Checkpoint #1

At this point, you should have completed initial code (datatypes and some functions) for the project. Stop by office hours in Lally 314 on Wednesday November 13th or on Friday November 15th after class at the latest and show your progress. You can send a representative of your team though, of course, all team members are welcome to attend. There will be no checkpointing after next week and I'll do the meetings on first-come-first-serve basis. So, it is probably a good idea to plan on showing your code as early as possible!

I will be looking at the code in your repository and looking for answers to the following questions:

1. What modules will comprise your project?
2. What data structures will you use to model your project?
3. What functions will you need to write? What are their type signatures?
4. What testing will you do?
5. What questions do you have for me?

The checkpoint is worth 10% of the project grade. If there is no work or very little work done, I'll mark a low grade for checkpoint #1.

You should have something like this. This is a sketch of Tic-Tac-Toe, however, keep in mind that your project should be a lot more involved than that. There is an example of Tic-Tac-Toe in the textbook as well, Chapter 11 and it might be useful to go over that chapter as well especially if you are working on a game.

```
import Data.Maybe (isJust)
import qualified Data.Map as M
import Test.HUnit
import Test.QuickCheck
import Control.Monad.State

-----------------------------
-- type definitions (model)
-----------------------------
data Player   = X | O deriving (Eq, Show)

data Location = Loc Int Int deriving (Eq, Ord, Show)

type Board    = M.Map Location Player

data Game     = Game { board :: Board , current :: Player } deriving (Eq,
Show)

data End      = Win Player | Tie deriving (Eq, Show)
```

```
----------------------------
-- function declarations
----------------------------
```

You should also sketch some of the functions that you plan to implement by giving their type signatures.

Each function should have a comment explaining what it should do.

Include as many functions here as you can, so that you can write and test several use cases.

```
-- | starting board for the game
initialGame :: Game
initialGame = undefined

-- | is the board still playable
checkEnd :: Board -> Maybe End
checkEnd = undefined

-- | is this location a valid move for the player
valid :: Board -> Location -> Bool
valid = undefined

-- | make a move at a particular location
makeMove :: Game -> Location -> Maybe Game
makeMove = undefined

-- | display the current game board
showBoard :: Board -> String
showBoard = undefined
```

You can write (and show me) code as well!

```
-- | Create a type class for the interface for the main game interface
-- so that it can be tested

class Monad m => Interface m where
    -- ask the current player for their next move
    getMove       :: Game -> m Location
    -- send a message to all players
    message       :: String -> m ()
    -- send a message to the indicated player
    playerMessage :: Player -> String -> m ()

-- | all valid locations
locations :: [Location]
locations  = [Loc x y | x <- [1 .. 3], y <- [1 .. 3] ]

-- | make moves until someone wins
playGame :: Interface m => Game -> m ()
playGame game = do
   playerMessage (current game) $ showBoard (board game)
   case checkEnd $ board game of
     Just (Win p)  -> message $ "Player " ++ show p ++ " wins!"
```

```
      Just Tie      -> message $ "It's a Tie!"
      Nothing       -> do
        playerMessage (current game) $ "It's your turn"
        move <- getMove game
        case makeMove game move of
          Just game' -> playGame game'
          Nothing    -> error "BUG: move is invalid!"

instance Interface IO where
   getMove       = undefined
   playerMessage = undefined
   message       = undefined

main :: IO ()
main = playGame initialGame

-----------------------------
-- Test Cases
-----------------------------
```

Your checkpoint *must* also include plans for testing your code. Ideally you would write testing code for one or two use cases. The testing code does not need to be complete, but you should have at least some code, not just prose describing what the test will be doing. Many teams have stated that they are planning on using HUnit or Quickcheck, so go ahead and write test cases using these frameworks!

Code example is due to a writeup by Stephanie Weirich; text is adapted to our class.