Dots Game Analysis

User Specification: Interact with the computer to play the dots game. The game will facilitate the human and the computer taking turns **selecting lines** to connect the dots to **capture cells**.

I. Prompt for instructions
   a. Prompt for instructions: if "yes", display the instructions

II. If the human would like to go first then
   a. Human selects a line
      i. If the line is valid then the computer will track it. Otherwise redo the selection and go to II. a.
      ii. If the line is a capture then
         1. identify it with the initial 'B', increment captures
         2. if the number of captures equals the maximum, the human wins, otherwise go to II. a.
   b. Computer selects a valid line and tracks it
      i. If the line is a capture then
         1. identify it with the initial 'A', increment captures
         2. if the number of captures equals the maximum, the computer wins, otherwise go to II. b.

III. Computer calculates whether the captures are equal to the maximum captures.
   a. If equal then
      i. Summarize the results for winner and loser
      ii. Inquire about another game
   b. Otherwise go to II. a.

Areas of Expertise: data representation (~~beginner~~ intermediate), human computer interaction in a text based fashion (beginner), artificial intelligence (beginner)

Time Required: ~~40 hours~~ ~~60 hours~~ ~~32.5 hours~~ 84 hours

Costs: ~~no costs~~ $0.50

Definitions:

Selecting lines – lines are selected between two adjacent dots in a horizontal or vertical arrangement only.
Capture cells – cells are captured when four lines enclose a cell of four dots in the smallest square possible. When a cell is captured it is counted as one point for the player who completes the capture. The player who completes the capture will take another turn.
Maximum captures – The maximum is defined as MAX_ROW * MAX_COL.

Dots Game Design

Software Requirements

Modules: Hide the computer's move calculations, place function declarations and defines in separate header files.

Languages and Tools: Use C, it is not necessary to be ISO compatible as this will be for my own investigation and practice.

Time Spent:

(Concept: 3 hours, pre analysis)
Analysis: 4.5 hours
Design: 16 hours
Implementation: 53 hours
Testing: 12 hours
Total: 84

Cost: $0.50

Final Review: The project works as tested with 2x2 and 3x2 grids. There is an issue with scoring odd x odd grids (e.g. 3x3) where the integer division rounds down.

The file size for a 3x2 game is the same as a 9x6 game. (contiguous memory stacks)

The game play is a losing battle for the computer as is. There needs to be some improvement over just guessing random valid lines. The computer invariably gives the human too much opportunity to capture cells. **Continue development C++ 11/28/11.**

Need to compile with –d1 option before releasing.

Tested release version 1/23/12, the computer lost horribly! It still is not "seeing" the weight of the tunnels within the 10 random guesses available. A better method of "seeing" and avoiding the longer tunnels as they build up is needed. Alternative: the lists are dynamically allocated and may allow for brute force guessing. Additionally, it seems that smart guessing starts too late (20% of moves). A problem was noted in the computer not reporting a move, just "My Capture!."

Corrected release version 7/13/12, a bug was found in the random() generator (library). The correction was incorporated into the game.

Improved release version 5/21/13, increase the number of random guesses to 50 to create competitiveness.

Smart Guessing

Original problem – Select a move that will 1) not provide opportunity for capture or 2) provide a minimal opportunity for capture.

Detailed sub-problems – First, select a guess G. Then:

A. Check if the initial is ‘3’. If so, use the move since the computer will capture.
    a. Increment move counter, return G.
B. Point to the first list. Check the move for potential capture.
    a. If this cell’s initial is ‘2’ then list it and check the other possibility.
        i. Put the move in pointed to list.
        ii. Get the other possible direction P. If P’s direction’s cell is valid, create a new move and check the new move for potential capture.
C. Check the move’s complement for potential capture.
    a. If the compliment is a valid cell, create a move with this valid complementary cell and the direction from which it originated.
    b. If this newly created cell’s initial is ‘2’ then list it and check the other possibility.
        i. Put the move in pointed to list.
        ii. Get the other possible direction P. If P’s direction’s cell is valid, create a new move and check the new move for potential capture.
D. If P’s list size > 0 and lists are not used up, increment list pointer and select a new guess G and go to (B.a). Otherwise return G.
    a. Increment move counter, return G.
E. After a maximum of N smart guesses, use the first guess from the list with the least items.
    a. Assign G the row and column and compass direction of the smart guess.

N: The maximum quantity of items in list, suggested starting value is five items.
P: Pointer to the list currently being added to.
G: A structure with members: enumeration {north, south, east, west}, compass; integer, row; integer, column.
Valid cell: a valid cell is not off the grid and not already in the list.