# TicTacToe.lhs

James Cook

2007

```
module TicTacToe where
  import Data.Map
  import Env
```

Data types describing the game. The whole state of the game is “whose turn it is” and the marks on the board.

```
data XO = X | O
  deriving (Eq, Show, Enum)
data Int3 = I | II | III
  deriving (Eq, Ord, Show)
type Board = Map (Int3, Int3) XO
data Game = Game{
  whoseTurn :: XO,
  board :: Board
  } deriving (Eq, Show)
```

Some very basic operations
newGame could be pulled into some general “class”, i suppose

```
newGame :: Game
newGame = Game{whoseTurn = X, board = empty}
mark :: Game → (Int3, Int3) → Game
mark game square = game{whoseTurn = otherGuy, board = marked}
  where
    otherGuy = succ (whoseTurn game)
    marked = insert square (whoseTurn game) (board game)
```

Tic Tac Toe game logic. These definitions encapsulate the “rules” of the game in a set of functions that can easily be wrapped into several different agent-based evaluation strategies.

First, the queries:

- WhoseTurn : either player may ask

- What's in a square : either player may ask

```
data WhoseTurn = WhoseTurn
  deriving (Eq, Show)
instance EnvQuery Game XO WhoseTurn XO
  where queryEnv game _ _ = whoseTurn game
instance EnvQuery Game XO (Int3, Int3) (Maybe XO)
  where queryEnv game _ square = Data.Map.lookup square (board game)
```

Second, the actions (of which there are only one):

- Mark a square: Only the player whose turn it is may do this. Additionally, if the square is taken, the action fails.

```
instance EnvAction Game XO (Int3, Int3)
  where
    actEnv game xo square
      | xo ≢ (whoseTurn game)
        = Right "Not your turn, bud!"
      | Data.Map.lookup square (board game) ≢ Nothing
        = Right "That square is taken."
      | otherwise
        = Left (mark game square)
```