Specification for Program *AI-Tic-tac-toe*

Summary

This program allows the user to play tic-tac-toe against a computer opponent.

Program description

The state of the game is determined by the configuration of x's and o's on the board, and the difficulty level.

The display always shows the game board, consisting of two vertical and two horizontal line segments, a rectangular 'reset' button, and three circular difficulty buttons labeled 'easy', 'medium', and 'hard'. Exactly one of the round difficulty buttons will have a smaller circle within it, indicating the current difficulty level. If the game is over, a message is also displayed saying whether 'x' won, 'o' won, or the game was a tie.

The program plays according to the usual rules and has three difficulty settings: *easy, medium,* or *hard*.

The actions

The human always plays *x* and goes first. Whenever it is the human player's turn, he may place an *x* in an empty cell by clicking inside that cell. The computer will move automatically when it is o's turn. When the game is over, a message is displayed saying whether x won, o won, or the game was a tie. No moves can be made when the game is over.

In *easy* mode the computer moves at random; in *medium* mode the computer uses a simple strategy : win if you can; otherwise block if you must; otherwise, move in the center if possible; otherwise move in a corner if possible; otherwise move at random. In *hard* difficulty the computer plays perfectly: it cannot be beaten, and will win from any position where the computer can force a win.

The player may restart the game at any time by hitting the *reset button*. When this is done, all cells become empty. The difficulty level does not change when *new game* is pressed. The player can also change the difficulty level at any time, by clicking the appropriate button, and this action does not change the contents of the game board. That is, the player can change the difficulty setting in the middle of a game.

The game board is displayed at all times, showing the contents of each cell. Also displayed is a *reset* button, and three buttons for *easy, medium, and hard* difficulties respectively. One of these buttons will be highlighted at any time

The functions you write are:

- `display : Position * Position * Difficulty -> Sprite`
- `computerMove: Position*Position*Difficulty -> Cell`
- `easyClicked:pairOfInts -> Bool`
- `mediumClicked:pairOfInts -> Bool`
- `hardClicked: pairOfInts -> Bool`
- `inResetButton: pairOfInts -> Bool`
- `gameOver : Position * Position -> Bool`
- `legalMove : Cell * Position * Position -> Bool`
- `inCell: Cell * int * int -> Bool`

Note that these functions are specified fully in the starter code file.

Design for computerMove and mComputerMove

```
# computerMove: Position * Position * Difficulty -> Cell
# computerMove(xs,os,dif) is the move the computer makes in state (xs,os)
# on difficulty dif.

def computerMove(xs,os,dif):
   if dif == 'easy': return eComputerMove(xs,os)
   if dif=='medium': return mComputerMove(xs,os)
   if dif=='hard'  : return hComputerMove(xs,os)

# eComputerMove: position*position -> Cell
#
# eComputerMove(xs,os) is a random cell that is unoccupied in state (xs,os)

def eComputerMove(xs,os):
    emptyCells = [c for c in range(1,10) if not c in xs|os]
    return choice(emptyCells)
```

```
# mComputerMove: position * position * difficulty -> Cell
# If the game is not over, mComputerMove(xs,os) is the move the computer
# makes in state (xs,os) on meduium difficulty.

def mComputerMove(xs,os):
   winners = [w for w in range(1,10) if winner(w)]
   if len(winners)>0: return winner[0]
   blockers = [w for w in range(1,10) if blocker(w)]
   if len(blockers)>0: return blocker[0]
   if not occupied(5,xs,os): return 5
   corners = [w for w in [1,3,7,9] if not occupied(w,xs,os)]
   if len(corners)>0: return corners[0]
   open = [c for c in range(1,10) if not occupied(c,xs,os)]
   return open[0]




# winner: position*position*Cell -> Bool
# winner(xs,os,c) iff that in state (xs,os), o wins by moving in c.

# blocker: position*position*Cell -> Bool
# blocker(xs,os,c) iff that in state (xs,os), there is a row in which x has
# two marks and c is unoccupied.
```