Lab #4

CS-2050

February 16, 2024

1 Requirements

In this lab, you will recreate your game from last lab, using a struct instead. Just as with before, the arena will be an array, where each cell (or index) represents a ship. Note that, again, in this version of the game, **each ship is exactly one cell in size**.

You must write the functions required to create and manage the game board, keep track of scoring, and put away (or free) the game board.

In this lab, you are given the following struct definition:

```
typedef struct {
    int shots;
    int hits;
    float score;
    int arena[BOARD_SIZE];
} GameBoard;
```

1.1 newBoard

```
GameBoard * newBoard()
```

Info: This function creates a new empty game board, where the user (IE: the main function) cannot directly interact with the arena. The game board must store information using the struct definition given in the starter code, which includes the number of shots taken, the number of successful hits, and the score. The score is calculated by `hits / shots`. This function will return a pointer to the new GameBoard on success, or NULL on failure.

An empty cell is represented by a 0 in the array, a ship is represented by a 1 in the array, and a destroyed ship is represented by a -1 in the array.

Value	Meaning
-1	Destroyed shp
0	Empty cell
1	Ship

1.2 takeShot

```
int takeShot(GameBoard *board, int cell)
```

Info: This function takes a game board, and a cell (IE: index) to shoot at. If the given cell is occupied by a ship (1), then the cell is changed to a -1 to indicate a hit. If the given cell is either empty (0), or destroyed (-1), then the cell is not changed and no hit occurs.

This function must also update the relevant game values after each shot. It will return 1 on a hit, or 0 on a miss.

1.3 countRemainingShips

```
int countRemainingShips(GameBoard *board)
```

Info: This function takes a board, and returns the number of remaining (not destroyed) ships on it.

1.4 getShotsTaken

```
int getShotsTaken(GameBoard *board)
```

Info: This function takes a board, and returns the number of shots taken.

1.5 getHits

```
int getHits(GameBoard *board)
```

Info: This function takes a board, and returns the number of successful hits.

1.6 getScore

```
float getScore(GameBoard *board)
```

Info: This function takes a board, and returns the score. **Note that this function must not calculate the score**.

1.7 placeShip

```
int placeShip(GameBoard *board, int cell)
```

Info: This function takes a board, and places a ship on it in the given cell. If the given cell was already occupied (1) or destroyed (-1), then the cell is not updated and this function returns 0. If the given cell was empty (0), then the cell is set to 1, and this function returns 1.

1.8 endGame

```
void endGame(GameBoard *board)
```

Info: This function takes a board, and frees the memory allocated to it.

Grading: 15 points

1. Write required *newBoard* function
   * 5 points
2. Write required *takeShot* function
   * 3 points
3. Write required *countRemainingShips* function
   * 2 points
4. Write required *getShotsTaken* function
   * 1 points
5. Write required *getHits* function
   * 1 points
6. Write required *getScore* function
   * 1 points
7. Write required *placeShip* function
   * 1 points
8. Write required *endGame* function
   * 1 points

Notice:

1. All of your lab submissions **must** include documentation to receive full points.
2. All of your lab submissions must compile under GCC using the $-Wall$, $-Werror$, and $-Wpedantic$ flags to be considered for a grade.
3. You are expected to provide proper documentation in every lab submission, in the form of code comments. For an example of proper lab documentation and a clear description of our expectations, see the lab intro PDF.