CS2303 Systems Programming Concepts **C08**

Program 2 **36 Points**

Due: January 28, 2008 at 11:59 p.m.

Motion Collision Simulation using Arrays in C

This assignment emphasizes the use of ***arrays*** in C and provides practice with manipulating subscripts and abstracting with 3-dimensional arrays. You may use ***structs*** in this assignment but it is possible to complete this program successfully only using ***arrays.*** Moreover, since this assignment is being done early in the course, students can keep the playing field arrays and other player attribute data structures as global variables. However, this does **NOT** preclude the expectation that the control structure of this program consist of many small functions and routines where appropriate parameters are passed.

Note well – While the program assignment is given with fixed sizes for the playing field and the team sizes, the expectation is that students will use ***#define*** statements such that the program can easily be adapted to variable sizes by simply changing a few ***#define*** statements.

This assignment comes in three versions with differing maximum points:

I.	Hardest	36 Points
II.	Slightly Easier	34 Points
III.	Easy	32 Points

Each student must choose and indicate in their **README** file the game they have chosen to simulate for program 2.

Version I (Hardest Version): Motion Smashball

Your task is to simulate a game between two teams of nine players. The reason for nine players is to facilitate reasonable printing out of the playing field onto the screen or into an output file.

Smashball is played on a 25 by 25 square field. When printing out game results, be sure to print out the field with a border around the field.

Main Assignment

1. Game Initialization

   a. Use a random number generator to randomly place all the players on both teams in the playing field. If the random process produces two players starting on the same spot, redraw new random positions such that all players start at unique spots.

   b. Use a random number generator to generate an initial motion direction for each player. The four motion choices for this assignment are: East, West, North and South. These directions correspond to the respective movement of each player on the playing field.

2. Playing Smashball

   a. Smashball is played in rounds with each remaining player from both teams moving one hop in their current direction. Alternate turns of the players from both teams such that each remaining player gets one hop per round using as fair a policy as possible to reduce bias (Explain your turn policy in your READEME file).

   b. Player Elimination: As each player hops to its next spot, the program checks to see if that spot is currently occupied by a player on the opposing team. If this is the case, the ***hopping player*** is instantly ***smashed*** and eliminated from the game and taken off the playing field.

   c. Hop Boosting: As each player hops to its next spot, the program checks to see if that spot is currently occupied by a teammate. If this is the case, the ***hopping player*** moves again (effectively hopping over any teammates it encounters).

   d. Hop Bouncing: When a player attempts to make its next hop and it currently resides on an edge spot such that its next hop would result it hopping into a border spot, the player 'bounces off' the border by reversing its direction and making its hop into the first available spot in the opposite direction. For example, if a player currently residing at the spot identified by row 3, column 25 on the playing field attempts to hop to the East when its turn to move comes, then this play changes direction to West and then attempts to hop into the spot at row 3, column 24. Note – rules b and c also apply to this type of hop.

3. Ending the Game

   Smashball continues for a fixed number of rounds (e.g., 100 rounds) or it terminates early if one of the teams win. A team wins when all players from the opposing team have been eliminated. If both teams finish with surviving players the game is declared a tie.

Version II: (Slightly Easier): Motion Smashball with Teammate Collisions

Everything in Version II of Motion Smashball is the same as above except that rule 2c, hop boosting, is eliminated and rule 2b is changed to:

> 2b'. Player Elimination: As each player hops to its next spot, the program checks to see if that spot is currently occupied by any other player. If this is the case, the ***hopping player*** is instantly eliminated from the game and taken off the playing field. Thus, in this version of Smashball, players can eliminate their teammates.

Version III: (Easy) Brownian Motion

Version III, titled 'Brownian Motion' differs from the previous two versions of the assignment in that there only exists one team with 25 players. Everything else in Brownian Motion is the same as Version II above except that steps 1 and 3 are changed to the following:

1. Game Initialization

   a. Use a random number generator to randomly place all the players in the playing field. If the random process produces two players starting on the same position, redraw new random positions such that all players start on unique spots.

   b. Use a random number generator to generate an initial motion direction for each player. The four motion choices for this assignment are East, West, North and South. These directions correspond to the respective movement of each player on the playing field.

2. Ending the Game

   Brownian Motion continues for a fixed number of rounds (e.g., 100 rounds) or it terminates early if only **ONE** player remains.

Program Output

Your output routines need to be designed to produce line oriented output to be sent to the screen or to an output file for grading analysis.

Your program must include a function that prints out the playing field with the border and the current player positions. Identify the player positions by printing out a maximum of either two-digit integers or ASCII strings of length 2. Note, using appropriate integer outputs is acceptable and easier that using strings. An example of using integer outputs would be to indicate the players on the first team by integers from 11 to 19 and players on the opposing team by integers between 21 and 29. The fancier, but optional, output is to indicate the players on two teams, say the Patriots versus the Colts, by indicating a players spot on the field using strings ‘P1’ to ‘P9’ to indicate Patriot players and strings ‘C1 to C9’ to indicate Colt players.

Your program should include a second print function that prints out a table of the remaining current player motion directions prior to starting a new round.

Standard Output

To simplify the grading process, standard output consists of:

1. a printout of the playing field showing all the initial players prior to the start of the first round of the game and a printout of tables showing the initial motion directions of each player.
2. a printout of the playing field showing the position of all the remaining players after the completion of every 10th round.
3. a final printout of the playing field showing all surviving players after the game has terminated. This should be followed by printed lines (depending on the game simulated) indicating the winning team or in the case of a tie, a count of the number of surviving players for both teams. If Brownian Motion ends early, print out a line identifying the lucky Lone Survivor!

Program Development Hints

Remember this program will be graded taking into account good organization and the use of multiple functions and routines. It is highly recommended that students utilize a debugger and/or include a conditional define debug option that can easily be turned on and off. While debugging your program, you should print out significant events such as player eliminations or player bounces.

What to turn in for Program 2

Turn in your assignment using the ***turnin*** program on the CCC machines. You should turn in a tarred file that includes your source code, a make file, a README file and at least two sample output files. The make file should include the ability to cleanup leftover output and intermediate files between compile and execution cycles. The README file should provide any information to help the TA or SA test your program, change the playing field size and evaluate your ouput files for grading purposes. The two sample output files should show the output from your program for two different playing field sizes.