1. One cannot *pass* (skip a turn) unless there are NO moves it can make (this is part of the regular rules).  In our case, if a player passes, and there IS a move that could have been made, the game ends immediately.
2. One cannot move unless that move involves flipping at least one of the opponent's discs (this is part the regular rules).  In our case, if a player makes such an illegal move, the game ends immediately.
3. It is possible a game can end without the entire board being filled.  So we normally end the game when there are two (legal) passes in a row.

**Functional Requirements**

1. **Run modes**.  The program can be run either as a command line program or from the interactive environment by typing "main".
2. **Command line**.  If the program is run from the command line, it can take either 0 or 2 arguments.  If there no arguments, then the programs goes into interactive mode (see the "interactive mode" requirement).  If there are two arguments, then these are to be taken as a strategy name for the black and white players respectively.  If these are both legal strategy names, then the program continues by playing the two strategies against one another.  If one or both of these are not legal strategy names, then the program prints a list of legal strategy names (see requirement "[printing strategy names](http://kremer.cpsc.ucalgary.ca/courses/cpsc449/W2015/assnHaskell.html#printingStrategyNames)") and quits.
3. **Interactive startup**.  If the program is run from within the interactive environment by typing "main", then the program will immediately enter interactive mode (see the "[interactive mode](http://kremer.cpsc.ucalgary.ca/courses/cpsc449/W2015/assnHaskell.html#interactiveReq)" requirement).
4. **Interactive mode**. If the program enters interactive mode then it will print a descriptive message, including the list of possible strategy names (see requirement "[printing strategy names](http://kremer.cpsc.ucalgary.ca/courses/cpsc449/W2015/assnHaskell.html#printingStrategyNames)").  It will then prompt for a black strategy name, and then prompt for a white strategy name.  If one are both of these are not legal strategy names, then the program prints a list of legal strategy names and quits. If these are both legal strategy names, then the program continues by playing the two strategies against one another.
5. **Printing strategy names**. Whenever the program prints a list of strategy names, it will print each strategy on a separate line with exactly two spaces (only) before the name of each strategy.  A newline should immediately follow each strategy name.  No other program output shall begin a line with exactly two spaces followed immediately by a non-space character.
6. **One game**. If the program obtains the two legal strategy names, it will play exactly one game and then quite.
7. **Game play**. The game is played by first playing the black strategy (a function type Chooser = GameState -> Cell -> Maybe (Int,Int)) alternatively with the white strategy.
8. **Game tracing**. The program must print out a trace of the game starting with the initial state (the board for which you can get from calling OthelloTools.initBoard) to the final state.  The format of the state must include the state of the board in the form:  
     
   <play-description>  
    \_ \_ \_ \_ \_ \_ \_ \_  
   |W|\_|B|\_|B|B|\_|\_|  
   |W|\_|B|B|B|B|\_|\_|  
   |W|W|B|W|B|B|B|B|  
   |W|B|B|B|B|B|B|B|  
   |W|\_|W|B|B|B|B|B|  
   |W|B|W|W|B|B|B|B|  
   |W|B|B|W|W|B|B|B|  
   |W|B|W|W|W|W|W|W|  
     
   That is, a <play-description> line, followed immediately by an intro line consisting of <space><underscore> repeated 8 times, and then each of 8 lines consisting of alternating vertical bars and cell characters ("B" for black, "W" for white, and underscore for empty) starting and ending with a vertical bar.  There can be no characters (white space or otherwise) before or after lines these lines and these lines must be consecutive (no lines interspersed between them).  The <play-description> line is of the form:  
     
   (*player*,*played*)  
     
   with no space before or after the parentheses, where *player* may be "Black" or "White", and *played* may be one of  
     - "Played (*int1*, *int2*)" (where int1 and int2 are integers in the range 0..7 inclusive)  
     - "Init"  
     - "Passed"  
     - "Goofed (*int1*,*int2*)"  
     - "IllegalPass".  
   No other lines in the output shall begin with the character "(", including any "conversational" lines in the interactive mode of the program (see the [interactive mode requirement](http://kremer.cpsc.ucalgary.ca/courses/cpsc449/W2015/assnHaskell.html#interactiveReq)).  
     The show function associated with the GameState type in OthelloTools will easily achieve this for you -- it has implemented this for you already.  For an example, see [my listing of a complete run](http://kremer.cpsc.ucalgary.ca/courses/cpsc449/W2015/assnHaskellExample.txt). **It is absolutely critical that you follow this requirement exactly as we will use a program that parses your output to check your work.**
9. **Strategies**: You should implement at least 3 different strategies to choose from.  For best learning, each group member should independently implement their own strategy.  All strategies should have a type of type Chooser = GameState -> Cell -> Maybe (Int,Int).  Where returning a value of Nothing indicates a pass (ie: the player cannot make a move).

**Non-Functional Requirements**

1. **Documentation**.  Your source code should be well documented to the [haddock](https://www.haskell.org/haddock/doc/html/index.html) standard.  As part of the marking, we may run haddock on your source code to see the "haddock coverage" percentage.