:
            # incorrectly identifies pieces that are not arrived as though
            # they had actually arrived
            return False
        else:
            # correct
            return True
    else:
        # incorrect
        return False


def generateTestJSON(testNum):
    """
    Function: generateTestJSON
        => Description:
            Creates sample JSON input for testing the validity of AI moves
        => Parameters:
            . testNum: int determines which JSON to produce depending on test
        =>
    """

    stringJSON = None

    if testNum == 1:
        # test if pieces are set correctly
        stringJSON = {
            'board': {
                'pieces': [
                    {'y': 1,'x': 1},
                ],
            },
        }
    elif testNum == 2:
```

```
    # test if destinations are set correctly
    stringJSON = {
        'board': {
            'destinations': [
                {'y': 1,'x': 1},
            ],
        },
    }
elif testNum == 3:
    # test if checkIfArrived properly identifies pieces that have arrived
    # and does not misidentify any
    stringJSON = {
        'board': {
            'pieces': [
                {'y': 1,'x': 1},
                {'y': 2,'x': 1},
            ],
            'destinations': [
                {'y': 1,'x': 1},
                {'y': 1,'x': 0},
            ],

        },
    }
else:
    stringJSON = {
        "board": {
            "pieces": [
                {"y":7,"x":7,"team":0},
                {"y":9,"x":6,"team":0},
                {"y":10,"x":6,"team":0},
                {"y":11,"x":6,"team":0},
                {"y":7,"x":8,"team":0},
                {"y":3,"x":13,"team":0},
                {"y":2,"x":15,"team":0},
                {"y":3,"x":15,"team":0},
                {"y":2,"x":14,"team":0},
                {"y":3,"x":14,"team":0},
                {"y":4,"x":14,"team":0},
                {"y":11,"x":8,"team":0}
            ],
            "destinations": [
                {"y":0,"x":17,"team":-1},
                {"y":1,"x":17,"team":-1},
                {"y":2,"x":17,"team":-1},
                {"y":0,"x":16,"team":-1},
                {"y":1,"x":16,"team":-1},
                {"y":2,"x":16,"team":-1},
                {"y":0,"x":15,"team":-1},
                {"y":1,"x":15,"team":-1},
                {"y":2,"x":15,"team":-1}
            ],
            "boardSize": 18,
            "enemy": [
                {"y":8,"x":9,"team":1},
                {"y":9,"x":9,"team":1},
                {"y":10,"x":9,"team":1},
                {"y":11,"x":9,"team":1},
                {"y":8,"x":10,"team":1},
                {"y":9,"x":10,"team":1},
                {"y":10,"x":10,"team":1},
                {"y":11,"x":10,"team":1},
                {"y":8,"x":11,"team":1},
```

```
                    {"y":9,"x":11,"team":1},
                    {"y":10,"x":11,"team":1},
                    {"y":11,"x":11,"team":1}
                ],
                "currPiece": 9,
                "moveCount": 13,
            }
        }

    return json.dumps(stringJSON)


#------------------------------------------------------------------------------
#TESTS
# print "1) Testing if setPieces() returns the correct result:"
# print ">> " + str(testSetPieces())
# print
# print "2) Testing if setDestinations() returns the correct result:"
# print ">> " + str(testSetDestinations())
# print
# print """3) Testing if checkIfArrived() correctly identifies when pieces reach
#     their destinations for both the pieces and the destinations:"""
# print ">> " + str(testCheckIfArrived())
#------------------------------------------------------------------------------
# GET DATA
postData = cgi.FieldStorage()
gameData = ast.literal_eval(postData.getvalue('board'))

# set pieces as cells
pieces = gameData["pieces"]
pieces = setPieces(pieces)

# set destinations as cells
destRegion = gameData["destinations"]
destRegion = setDestinations(destRegion)

# get the number of the piece to move
pieceToMove = gameData["currPiece"]

# check if pieces are in destinations
checkIfArrived(pieces,destRegion)

# make next move
nextMove = getMove(pieces,destRegion,pieceToMove)

# return JSON of next move
nextMove = {
    'from': {'x': nextMove['piecex'], 'y': nextMove['piecey']},
    'to': [{'x': nextMove['destx'], 'y': nextMove['desty']}]
    }

# return the next move
print 'Content-Type: application/json\n'
print
print str(json.dumps(nextMove))
```

halmaAI_smart.py

```
#!/usr/local/bin/python
# -*- coding: UTF-8 -*-

# AUTHOR: NATHAN MOORE
# URL: http://lyle.smu.edu/~ndmoore/cgi-bin/halmaAI_smart.py
# Takes enemy pieces into account

import cgi
import cgitb
import json
import ast

cgitb.enable()


class Cell:
    """ Class: Cell
    => Description:
        Intended to represent pieces and destinations
    => Class variables:
        . int x: coordinate on game board
        . int y: coordinate on game board
        . bool arrived: true if the piece has arrived at destination
        * Upper left corner of board is 0,0
    """

    def __init__(self,x,y):
        self.x = x
        self.y = y
        self.arrived = False


    @property
    def x(self):
        return self.__x


    @property
    def y(self):
        return self.__y


    @property
    def arrived(self):
        return self.__arrived


    @x.setter
    def x(self,x):
        self.__x = x


    @y.setter
    def y(self,y):
        self.__y = y


    @arrived.setter
    def arrived(self,arrived):
        self.__arrived = arrived
```

```
def getMove(pieces,destRegion,enemy,pieceToMove):
    """ Function: getMove
        => Description:
            Calculates the next move the AI should make
        => Parameters:
            . pieces: list of cells representing the pieces on
                the board
            . destRegion: list of cells representing the
                destination area
            . enemy: list of enemy pieces
            . pieceToMove: the piece to move next
        => Returns:
            . nextMove: list with the location of the piece to
                move and the location to move that piece to
    """

    nextMove = {}
    # Select piece to move and destination to move to:
    # Make sure that the piece isn't already in the destination
    #     region
    nextMove["destx"] = pieces[pieceToMove].x
    nextMove["desty"] = pieces[pieceToMove].y
    nextMove["piecex"] = pieces[pieceToMove].x
    nextMove["piecey"] = pieces[pieceToMove].y

    if pieceToMove > 8:
        destCell = destRegion[pieceToMove-3]
    else:
        destCell = destRegion[pieceToMove]

    if (pieces[pieceToMove].arrived and destCell.arrived):
        return nextMove
    elif (not pieces[pieceToMove].arrived) and destCell.arrived:
        numDests = len(destRegion)
        for dest in range(0,numDests):
            if not destRegion[dest].arrived:
                destCell = destRegion[dest]
                break

    # if the piece to move has not arrived at its destination
    # simulate the next move to take
    directMove = makeMove(pieces[pieceToMove],destCell)
    # Modify the move if a jump is available
    #  Check if a piece is already in the location you want to
    #     move to
    #  if yes: check next location along path
    #      if yet another piece: skip move
    #      if not: move to that second location (jump)
    #  if no: move to location
    nextMove = determineJump(pieces,pieceToMove,directMove,destCell,enemy)

    return nextMove


def makeMove(pieceToMove,destCell):
    """ Function: move
        => Description:
             Simulates the move for the selected piece to the
             desired destination directly (preferred: diagonal move)
```

```
        => Parameters:
             . Cell pieceToMove: cell representing the piece to
                 move
             . Cell destCell: cell representing the destination
        => Returns:
             . Cell directMove: cell representing the cell to move
                 to
    """

    # calculate direct move toward destination
    directMove = Cell(0,0)

    if pieceToMove.x > destCell.x:
        # move left
        directMove.x = pieceToMove.x-1
    elif pieceToMove.x < destCell.x:
        # move right
        directMove.x = pieceToMove.x+1
    else:
        # don't move
        directMove.x = pieceToMove.x

    # calculate vertical movement
    if pieceToMove.y > destCell.y:
        # move down
        directMove.y = pieceToMove.y-1
    elif pieceToMove.y < destCell.y:
        # move up
        directMove.y = pieceToMove.y+1
    else:
        # don't move
        directMove.y = pieceToMove.y

    return directMove


def determineJump(pieces,pieceToMove,move,destCell,enemy):
    """ Function: determineJump
         => Description:
             Determines if there is a piece to jump along the direct
             path to the destination and adjusts the next move to
             account for such a jump
             If there is a piece at the location you want to jump
                 to, then skip the move altogether
         => Parameters:
             . pieces: array of cells representing pieces
             . Int pieceToMove: index of the piece to move
             . move: cell representing the move to make
             . Cell destCell: cell representing the destination
             . enemy: array of enemy pieces
         => Returns:
             . Cell newMove: the cell to move to
    """

    numPieces = len(pieces)
    newMove = {}
    newMove["destx"] = move.x
    newMove["desty"] = move.y
    newMove["piecex"] = pieces[pieceToMove].x
    newMove["piecey"] = pieces[pieceToMove].y
    newMove["jump"] = True

    for piece in range(0,numPieces):
```

```
        # check location of pieces to find if there is one to jump
        if ((pieces[piece].x == move.x and pieces[piece].y == move.y)
             or
             (enemy[piece].x == move.x and enemy[piece].y == move.y)):
            # calculate next potential move to simulate jump
            jumpMove = makeMove(move,destCell)

            # prevent L-shaped jumps: if x-displacement or y-displacement are
            # 2x the other, then that is an L shaped move
            if (( abs(jumpMove.x - pieces[pieceToMove].x) ==
                abs(2 * (pieces[pieceToMove].y - jumpMove.y)) ) or
                ( abs(pieces[pieceToMove].y - jumpMove.y) ==
                 abs(2 * (jumpMove.x - pieces[pieceToMove].x)) )):

                newMove["destx"] = pieces[pieceToMove].x
                newMove["desty"] = pieces[pieceToMove].y
                newMove["jump"] = False

            # if there's a piece to jump
            # look at next cell to see if jump is possible
            for i in range(0,numPieces):

                if ((pieces[i].x == jumpMove.x and pieces[i].y == jumpMove.y)
                    or
                    (enemy[i].x == jumpMove.x and enemy[i].y == jumpMove.y)):
                    # pieces are in the way of jumping and moving so don't move
                    newMove["destx"] = pieces[pieceToMove].x
                    newMove["desty"] = pieces[pieceToMove].y
                    newMove["jump"] = False
                    break

            if (newMove["jump"]):
                newMove["destx"] = jumpMove.x
                newMove["desty"] = jumpMove.y

            break

    return newMove

def setPieces(pieces):
    """ Function: setPieces
         => Description:
             Creates cell objects for each piece x,y coordinate
         => Parameters:
             . pieces: array of piece coordinates
         => Returns:
             . piecesAsCells: array of cells with piece coords
    """

    piecesAsCells = []
    numPieces = len(pieces)

    for piece in range(0,numPieces):

        newPiece = Cell(pieces[piece]["x"],pieces[piece]["y"])
        piecesAsCells.append(newPiece)

    return piecesAsCells


def testSetPieces():
```

```
    stringJSON = json.loads(generateTestJSON(1))
    gameData = stringJSON["board"]

    pieces = gameData["pieces"]
    pieces = setPieces(pieces)

    if pieces[0].x == 1 and pieces[0].y == 1:
        return True
    else:
        return False


def setDestinations(dests):
    """ Function: setDestinations
         => Description:
             Creates cell objects for each destination x,y coord
         => Parameters:
             . dests: array of destination coords
         => Returns:
             . destsAsCells: array of cells with destination coords
    """

    destsAsCells = []
    numDests = len(dests)

    for dest in range(0,numDests):

        newDest = Cell(dests[dest]["x"],dests[dest]["y"])
        destsAsCells.append(newDest)

    return destsAsCells


def testSetDestinations():


    stringJSON = json.loads(generateTestJSON(2))
    gameData = stringJSON["board"]

    destRegion = gameData["destinations"]
    destRegion = setDestinations(destRegion)

    if destRegion[0].x == 1 and destRegion[0].y == 1:
        return True
    else:
        return False


def checkIfArrived(pieces,dests):
    """
     Function: checkIfArrived
         => Description:
             Checks if pieces are at final destinations and sets
             arrived to true for each piece at its destination
             and for each destination cell that is occupied
         => Parameters:
             . pieces: array of piece coords
             . dests: array of destination coords
         => Returns:
             . pieces: array of cells with piece coords w/updates
                 for any cell at its final destination
    """
```

```
    numDests = len(dests)
    numPieces = len(pieces)

    # check if any pieces at the back of the destination region
    for dest in range(0,numDests):

        for piece in range(0,numPieces):

            if (pieces[piece].x == dests[dest].x and
                pieces[piece].y == dests[dest].y):

                pieces[piece].arrived = True
                dests[dest].arrived = True
                break


def testCheckIfArrived():


    stringJSON = json.loads(generateTestJSON(3))
    gameData = stringJSON["board"]
    pieces = gameData["pieces"]
    pieces = setPieces(pieces)
    destRegion = gameData["destinations"]
    destRegion = setDestinations(destRegion)

    checkIfArrived(pieces,destRegion)

    if pieces[0].arrived and destRegion[0].arrived:
        # correctly identifies arrived pieces
        if pieces[1].arrived or destRegion[1].arrived:
            # incorrectly identifies pieces that are not arrived as though
            # they had actually arrived
            return False
        else:
            # correct
            return True
    else:
        # incorrect
        return False


def generateTestJSON(testNum):
    """
    Function: generateTestJSON
        => Description:
            Creates sample JSON input for testing the validity of AI moves
        => Parameters:
            . testNum: int determines which JSON to produce depending on test
        =>
    """

    stringJSON = None

    if testNum == 1:
        # test if pieces are set correctly
        stringJSON = {
            'board': {
                'pieces': [
                    {'y': 1,'x': 1},
                ],
            },
        }
```

```
elif testNum == 2:
    # test if destinations are set correctly
    stringJSON = {
        'board': {
            'destinations': [
                {'y': 1,'x': 1},
            ],
        },
    }
elif testNum == 3:
    # test if checkIfArrived properly identifies pieces that have arrived
    # and does not misidentify any
    stringJSON = {
        'board': {
            'pieces': [
                {'y': 1,'x': 1},
                {'y': 2,'x': 1},
            ],
            'destinations': [
                {'y': 1,'x': 1},
                {'y': 1,'x': 0},
            ],

        },
    }
else:
    stringJSON = {
        "board": {
            "pieces": [
                {"y":7,"x":7,"team":0},
                {"y":9,"x":6,"team":0},
                {"y":10,"x":6,"team":0},
                {"y":11,"x":6,"team":0},
                {"y":7,"x":8,"team":0},
                {"y":3,"x":13,"team":0},
                {"y":2,"x":15,"team":0},
                {"y":3,"x":15,"team":0},
                {"y":2,"x":14,"team":0},
                {"y":3,"x":14,"team":0},
                {"y":4,"x":14,"team":0},
                {"y":11,"x":8,"team":0}
            ],
            "destinations": [
                {"y":0,"x":17,"team":-1},
                {"y":1,"x":17,"team":-1},
                {"y":2,"x":17,"team":-1},
                {"y":0,"x":16,"team":-1},
                {"y":1,"x":16,"team":-1},
                {"y":2,"x":16,"team":-1},
                {"y":0,"x":15,"team":-1},
                {"y":1,"x":15,"team":-1},
                {"y":2,"x":15,"team":-1}
            ],
            "boardSize": 18,
            "enemy": [
                {"y":8,"x":9,"team":1},
                {"y":9,"x":9,"team":1},
                {"y":10,"x":9,"team":1},
                {"y":11,"x":9,"team":1},
                {"y":8,"x":10,"team":1},
                {"y":9,"x":10,"team":1},
                {"y":10,"x":10,"team":1},
                {"y":11,"x":10,"team":1},
```

```
                    {"y":8,"x":11,"team":1},
                    {"y":9,"x":11,"team":1},
                    {"y":10,"x":11,"team":1},
                    {"y":11,"x":11,"team":1}
                ],
                "currPiece": 9,
                "moveCount": 13,
            }
        }

    return json.dumps(stringJSON)


#--------------------------------------------------------------------------------
# TESTS
# print "1) Testing if setPieces() returns the correct result:"
# print ">> " + str(testSetPieces())
# print
# print "2) Testing if setDestinations() returns the correct result:"
# print ">> " + str(testSetDestinations())
# print
# print """3) Testing if checkIfArrived() correctly identifies when pieces reach
#     their destinations for both the pieces and the destinations:"""
# print ">> " + str(testCheckIfArrived())
#--------------------------------------------------------------------------------
# GET DATA
postData = cgi.FieldStorage()
gameData = ast.literal_eval(postData.getvalue('board'))

# set pieces as cells
pieces = gameData["pieces"]
pieces = setPieces(pieces)

# set enemy pieces as cells
enemy = gameData["enemy"]
enemy = setPieces(enemy)

# set destinations as cells
destRegion = gameData["destinations"]
destRegion = setDestinations(destRegion)

# get the number of the piece to move
pieceToMove = gameData["currPiece"]

# check if pieces are in destinations
checkIfArrived(pieces,destRegion)

# make next move
nextMove = getMove(pieces,destRegion,enemy,pieceToMove)

# return JSON of next move
nextMove = {
    'from': {'x': nextMove['piecex'], 'y': nextMove['piecey']},
    'to': [{'x': nextMove['destx'], 'y': nextMove['desty']}]
    }

# return the next move
print 'Content-Type: application/json\n'
print
print str(json.dumps(nextMove))
```

halma.html

```
<!DOCTYPE html>
<html lang="en">
    <!-- v 1.2 give choice for POST with or without parm (board=jsonstr)
    -->
    <!-- URL: http://lyle.smu.edu/~ndmoore/Halma 3.0/halma.html -->
    <head>
        <meta charset="UTF-8"/>
        <meta name="robots" content="noindex"/>
        <title>Halma UI</title>
        <script src="jquery-2.1.1.js"></script>
        <script src="halma.js"></script>
    </head>
    <body>
        <div id ="initialization">
            <h1>Halma 3.0a</h1>
            <p>Version 3.0 incorporates our python AI and uses a timer.</p>
            <p>Version 3.0 also is a multi-move Halma game, allowing each team
                to move all its pieces at once. </p>
            <ul>
                <li>Rejects a move request if a team tries to move to a
                    location occupied by either team.</li>
                <li>Displays an ALERT if the data from the AI is NOT JSON
                    and displays the  data</li>
                <li>Displays an ALERT if an AI tries to move a piece not on
                    its team</li>
                <li>Displays an ALERT if an AI tries to move to a square
                    already occupied</li>
                <li>Displays all DATA sent to each AI and displays all DATA
                    received from the AI via console.log which can viewed by
                    getting your browser to display the Javascript log
                    (how to do this varies from browser to browser)</li>
            </ul>

            <form id="initForm">

                <!--  radio buttons: replaced with test for .py extension
                <p>
                  <input type="radio" name="POSToption" id="NoParm"
                    value="NoParm">Send only JSON (No Parameter)
                  <input type="radio" name="POSToption" value="YesParm">
                    Send JSON with Parameter board=
                </p>
                -->

                <label>Team1 URL:</label>
                <input style="width: 400px" type="text" id="team1Url"
                    value="http://lyle.smu.edu/~ndmoore/cgi-bin/halmaAI_dumb.py"
                    required />
                <label>   Team1 Name:</label>
                <input style="width: 100px" type="text" id="team1Name"
                    value="bozos" required /><br/>


                <label>Team2 URL:</label>
                <input style="width: 400px" type="text" id="team2Url"
                    value="http://lyle.smu.edu/~ndmoore/cgi-bin/halmaAI_smart.py"
                    required />
                <label>   Team2 Name:</label>
                <input style="width: 100px" type="text" id="team2Name"
                    value="smarties" required /><br/>
```

```
                <input type="button" id="playGame" value="Play!"
                    onclick="checkInputs()" />

                <p style="color: red">*Your AI must be on the same server
                    as this UI<br/>
                    Begin your URL with http:// <br/>
                    JSON with boardSize, pieces and destinations will be
                    sent via HTTP POST<br/>

                </p>
                <p style="color: blue">The JSON from the HalmaUI is sent via
                    HTTP POST.
                    To obtain the JSON string with PHP use:
                        $jsonStr = file_get_contents("php://input");
                </p>
                <br />

                <input type="submit" id="hiddenSubmit" hidden />
                <br/>

            </form>
        </div>
        <div id ="game">
            <h4>Halma UI v3.0  <span id="winnerCircle"> </span></h4>

            <p><label>HalmaAI <span id="AITeamName">
                </span>: JSON Move Request= </label>
            <label id="responseString"  style="width:400">
                -- json shown here --</label>
            </p>
            <p id="moves">Moves: <span id="movecount">0</span>
                <input type="button" id="startGame" value="Start Game"
                    onclick="startGame()" />

                <input type="button" id="restartGame" value="New Game"
                    onclick="refreshGame()" />
            </p>
        </div>

    </body>
</html>
```

halma.js

```
/* V 3.0

 */

//change to 0 to use with Initialization form
var kBoardWidth = 18;
//change to 0 to use with Initialization form
var kBoardHeight = 18;
var kPieceWidth = 20;
var kPieceHeight = 20;
// change to 0 to use with initialization form
var kPixelWidth = 1 + (kBoardWidth * kPieceWidth);
// change to 0 to use with initialization form
var kPixelHeight = 1 + (kBoardHeight * kPieceHeight);
//change to "" to use with initialization form
var destinationCorner = "upperRight";
```

```
//change to "" to user with initialization form
var piecesCorner = "lowerLeft";

var gCanvasElement;
var gDrawingContext;
var gPattern;

// so we can push and append
var gPieces = [];
var gDestinations;
var gNumDests;
var gSelectedPieceIndex;
var gSelectedPieceHasMoved;
var gMoveCount = 0;
var gMoveCountElem;
var gGameInProgress;


// v 2.0
// filled in newGame - array Of Teams
var gTeamList = [];
var gTurnCount = 0;
var gNumTeams  = 2;

// ?? // wait, really? There's a comment that is a double ? in the code??
var url = "";

function Team(teamIdx, startArea, destArea, color) {
    this.teamIdx = teamIdx;
    this.startArea = startArea;
    this.destArea  = destArea;
    this.color     = color;
    this.teamPieces = [];
    this.teamDestinations = [];
    this.teamUrl    = '';
    // changed if python AI ends with py
    this.sendPostNoParm = true;
    // comes in from HTML
    this.name = "nonames";
    this.badMoveCount = 0;
    this.numberJumps  = 0;
    this.maxJump      = 0;
}

function Cell(y, x) {
    this.y = y;
    this.x = x;
    // none
    this.team = -1;
    this.toString = function() {
        return  JSON.stringify(this);
    };
}

function isFreeCell(c1, gPiecesArr) {
    for(var i=0; i<gPiecesArr.length; i++) {
        if(c1.x === gPiecesArr[i].x &&
           c1.y === gPiecesArr[i].y)
        return false;
    }
    return true;
}
```

```
function isCellOnTeam(cell, teamArr) {

    for(var i=0; i<teamArr.length; i++) {
        if(cell.x === teamArr[i].x &&
           cell.y === teamArr[i].y)
        return true;
    }
    return false;
}

function isThereAPieceBetween(cell1, cell2) {
    /* note: assumes cell1 and cell2 are 2 squares away
     either vertically, horizontally, or diagonally */
    var rowBetween = (cell1.y + cell2.y) / 2;
    var columnBetween = (cell1.x + cell2.x) / 2;
    for (var i = 0; i < gPieces.length; i++) {
        if ((gPieces[i].y === rowBetween) &&
                (gPieces[i].x === columnBetween)) {
            return true;
        }
    }
    return false;
}

function isTheGameOver() {
    for (var i = 0; i < gPieces.length; i++) {
        if (gPieces[i].y > 2) {
            return false;
        }
        if (gPieces[i].x < (kBoardWidth - 3)) {
            return false;
        }
    }
    return true;
}

function drawBoard() {
    if (gGameInProgress && isTheGameOver()) {
        endGame();
    }

    gDrawingContext.clearRect(0, 0, kPixelWidth, kPixelHeight);

    gDrawingContext.beginPath();

    /* vertical lines */
    for (var x = 0; x <= kPixelWidth; x += kPieceWidth) {
        gDrawingContext.moveTo(0.5 + x, 0);
        gDrawingContext.lineTo(0.5 + x, kPixelHeight);
    }

    /* horizontal lines */
    for (var y = 0; y <= kPixelHeight; y += kPieceHeight) {
        gDrawingContext.moveTo(0, 0.5 + y);
        gDrawingContext.lineTo(kPixelWidth, 0.5 + y);
    }

    /* draw it! */
    gDrawingContext.strokeStyle = "#ccc";
    gDrawingContext.stroke();

    /* draw all pieces  */
    for (var i = 0; i < gPieces.length; i++) {
```

```
        drawPiece(gPieces[i], (i === gSelectedPieceIndex));
    }

    gMoveCountElem.innerHTML = gMoveCount;

    saveGameState();
}

function drawDotOnActivePiece(p) {
    var column = p.x;
    var row    = p.y;
    var x = (column * kPieceWidth) + (kPieceWidth / 2);
    var y = (row * kPieceHeight) + (kPieceHeight / 2);
    var radius = (kPieceWidth / 2) - (kPieceWidth / 10);

     // try interior colored circle
    gDrawingContext.beginPath();
    gDrawingContext.arc(x, y, radius/2, 0, Math.PI * 2, false);
    gDrawingContext.closePath();
    gDrawingContext.strokeStyle = "#000";
    gDrawingContext.stroke();
}

function drawPiece(p, selected) {
    var column = p.x;
    var row    = p.y;
    //console.log("draw Piece at Row: " + row + ", Col: " + column);
    var x = (column * kPieceWidth) + (kPieceWidth / 2);
    var y = (row * kPieceHeight) + (kPieceHeight / 2);
    var radius = (kPieceWidth / 2) - (kPieceWidth / 10);
    gDrawingContext.beginPath();
    gDrawingContext.arc(x, y, radius, 0, Math.PI * 2, false);
    gDrawingContext.closePath();
    gDrawingContext.strokeStyle = "#000";
    gDrawingContext.stroke();

    // find the team piece p is part of and use team color
    //
    // version 1.2
    fillColor = gTeamList[p.team].color;

    // fill in team color
    gDrawingContext.fillStyle = fillColor;
    gDrawingContext.fill();

    // draw current piece black
    if(selected) {
        gDrawingContext.fillStyle = "#000";
        gDrawingContext.fill();

        // try interior colored circle
        gDrawingContext.beginPath();
        gDrawingContext.arc(x, y, radius/2, 0, Math.PI * 2, false);
        gDrawingContext.closePath();
        gDrawingContext.strokeStyle = "#000";
        gDrawingContext.stroke();
        // fill in team color
        gDrawingContext.fillStyle = fillColor;
        gDrawingContext.fill();
    }
```

```
    // todo: draw cells in destination area differently
    // assume destination areas are upperRight and upperLeft
    if ((column < 3 && row < 3) ||
        (column >= kBoardWidth - 3 && row < 3)  )   {
        gDrawingContext.fillStyle = "#00ff00";
        gDrawingContext.fill();

        // draw inner circle? //again, what is up with this comment?
    }

}

if (typeof resumeGame !== "function") {
    saveGameState = function() {
        return false;
    };
    resumeGame = function() {
        return false;
    };
}

function newGame() {
    // set up team that knows origin("lowerLeft or lowerRight,
    //                               destintion, color, url or localJSopponent
    // based on that each team gets its pieces
    // which get appended via push to gPieces
    // global gTeamList holds Team instances

    // set up teams
    var team0 = new Team(0, "lowerLeft", "upperRight", "#CC0099");
    var team1 = new Team(1, "lowerRight", "upperLeft", "#006699");
    team1.teamPieces.reverse();
    gTeamList[0] = team0;
    gTeamList[1] = team1;

    var url1 = document.getElementById("team1Url").value;
    var url2 = document.getElementById("team2Url").value;

    // TEAMS : URLs SET - if AI is python; send json with board=  as parm
    gTeamList[0].url = url1;
    if (endsWith(url1, "py")) gTeamList[0].sendPostNoParm = false;
    console.log("team send parm with POST = " + gTeamList[0].sendPostNoParm);
    gTeamList[0].name = document.getElementById("team1Name").value;

    gTeamList[1].url = url2;  //could be localjs
    if (endsWith(url2, "py")) gTeamList[1].sendPostNoParm = false;
    gTeamList[1].name = document.getElementById("team2Name").value;

    for ( i=0; i<gTeamList.length; i++) {

        // setUpTeamPieces
        // setUpTeamDestinations
        // return array filled with cells from a corner
        teamPieceArr = setUpTeamPieces(gTeamList[i].startArea);
        teamDestArr  = setUpTeamDestinations(gTeamList[i].destArea);
        gTeamList[i].teamPieces      = teamPieceArr;
        gTeamList[i].teamDestinations = teamDestArr;

        // add team pieces to gPieces -- game engine sees just pieces
        for (k=0; k<teamPieceArr.length; k++) {
            // team pieces now know their team (0 or 1)
            teamPieceArr[k].team = i; // i is teamIdx
            gPieces.push(teamPieceArr[k]);
```

```
        }

    }

    //gNumDests = gDestinations.length;  //??
    gSelectedPieceIndex = -1;
    gSelectedPieceHasMoved = false;
    gMoveCount = 0;
    gGameInProgress = true;
    //alert("ready to draw gPieces length = " + gPieces.length);
    drawBoard();
}

// returns array of pieces (Cells) for a team
// the Cells should be added to gPieces

function setUpTeamPieces(piecesCorner) {

    if (piecesCorner === "lowerLeft") {
        teamPieces = [
            new Cell(kBoardHeight - 4, 0),
            new Cell(kBoardHeight - 3, 0),
            new Cell(kBoardHeight - 2, 0),
            new Cell(kBoardHeight - 1, 0),
            new Cell(kBoardHeight - 4, 1),
            new Cell(kBoardHeight - 3, 1),
            new Cell(kBoardHeight - 2, 1),
            new Cell(kBoardHeight - 1, 1),
            new Cell(kBoardHeight - 4, 2),
            new Cell(kBoardHeight - 3, 2),
            new Cell(kBoardHeight - 2, 2),
            new Cell(kBoardHeight - 1, 2)];
    }

    else if (piecesCorner === "lowerRight") {
        teamPieces = [
            new Cell(kBoardHeight - 4, kBoardWidth - 3),
            new Cell(kBoardHeight - 3, kBoardWidth - 3),
            new Cell(kBoardHeight - 2, kBoardWidth - 3),
            new Cell(kBoardHeight - 1, kBoardWidth - 3),
            new Cell(kBoardHeight - 4, kBoardWidth - 2),
            new Cell(kBoardHeight - 3, kBoardWidth - 2),
            new Cell(kBoardHeight - 2, kBoardWidth - 2),
            new Cell(kBoardHeight - 1, kBoardWidth - 2),
            new Cell(kBoardHeight - 4, kBoardWidth - 1),
            new Cell(kBoardHeight - 3, kBoardWidth - 1),
            new Cell(kBoardHeight - 2, kBoardWidth - 1),
            new Cell(kBoardHeight - 1, kBoardWidth - 1)];
    }
    else alert("setUpTeamPieces does not understand: " + piecesCorner);

    return teamPieces;
}
  function setUpTeamDestinations(destinationCorner) {
    if (destinationCorner === "lowerLeft") {
        teamDestinations = [
            new Cell(kBoardHeight - 1, 0),
            new Cell(kBoardHeight - 2, 0),
            new Cell(kBoardHeight - 3, 0),

            new Cell(kBoardHeight - 1, 1),
            new Cell(kBoardHeight - 2, 1),
            new Cell(kBoardHeight - 3, 1),
```

```
            new Cell(kBoardHeight - 1, 2),
            new Cell(kBoardHeight - 2, 2),
            new Cell(kBoardHeight - 3, 2)

        ];
    }
    else if (destinationCorner === "upperLeft") {
        teamDestinations = [new Cell(0, 0),
            new Cell(1, 0),
            new Cell(2, 0),
            new Cell(0, 1),
            new Cell(1, 1),
            new Cell(2, 1),
            new Cell(0, 2),
            new Cell(1, 2),
            new Cell(2, 2)];
    }
    else if (destinationCorner === "upperRight") {
        teamDestinations = [new Cell(0, kBoardWidth - 1),
            new Cell(1, kBoardWidth - 1),
            new Cell(2, kBoardWidth - 1),
            new Cell(0, kBoardWidth - 2),
            new Cell(1, kBoardWidth - 2),
            new Cell(2, kBoardWidth - 2),
            new Cell(0, kBoardWidth - 3),
            new Cell(1, kBoardWidth - 3),
            new Cell(2, kBoardWidth - 3)];
    }
    else if (destinationCorner === "lowerRight") {
        teamDestinations = [new Cell(kBoardHeight - 1, kBoardWidth - 1),
            new Cell(kBoardHeight - 2, kBoardWidth - 1),
            new Cell(kBoardHeight - 3, kBoardWidth - 1),
            new Cell(kBoardHeight - 1, kBoardWidth - 2),
            new Cell(kBoardHeight - 2, kBoardWidth - 2),
            new Cell(kBoardHeight - 3, kBoardWidth - 2),
            new Cell(kBoardHeight - 1, kBoardWidth - 3),
            new Cell(kBoardHeight - 2, kBoardWidth - 3),
            new Cell(kBoardHeight - 3, kBoardWidth - 3)];
    }
    else alert("setUpDestinations does not understand: " + destinationCorner);
    return teamDestinations;
}

function endGame() {
    gSelectedPieceIndex = -1;
    gGameInProgress = false;
    $('#restartGame').show();
}

function refreshGame() {
    location.reload();
}

function initGame(canvasElement, moveCountElement) {
    /*************************************************
     * Uncomment following code to use initialization form
     *************************************************/
//    var boardSize = document.getElementById("boardSize").value;
//    kBoardHeight = boardSize;
//    kBoardWidth = boardSize;
//    kPixelWidth = 1 + (kBoardWidth * kPieceWidth);
//    kPixelHeight = 1 + (kBoardHeight * kPieceHeight);
```

```
//
//    destinationCorner = $('input:radio[name=destCorner]:checked').val();
//    piecesCorner = $('input:radio[name=pieceCorner]:checked').val();



    if (!canvasElement) {
        canvasElement = document.createElement("canvas");
        canvasElement.id = "halma_canvas";
        document.body.appendChild(canvasElement);
    }
    if (!moveCountElement) {
        moveCountElement = document.createElement("p");
        document.body.appendChild(moveCountElement);
    }
    gCanvasElement = canvasElement;
    gCanvasElement.width = kPixelWidth;
    gCanvasElement.height = kPixelHeight;
    gMoveCountElem = moveCountElement;
    gDrawingContext = gCanvasElement.getContext("2d");
    if (!resumeGame()) {
        newGame();
    }
    $('#initialization').hide();
    $('#game').show();
}

function startGame() {
    //this is not the function that will repeatedly call makeMove so that the
    //game plays on its own
    setInterval(function(){makeMove()},3000);
    $('#startGame').hide();

}
//
// Called when MOVE is called by interval
// todo: add turns
function makeMove() {

    if (isGameOver() ) return;

    gMoveCount++;
    document.getElementById("movecount").innerHTML = gMoveCount;

    var currentTeam = gTurnCount++ % gNumTeams;

    for(var pieceNum = 0; pieceNum < 12; pieceNum++){

        var resp;  // response from AJAX call:  if Python send with parm
        if (gTeamList[currentTeam].sendPostNoParm) {
            resp = makeAjaxPostMoveRequestNoParm(currentTeam, pieceNum);
        }
        else resp = makeAjaxPostMoveRequestWithParm(currentTeam, pieceNum);

         // AJAX CALL : get move for current team - as text (10/25/14)
        //  convert text to json to show any errors in AI output
        var move;
        try {
            move = JSON.parse(resp);
        }
        catch (e) {
            alert("Data from AI at:" + gTeamList[currentTeam].url +
                    "is NOT JSON! Output received was:" + resp);
```

```
    return;
}

// debug
console.log("AI move Request: " + JSON.stringify(move));

//fc: display incoming json and Team Name
var teamSpan         = document.getElementById("AITeamName");
teamSpan.innerHTML   = gTeamList[currentTeam].name;
teamSpan.style.color = gTeamList[currentTeam].color;
//might need to change this to instead append the current move and
//clear it when the next player goes
document.getElementById("responseString").innerHTML =
    JSON.stringify(move);

 // is piece to move on current team? if not exit
 var locPiece = move.from;
 var currPieceLoc = new Cell(locPiece.y, locPiece.x);

 if(!isCellOnTeam(currPieceLoc, gTeamList[currentTeam].teamPieces)) {
     //update bad move count
     alert("BAD MOVE Request: Requested Piece to Move not Valid");
     break;
 }

    var movePieceLocs = move.to;

    // create moves - array of Cells where AI wants to move
    var moves = [];
    for(var i = 0; i < movePieceLocs.length; i++) {
        moves.push(new Cell(movePieceLocs[i].y, movePieceLocs[i].x));
    }


   // 10.29.14: need temp array since can't pass moves to function
   // w/out values changing
   var workingMovesArr = [];
   for (var i = 0; i < moves.length; i++) {
         workingMovesArr[i] = moves[i]; // copy over so can pass it
   }

   // check that the move sequence requested is valid
   if (!isValidMoveRequest(currPieceLoc, workingMovesArr, gPieces) ){
        alert("Illegal Move request from AI " +
              gTeamList[currentTeam].name + " " + JSON.stringify(move)
              + ". Penalty will be loss of move" );
        break;  // no need to proceed
   }


    // if moves array is ok, reset the piece we are moving
    // find ref to the actual piece in the teamPieces array

    var currentPieceIdx = -1;
    for(var i=0; i< gTeamList[currentTeam].teamPieces.length; i++ ){
        if (currPieceLoc.x ===
                gTeamList[currentTeam].teamPieces[i].x &&
                currPieceLoc.y ===
                gTeamList[currentTeam].teamPieces[i].y) {
            currentPieceIdx = i;
            break;
        }
    };
```

```
            // we have a problem if we can't find current piece already found
            if (currentPieceIdx === -1) {
                alert("SYSTEM ERROR 1: CUrrent Piece IDX not FOUND!??");
                break;
            }

            // update current Piece position to last entry in move request list
            gTeamList[currentTeam].teamPieces[currentPieceIdx].y =
                moves[moves.length - 1].y;
            gTeamList[currentTeam].teamPieces[currentPieceIdx].x =
                moves[moves.length - 1].x;
        }
        // Draw the board and all the pieces in their team colors
        drawBoard();

        // draw the active piece with a black dot
        drawDotOnActivePiece(gTeamList[currentTeam].teamPieces[currentPieceIdx]);

        // currPieceLoc now holds position where piece was before move-
        // draw small dot to indicate original position
            var x = (currPieceLoc.x * kPieceWidth) + (kPieceWidth / 2);
            var y = (currPieceLoc.y * kPieceHeight) + (kPieceHeight / 2);
            var radius = (kPieceWidth / 2) - (kPieceWidth / 10);
            gDrawingContext.beginPath();
            gDrawingContext.arc(x, y, radius / 3, 0, Math.PI * 2, false);
            gDrawingContext.closePath();
            gDrawingContext.strokeStyle = "#000";
            gDrawingContext.stroke();
            gDrawingContext.fillStyle = "#f00";
            gDrawingContext.fill();


            // draw breadcrumbs ..all except last move as dot -
            for(var i = 0; i < moves.length - 1; i++) {
                var x = (moves[i].x * kPieceWidth) + (kPieceWidth / 2);
                var y = (moves[i].y * kPieceHeight) + (kPieceHeight / 2);
                console.log ("breadcrumb at"  + x + "," + y);
                var radius = (kPieceWidth / 2) - (kPieceWidth / 10);
                gDrawingContext.beginPath();
                gDrawingContext.arc(x, y, radius/3, 0, Math.PI * 2, false);
                gDrawingContext.closePath();
                gDrawingContext.strokeStyle = "#000";
                gDrawingContext.stroke();
                gDrawingContext.fillStyle = "#f00";
                gDrawingContext.fill();
            }



    }

function makeAjaxPostMoveRequestNoParm(teamIdx, pieceToMove) {

    var move = "No move received. See Alerts.";  // overwrite w/ HTTP response

      $.ajax({
        type: 'POST',
        url: gTeamList[teamIdx].url,
        dataType: "text",
        async: false,
        data: boardToJSON(teamIdx, pieceToMove),
        success: function(msg) {
```

```
            move = msg;
        },
        error: function(jqXHR, exception) {
            if (jqXHR.status === 0) {
                alert('Unable to connect.\n Verify Network.');
            } else if (jqXHR.status === 404) {
                alert('Requested URL of HalmaAI not found. [404]');
            } else if (jqXHR.status === 500) {
                alert('Internal Server Error [500].');
            } else if (exception === 'parsererror') {
                alert('Data from HalmaAI was not JSON :( Parse failed.');
            } else if (exception === 'timeout') {
                alert('Time out error.');
            } else if (exception === 'abort') {
                alert('Ajax request aborted.');
            } else {
                alert('Uncaught Error.\n' + jqXHR.responseText);
            }
        }

      });

      return move;
}

function makeAjaxPostMoveRequestWithParm(teamIdx, pieceToMove) {

var move;

      $.ajax({
        type: 'POST',
        url: gTeamList[teamIdx].url,
        dataType: "text",
        async: false,
        data: {board: boardToJSON(teamIdx, pieceToMove)},
        success: function(msg) {
            move = msg;
        },
        error: function(jqXHR, exception) {
            if (jqXHR.status === 0) {
                alert('Unable to connect.\n Verify Network.');
            } else if (jqXHR.status === 404) {
                alert('Requested URL of HalmaAI not found. [404]');
            } else if (jqXHR.status === 500) {
                alert('Internal Server Error [500].');
            } else if (exception === 'parsererror') {
                alert('Data from HalmaAI was not JSON :( Parse failed.');
            } else if (exception === 'timeout') {
                alert('Time out error.');
            } else if (exception === 'abort') {
                alert('Ajax request aborted.');
            } else {
                alert('Uncaught Error.\n' + jqXHR.responseText);
            }
        }

      });

      return move;
}

function checkInputs() {
```

```
    var $myForm = $('#initForm');
    if (!$myForm[0].checkValidity()) {
        $myForm.find(':submit').click();
        return;
    }
    else {
        // calls new game and sets up teams
        initGame(null, document.getElementById('movecount'));
    }
}

$(document).ready(function() {
    $('#game').hide();
    $("#restartGame").hide();
    $('#initialization').show();
    $('#NoParm').click();   // default for radio button POST option


});

// format  data based on team turn
function boardToJSON(teamIdx, pieceToMove) {
    var enemyIdx = 0;
    if(teamIdx===0) enemyIdx = 1;

    var jsonStr =  JSON.stringify({
        "pieces" : gTeamList[teamIdx].teamPieces,
        "destinations" : gTeamList[teamIdx].teamDestinations,
        "boardSize" : kBoardHeight,
        "enemy" : gTeamList[enemyIdx].teamPieces,
        "currPiece" : pieceToMove,
        "moveCount" : gMoveCount

    });

    // debug via console
    console.log("--------------------------------");
    console.log("Team: " + gTeamList[teamIdx].name +
                " URL: " + gTeamList[teamIdx].url);
    console.log("Data TO AI " + jsonStr);


    return jsonStr;

}

function endsWith(str, suffix) {
    return str.indexOf(suffix, str.length - suffix.length) !== -1;
}

function areAllDestinationsFilled(destArr, pieceArr) {

    for (var i=0; i< destArr.length; i++) {
        // is piece at this loc
       var destOccupied = false; // find only one
       for (var k=0; k<pieceArr.length; k++) {
          if (destArr[i].x === pieceArr[k].x &&
                destArr[i].y === pieceArr[k].y  )  {
            destOccupied = true;
            break;
          }
       }
       if(destOccupied === false) return false;
```

```
    }
    // we get this far it is true that all are occupied
    return true;
}

function isGameOver() {
    // for each team
    // check if all destination pieces are occupied by one of their pieces

    for (var i=0; i<gTeamList.length; i++) {
        if (areAllDestinationsFilled( gTeamList[i].teamDestinations,
                                      gTeamList[i].teamPieces)) {
            var elt = document.getElementById('winnerCircle');
            elt.style.fontsize = 102;
            elt.innerHTML =" !!!!!WE HAVE A WINNER!!!!! >> " +
                    gTeamList[i].name +  " << "
            endGame();
            return true;
         }

    }
    // no winner if we get here
    return false;

}

function isOneSpaceAway(c1,c2) {
    diffx = Math.abs(c1.x - c2.x);
    diffy = Math.abs(c1.y - c2.y);
    diffxy = diffx + diffy;
    if  (diffxy === 1) return true;  // x y axis
    if (diffx===1 && diffy===1) return true;
    return false;  // not linear or diagonal
}

function isTwoSpacesAway(c1,c2) {
    diffx = Math.abs(c1.x - c2.x);
    diffy = Math.abs(c1.y - c2.y);
    // check x and y
    if  ((diffx === 2 && diffy === 0) ||
         (diffx === 0 && diffy === 2)  ) return true;  // x y axis
    // check diagonal
    if (diffx===2 && diffy===2) return true;
    return false;  // not linear or diagonal
}

// checks that src & dest are one cell apart and dest is free
function isLegalOneSquareMove(src, dest, gPiecesArr) {
   return (isOneSpaceAway(src,dest) && isFreeCell(dest, gPiecesArr));
}

// checks that 1) src & dest are two cells apart 2) dest is free
//             3) there exists a piece between src and dest
function isLegalTwoSquareJump(src, dest, gPiecesArr) {
    return (isTwoSpacesAway(src,dest) && isFreeCell(dest, gPiecesArr) &&
              isThereAPieceBetween(src, dest, gPiecesArr) );
}

// jumpArr will have original source piece followed by jump locations
function isArrayOfValidJumps(src, jumpArr, gPiecesArr) {
    // add src cell to array
    jumpArr.unshift(src);
    //consoleLogArray(jumpArr);
```

```
    while (jumpArr.length > 1) {
        // check first two cells for jump
        if (!isLegalTwoSquareJump(jumpArr[0], jumpArr[1], gPiecesArr)  ) {
          console.log("Illegal jump from " + jumpArr[0].toString() +
                  "to: " + jumpArr[1].toString());
          return false;
        }
        // remove first elt
        jumpArr.splice(0,1);
        //consoleLogArray(jumpArr);
    }

    // all valid jumps
    return true;
}

// checks if piece is holding its position
function  isPieceHoldingPosition(src, dest) {
    return (src.x === dest.x && src.y === dest.y);
}


// checks that array of requested moves is valid.
// if only one move in array, check either non-jump or one jump
// else check if all move pairs are jumping over some piece
function isValidMoveRequest(src, moveArr, gPieces) {
    if(moveArr.length === 0)  return false;

    if(moveArr.length === 1) {
        var dest = moveArr[0];  // only one
        return (isLegalOneSquareMove(src, dest, gPieces)  ||
                isLegalTwoSquareJump(src, dest, gPieces)  ||
                isPieceHoldingPosition(src,dest) );
    }

    // we have a multi jump request
    return isArrayOfValidJumps(src, moveArr, gPieces);
}
```