Exam 2: Valencia Battleship Game (Single Player Game)

You can do this assignment alone or you can form groups of 3 or less students. If you form a group, all names must be in the code. Only one member will submit the project.

You will be divided into teams. Your team will design a program that will be used to play Valencia Battleship. Valencia Battleship is played on a 10 x 10 grid. The game begins with the five ships (see below) being placed randomly somewhere in the 10 x 10 grid. You will not know the location of the ships, they will be randomly placed.

**The 5 Ships in the game:**

1. (S) Seminole State Ship (3 Holes)

2. (A) Air Force Academy (5 Holes)

3. (V) Valencia Destroyer (4 Holes)

4. (E) Eskimo University (3 Holes)

5. (D) Deland High School (2 Holes)

When the game starts, the screen will look as shown below.

The player will either fire a missile or quit the game. If they fire a missile, you will place an “H” if they hit a ship or an “M” if they miss a ship. You will adjust the missile counters. Once a ship has been sunk, you will replace the “H” in the sunk ship with the letter pertaining to the sunk ship. You will now show the ship as sunk.

**Placing Ships**

Ships cannot collide with other ships. No overlapping. Ships cannot run off the game board.

**MAIN MENU**

**GAME PLAY**

The goal is for the player to sink all ships with while using the least amount of missiles possible. A perfect game will use 17 missiles.

1. Starts a new game from scratch.
2. If an previous game was not finish, you can resume here, otherwise inform the user and start a new game
3. You will show the top 10 best scores sorted from best to not best. Remember the lower the score the better. You will display the top 10 scores to both the screen and to a text file named topTenScores.txt. The text file will live in the src folder.
4. If they quit, you will preserve what needs to be preserved and exit.

\*\* See attached Rubric for other requirements \*\*

//-----------------------------------------------------------------------------

// BATTLESHIP GAME PSEUDOCODE (UX Improved)

//-----------------------------------------------------------------------------

// I. CONSTANTS & GLOBAL DEFINITIONS

CONSTANT GRID\_SIZE = 10

CONSTANT MAX\_SHIPS = 5

CONSTANT SAVE\_FILE\_NAME = "battleship\_savegame.dat"

CONSTANT SCORE\_FILE\_NAME = "src/topTenScores.txt"

DEFINE SHIP\_TYPES = [

{ name\_long: "Seminole State Ship", letter: 'S', size: 3 },

{ name\_long: "Air Force Academy", letter: 'A', size: 5 },

{ name\_long: "Valencia Destroyer", letter: 'V', size: 4 },

{ name\_long: "Eskimo University", letter: 'E', size: 3 },

{ name\_long: "Deland High School", letter: 'D', size: 2 }

]

DEFINE EMPTY\_CELL = '~'

DEFINE MISS\_CELL = 'M'

DEFINE HIT\_CELL = 'H'

ENUM ShotParseError:

NO\_ERROR,

INVALID\_FORMAT,

OUT\_OF\_BOUNDS\_ROW,

OUT\_OF\_BOUNDS\_COL

END ENUM

// II. DATA STRUCTURES

STRUCTURE Coordinate:

row AS INTEGER

col AS INTEGER

END STRUCTURE

STRUCTURE Ship:

name\_long AS STRING

letter AS CHAR

size AS INTEGER

hits\_taken AS INTEGER

is\_sunk AS BOOLEAN

segments AS ARRAY[size] OF Coordinate

END STRUCTURE

STRUCTURE GameState:

computer\_ocean\_grid AS ARRAY[GRID\_SIZE][GRID\_SIZE] OF CHAR

player\_target\_grid AS ARRAY[GRID\_SIZE][GRID\_SIZE] OF CHAR

computer\_fleet AS ARRAY[MAX\_SHIPS] OF Ship

missiles\_fired\_count AS INTEGER

ships\_remaining\_count AS INTEGER

game\_in\_progress AS BOOLEAN

last\_shot\_coord AS Coordinate

last\_shot\_valid AS BOOLEAN // To know if last\_shot\_coord is meaningful

END STRUCTURE

STRUCTURE ScoreEntry:

player\_name AS STRING[50]

score\_value AS INTEGER

date\_time\_achieved AS STRING[30]

END STRUCTURE

// III. MAIN GAME FUNCTION

FUNCTION MainGame():

GameState current\_game

current\_game.game\_in\_progress = FALSE

current\_game.last\_shot\_valid = FALSE

BOOLEAN running = TRUE

WHILE running IS TRUE:

DisplayMainMenu()

choice = GetMenuChoice()

SWITCH choice:

CASE 1: // Start New Game

InitializeNewGame(current\_game)

PlayGame(current\_game)

CASE 2: // Resume Game

IF LoadGameState(current\_game) IS TRUE THEN

DisplayMessage("Game resumed.")

PlayGame(current\_game)

ELSE

DisplayMessage("No saved game found or error loading. Starting new game.")

InitializeNewGame(current\_game)

PlayGame(current\_game)

END IF

CASE 3: // View Top 10 Scores

ViewTopScores()

CASE 4: // How to Play

DisplayHelpScreen()

CASE 5: // Quit

IF current\_game.game\_in\_progress IS TRUE THEN

prompt\_save = AskUser("Save current game before quitting? (Y/N)")

IF prompt\_save IS 'Y' THEN

SaveGameState(current\_game)

DisplayMessage("Game saved.")

END IF

END IF

running = FALSE

DisplayMessage("Exiting game. Goodbye!")

DEFAULT:

DisplayMessage("Invalid choice. Please try again.")

END SWITCH

END WHILE

END FUNCTION

// IV. MENU FUNCTIONS

FUNCTION DisplayMainMenu():

ClearScreen()

PrintLine("=========================")

PrintLine(" B A T T L E S H I P ")

PrintLine("=========================")

PrintLine("MAIN MENU")

PrintLine("1. Start New Game")

PrintLine("2. Resume Game")

PrintLine("3. View Top 10 Scores")

PrintLine("4. How to Play")

PrintLine("5. Quit Game")

PrintLine("-------------------------")

END FUNCTION

FUNCTION GetMenuChoice() AS INTEGER:

INPUT "Enter your choice (1-5): " choice

RETURN choice

END FUNCTION

FUNCTION DisplayHelpScreen():

ClearScreen()

PrintLine("--- HOW TO PLAY BATTLESHIP ---")

PrintLine("The goal is to sink all 5 enemy ships hidden on the grid.")

PrintLine("Ships are: Seminole (3), Air Force (5), Valencia (4), Eskimo (3), Deland (2).")

PrintLine("Enter coordinates as LetterNumber (e.g., A5, J10).")

PrintLine("'~' is empty water, 'M' is a miss, 'H' is a hit.")

PrintLine("When a ship sinks, its letter (S,A,V,E,D) will appear on the grid.")

PrintLine("Try to use the fewest missiles possible!")

PrintLine("A perfect game uses 17 missiles.")

PrintLine("------------------------------")

INPUT "Press Enter to return to menu..."

END FUNCTION

// V. GAME SETUP FUNCTIONS

FUNCTION InitializeNewGame(REFERENCE GameState game):

FOR r FROM 0 TO GRID\_SIZE - 1:

FOR c FROM 0 TO GRID\_SIZE - 1:

game.player\_target\_grid[r][c] = EMPTY\_CELL

game.computer\_ocean\_grid[r][c] = EMPTY\_CELL

END FOR

END FOR

FOR i FROM 0 TO MAX\_SHIPS - 1:

game.computer\_fleet[i].name\_long = SHIP\_TYPES[i].name\_long

game.computer\_fleet[i].letter = SHIP\_TYPES[i].letter

game.computer\_fleet[i].size = SHIP\_TYPES[i].size

game.computer\_fleet[i].hits\_taken = 0

game.computer\_fleet[i].is\_sunk = FALSE

END FOR

game.missiles\_fired\_count = 0

game.ships\_remaining\_count = MAX\_SHIPS

game.game\_in\_progress = TRUE

game.last\_shot\_valid = FALSE

SetupComputerShips(game)

DisplayMessage("New game initialized. Computer has placed its ships.")

END FUNCTION

FUNCTION SetupComputerShips(REFERENCE GameState game):

// Logic for placing ships remains the same as v1.

FOR i FROM 0 TO MAX\_SHIPS - 1:

current\_ship\_to\_place = game.computer\_fleet[i]

placed\_successfully = FALSE

WHILE placed\_successfully IS FALSE:

start\_row = RandomInteger(0, GRID\_SIZE - 1)

start\_col = RandomInteger(0, GRID\_SIZE - 1)

orientation = RandomInteger(0, 1)

IF IsValidShipPlacement(game.computer\_ocean\_grid, current\_ship\_to\_place, start\_row, start\_col, orientation) THEN

FOR j FROM 0 TO current\_ship\_to\_place.size - 1:

IF orientation IS 0 THEN

game.computer\_ocean\_grid[start\_row][start\_col + j] = current\_ship\_to\_place.letter

game.computer\_fleet[i].segments[j].row = start\_row

game.computer\_fleet[i].segments[j].col = start\_col + j

ELSE

game.computer\_ocean\_grid[start\_row + j][start\_col] = current\_ship\_to\_place.letter

game.computer\_fleet[i].segments[j].row = start\_row + j

game.computer\_fleet[i].segments[j].col = start\_col

END IF

END FOR

placed\_successfully = TRUE

END IF

END WHILE

END FOR

END FUNCTION

FUNCTION IsValidShipPlacement(READONLY ARRAY computer\_grid, READONLY Ship ship\_info, start\_row, start\_col, orientation) AS BOOLEAN:

// Logic remains the same as v1.

IF orientation IS 0 THEN

IF start\_col + ship\_info.size > GRID\_SIZE THEN RETURN FALSE

ELSE

IF start\_row + ship\_info.size > GRID\_SIZE THEN RETURN FALSE

END IF

FOR i FROM 0 TO ship\_info.size - 1:

current\_r = start\_row

current\_c = start\_col

IF orientation IS 0 THEN current\_c = start\_col + i ELSE current\_r = start\_row + i END IF

IF computer\_grid[current\_r][current\_c] IS NOT EMPTY\_CELL THEN RETURN FALSE

END FOR

RETURN TRUE

END FUNCTION

// VI. GAMEPLAY LOOP FUNCTION

FUNCTION PlayGame(REFERENCE GameState game):

WHILE game.ships\_remaining\_count > 0 AND game.game\_in\_progress IS TRUE:

ClearScreen()

DisplayPlayerTargetGrid(game.player\_target\_grid, game.last\_shot\_coord, game.last\_shot\_valid) // UX Note: Use colors here

DisplayShipStatusAndStats(game)

PrintLine("Enter 'quit' to return to main menu.")

shot\_input\_str = GetPlayerShotInput("Enter missile target (e.g., A1-J10): ")

IF shot\_input\_str IS "quit" THEN

// Save prompt logic remains same as v1

prompt\_save = AskUser("Save current game before returning to menu? (Y/N)")

IF prompt\_save IS 'Y' THEN SaveGameState(game); DisplayMessage("Game saved.") END IF

game.game\_in\_progress = FALSE

BREAK LOOP

END IF

parse\_error, shot\_row, shot\_col = ParseShotCoordinates(shot\_input\_str)

game.last\_shot\_valid = FALSE // Reset before new shot attempt

IF parse\_error IS NOT ShotParseError.NO\_ERROR THEN

SWITCH parse\_error:

CASE ShotParseError.INVALID\_FORMAT: DisplayMessage("Invalid coordinate format. Use LetterNumber (e.g., A5).")

CASE ShotParseError.OUT\_OF\_BOUNDS\_ROW: DisplayMessage("Invalid row. Must be 1-10.")

CASE ShotParseError.OUT\_OF\_BOUNDS\_COL: DisplayMessage("Invalid column. Must be A-J.")

END SWITCH

Pause(2 seconds)

CONTINUE LOOP

END IF

game.last\_shot\_coord.row = shot\_row

game.last\_shot\_coord.col = shot\_col

game.last\_shot\_valid = TRUE

IF game.player\_target\_grid[shot\_row][shot\_col] IS NOT EMPTY\_CELL AND game.player\_target\_grid[shot\_row][shot\_col] IS NOT HIT\_CELL THEN

IF game.player\_target\_grid[shot\_row][shot\_col] IS MISS\_CELL OR (game.player\_target\_grid[shot\_row][shot\_col] >= 'A' AND game.player\_target\_grid[shot\_row][shot\_col] <= 'Z') THEN

DisplayMessage("You've already fired there. Try a different spot.")

Pause(2 seconds)

CONTINUE LOOP

END IF

END IF

game.missiles\_fired\_count = game.missiles\_fired\_count + 1

result = ProcessPlayerShot(game, shot\_row, shot\_col) // Assumes ProcessPlayerShot updates computer\_fleet[...].is\_sunk

SWITCH result:

CASE "MISS":

PrintLine("\*\*\*\*\* M I S S \*\*\*\*\*") // UX Note: Use color

game.player\_target\_grid[shot\_row][shot\_col] = MISS\_CELL

CASE "HIT":

PrintLine("\*\*\*\*\* H I T \*\*\*\*\*") // UX Note: Use color

game.player\_target\_grid[shot\_row][shot\_col] = HIT\_CELL

CASE "SUNK":

// Find which ship was sunk

sunk\_ship\_name = ""

FOR i FROM 0 TO MAX\_SHIPS - 1:

IF game.computer\_fleet[i].is\_sunk IS TRUE THEN

is\_target\_ship = FALSE

FOR k FROM 0 TO game.computer\_fleet[i].size - 1:

IF game.computer\_fleet[i].segments[k].row IS shot\_row AND game.computer\_fleet[i].segments[k].col IS shot\_col THEN

is\_target\_ship = TRUE; BREAK

END IF

END FOR

IF is\_target\_ship THEN

sunk\_ship\_name = game.computer\_fleet[i].name\_long

UpdateTargetGridForSunkShip(game, game.computer\_fleet[i])

// ships\_remaining\_count is decremented in ProcessPlayerShot

BREAK

END IF

END IF

END FOR

PrintLine("\*\*\*\*\* YOU SUNK THE " + sunk\_ship\_name + "! \*\*\*\*\*") // UX Note: Use color

END SWITCH

Pause(1 second) // UX: Shorter pause for faster gameplay flow, or make it configurable

END WHILE

IF game.ships\_remaining\_count IS 0 THEN

ClearScreen()

DisplayPlayerTargetGrid(game.player\_target\_grid, game.last\_shot\_coord, FALSE) // No highlight on final board

DisplayShipStatusAndStats(game) // Show final status

PrintLine("CONGRATULATIONS! You sunk all enemy ships!")

PrintLine("Total missiles fired: " + game.missiles\_fired\_count)

IF game.missiles\_fired\_count IS 17 THEN PrintLine("A PERFECT GAME!") END IF

UpdateTopScores(game.missiles\_fired\_count)

game.game\_in\_progress = FALSE

prompt\_view = AskUser("View computer's ship placements? (Y/N)")

IF prompt\_view IS 'Y' THEN DisplayComputerOceanGrid\_Revealed(game.computer\_ocean\_grid)

END IF

END FUNCTION

// VII. GAMEPLAY HELPER FUNCTIONS

FUNCTION DisplayPlayerTargetGrid(READONLY ARRAY grid, READONLY Coordinate last\_shot, last\_shot\_is\_valid AS BOOLEAN):

// UX Note: Consider using box-drawing characters for grid lines.

// UX Note: Use colors for different cell types (EMPTY\_CELL, MISS\_CELL, HIT\_CELL, Sunk Ship Letters).

// UX Note: Highlight the cell at last\_shot.row, last\_shot.col if last\_shot\_is\_valid is TRUE.

Print(" ")

FOR c FROM 0 TO GRID\_SIZE - 1: Print(NumberToLetter(c) + " ") ENDFOR; PrintLine("")

FOR r FROM 0 TO GRID\_SIZE - 1:

Print( (r+1) + " " ); IF (r+1) < 10 THEN Print(" ") ENDIF

FOR c FROM 0 TO GRID\_SIZE - 1:

char\_to\_print = grid[r][c]

IF last\_shot\_is\_valid AND r IS last\_shot.row AND c IS last\_shot.col THEN

// UX Note: Print char\_to\_print with highlight (e.g., different background color)

Print("[" + char\_to\_print + "]") // Simple text highlight

ELSE

Print(" " + char\_to\_print + " ")

END IF

END FOR

PrintLine("")

END FOR

PrintLine("-------------------------")

END FUNCTION

FUNCTION DisplayShipStatusAndStats(READONLY GameState game):

PrintLine("--- STATUS ---")

PrintLine("Missiles Fired: " + game.missiles\_fired\_count)

// PrintLine("Enemy Ships Remaining: " + game.ships\_remaining\_count) // Can be deduced from list below

FOR i FROM 0 TO MAX\_SHIPS - 1:

ship = game.computer\_fleet[i]

status\_text = "OKAY"

IF ship.is\_sunk THEN status\_text = "SUNK"

ELSE IF ship.hits\_taken > 0 THEN status\_text = "HIT (" + ship.hits\_taken + "/" + ship.size + ")"

END IF

PrintLine(ship.letter + " - " + ship.name\_long + " (" + ship.size + "): " + status\_text)

// UX Note: Use color for status\_text

END FOR

PrintLine("-------------------------")

END FUNCTION

FUNCTION GetPlayerShotInput(prompt\_message AS STRING) AS STRING:

// Logic remains same as v1

INPUT prompt\_message, shot\_str

RETURN ToUpperCase(shot\_str)

END FUNCTION

FUNCTION ParseShotCoordinates(shot\_str AS STRING) RETURNING ShotParseError error\_code, INTEGER row, INTEGER col:

// Returns error\_code, -1, -1 if invalid

row = -1; col = -1

IF Length(shot\_str) < 2 OR Length(shot\_str) > 3 THEN RETURN ShotParseError.INVALID\_FORMAT, row, col

char\_col\_char = shot\_str[0]

IF char\_col\_char < 'A' OR char\_col\_char > ('A' + GRID\_SIZE - 1) THEN RETURN ShotParseError.OUT\_OF\_BOUNDS\_COL, row, col

col = LetterToNumber(char\_col\_char)

string\_row\_part = Substring(shot\_str, 1, Length(shot\_str)-1)

parsed\_row\_val = StringToInteger(string\_row\_part) // Assume returns -1 or error if not integer

IF parsed\_row\_val IS INVALID\_INTEGER THEN RETURN ShotParseError.INVALID\_FORMAT, row, col // Check if StringToInteger failed

IF parsed\_row\_val < 1 OR parsed\_row\_val > GRID\_SIZE THEN RETURN ShotParseError.OUT\_OF\_BOUNDS\_ROW, row, col

row = parsed\_row\_val - 1

RETURN ShotParseError.NO\_ERROR, row, col

END FUNCTION

FUNCTION ProcessPlayerShot(REFERENCE GameState game, r\_shot, c\_shot) AS STRING:

// Logic mostly same as v1, but ensures ships\_remaining\_count is updated if sunk

target\_cell = game.computer\_ocean\_grid[r\_shot][c\_shot]

IF target\_cell IS EMPTY\_CELL THEN RETURN "MISS"

ELSE

ship\_hit\_letter = target\_cell

found\_ship\_index = -1

FOR i FROM 0 TO MAX\_SHIPS - 1:

IF game.computer\_fleet[i].letter IS ship\_hit\_letter THEN found\_ship\_index = i; BREAK ENDIF

END FOR

IF found\_ship\_index IS NOT -1 THEN

// To prevent double-counting hits on already hit segments if computer\_ocean\_grid changes for hits

// Assume computer\_ocean\_grid stores original ship letter until that segment specifically is processed

// A more robust way: check if player\_target\_grid already has 'H' or ship letter here.

// But the pre-check in PlayGame should catch most re-shots.

// Mark hit on computer's ocean grid to avoid re-processing same segment for 'hits\_taken'

// For instance, by changing the case or using a parallel "hit map" for segments.

// Simplification: Assume a segment gives a hit only once towards sinking.

// The `player\_target\_grid` check in `PlayGame` helps avoid re-processing.

// If a particular segment (r\_shot, c\_shot) of a ship has not yet contributed to `hits\_taken`.

// Check if this specific segment (r\_shot,c\_shot) was already processed for a hit for this ship

// This requires knowing that computer\_ocean\_grid[r\_shot][c\_shot] was NOT previously processed.

// If computer\_ocean\_grid[r\_shot][c\_shot] is still UPPERCASE, it's a "fresh" hit segment.

IF IsUpperCase(game.computer\_ocean\_grid[r\_shot][c\_shot]) THEN

game.computer\_fleet[found\_ship\_index].hits\_taken = game.computer\_fleet[found\_ship\_index].hits\_taken + 1

game.computer\_ocean\_grid[r\_shot][c\_shot] = ToLowerCase(ship\_hit\_letter) // Mark as hit on computer's map

END IF

IF game.computer\_fleet[found\_ship\_index].hits\_taken >= game.computer\_fleet[found\_ship\_index].size THEN

IF game.computer\_fleet[found\_ship\_index].is\_sunk IS FALSE THEN

game.computer\_fleet[found\_ship\_index].is\_sunk = TRUE

game.ships\_remaining\_count = game.ships\_remaining\_count - 1

RETURN "SUNK"

ELSE

RETURN "HIT" // Hitting an already sunk ship part

END IF

ELSE

RETURN "HIT"

END IF

ELSE

RETURN "ERROR"

END IF

END IF

END FUNCTION

FUNCTION UpdateTargetGridForSunkShip(REFERENCE GameState game, READONLY Ship sunk\_ship):

// Logic remains same as v1

FOR i FROM 0 TO sunk\_ship.size - 1:

seg\_row = sunk\_ship.segments[i].row; seg\_col = sunk\_ship.segments[i].col

game.player\_target\_grid[seg\_row][seg\_col] = sunk\_ship.letter

END FOR

END FUNCTION

FUNCTION DisplayComputerOceanGrid\_Revealed(READONLY ARRAY grid):

PrintLine("--- COMPUTER'S SHIP PLACEMENTS ---")

// UX Note: Use colors for different ships

Print(" ")

FOR c FROM 0 TO GRID\_SIZE - 1: Print(NumberToLetter(c) + " ") ENDFOR; PrintLine("")

FOR r FROM 0 TO GRID\_SIZE - 1:

Print( (r+1) + " " ); IF (r+1) < 10 THEN Print(" ") ENDIF

FOR c FROM 0 TO GRID\_SIZE - 1:

Print(" " + grid[r][c] + " ") // Print computer's grid as is (may contain lowercase hit markers)

END FOR

PrintLine("")

END FOR

PrintLine("----------------------------------")

INPUT "Press Enter to continue..."

END FUNCTION

// VIII. GAME STATE PERSISTENCE FUNCTIONS (SaveGameState, LoadGameState - same as v1)

FUNCTION SaveGameState(READONLY GameState game): /\* ... \*/ END FUNCTION

FUNCTION LoadGameState(REFERENCE GameState game) AS BOOLEAN: /\* ... \*/ END FUNCTION

// IX. SCORING FUNCTIONS

FUNCTION ViewTopScores():

ClearScreen()

PrintLine("--- TOP 10 SCORES ---")

scores AS ARRAY[10] OF ScoreEntry

count = ReadScoresFromFile(scores)

IF count IS 0 THEN PrintLine("No scores recorded yet.")

ELSE

FOR i FROM 0 TO count - 1:

PrintLine((i+1) + ". " + scores[i].player\_name + " - " + scores[i].score\_value + " missiles (" + scores[i].date\_time\_achieved + ")")

END FOR

END IF

PrintLine("---------------------")

INPUT "Press Enter to return to menu..."

END FUNCTION

FUNCTION UpdateTopScores(new\_score\_value AS INTEGER):

scores AS ARRAY[11] OF ScoreEntry

count = ReadScoresFromFile(scores)

IF count < 10 OR new\_score\_value < scores[count-1].score\_value OR (count > 0 AND new\_score\_value < scores[count-1].score\_value) THEN

INPUT "You made the Top 10! Enter your name (3 chars, e.g., ACE): ", player\_name

// Truncate/validate name

new\_entry AS ScoreEntry

new\_entry.player\_name = Substring(player\_name, 0, 3) // Ensure 3 chars

new\_entry.score\_value = new\_score\_value

new\_entry.date\_time\_achieved = GetCurrentDateTimeString()

IF count < 10 THEN scores[count] = new\_entry; count = count + 1

ELSE scores[9] = new\_entry

END IF

SortScores(scores, count)

WriteScoresToFile(scores, MIN(count, 10))

DisplayMessage("Your score has been recorded!")

ViewTopScores()

ELSE

DisplayMessage("Good game! Not quite a top 10 score this time.")

END IF

END FUNCTION

FUNCTION ReadScoresFromFile(REFERENCE ARRAY scores\_array) AS INTEGER:

// Logic to read name, score, date\_time per line

// ... (similar to v1 but includes date\_time\_achieved)

// Ensure SortScores is called

num\_scores = 0

// Example line: ACE 25 2025-05-24\_21:30

RETURN num\_scores

END FUNCTION

FUNCTION WriteScoresToFile(READONLY ARRAY scores\_array, count AS INTEGER):

// Logic to write name, score, date\_time per line

// ... (similar to v1 but includes date\_time\_achieved)

END FUNCTION

FUNCTION SortScores(REFERENCE ARRAY scores\_array, count AS INTEGER): /\* ... same as v1 \*/ END FUNCTION

// X. UTILITY FUNCTIONS

FUNCTION GetCurrentDateTimeString() AS STRING:

// Returns current date and time as a formatted string (e.g., "YYYY-MM-DD HH:MM:SS")

RETURN "YYYY-MM-DD HH:MM" // Placeholder

END FUNCTION

// Other utility functions (RandomInteger, NumberToLetter, LetterToNumber, ClearScreen, DisplayMessage, Pause, AskUser) remain largely the same as v1.

// StringToInteger needs to be robust (e.g., indicate if conversion failed).

// IsUpperCase, ToLowerCase, Substring - standard string utilities.

*// Written by: Andrew Carling*

*// Date: 2025-05-24*

*// Course: C Programming*

*// Purpose: This program implements a simple Battleship game in C.*

*//-----------------------------------------------------------------------------*

*// I. INCLUDES AND DEFINITIONS*

*//-----------------------------------------------------------------------------*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>   *// For srand, rand, time, localtime, strftime*

#include <ctype.h>  *// For toupper, isalpha, isdigit, islower, isupper*

#include <stdbool.h> *// For bool type*

#define GRID\_SIZE 10

#define MAX\_SHIPS 5

#define MAX\_SHIP\_NAME\_LEN 50

#define MAX\_PLAYER\_NAME\_LEN 4 *// 3 chars + null terminator*

#define DATETIME\_STR\_LEN 20 *// For "YYYY-MM-DD HH:MM"*

#define SAVE\_FILE\_NAME "battleship\_save\_game.dat"

#define SCORE\_FILE\_NAME "topTenScores.txt"

*// Cell States for Grids*

#define EMPTY\_CELL '~'

#define MISS\_CELL 'M'

#define HIT\_CELL 'H'

*// Ship letters (S, A, V, E, D) will also be used.*

*// Shot Processing Results*

typedef enum {

    SHOT\_MISS,

    SHOT\_HIT,

    SHOT\_SUNK,

    SHOT\_ALREADY\_PROCESSED, *// Hit a spot that was already a hit or part of a sunk ship*

    SHOT\_ERROR

} ShotProcessResult;

*// Coordinate Parsing Results*

typedef enum {

    PARSE\_OK,

    PARSE\_ERROR\_FORMAT, *// General format issue*

    PARSE\_ERROR\_COL\_RANGE, *// Column char out of A-J range*

    PARSE\_ERROR\_ROW\_NAN, *// Row part is not a number*

    PARSE\_ERROR\_ROW\_RANGE *// Row number out of 1-10 range*

} ShotParseError;

*//-----------------------------------------------------------------------------*

*// II. DATA STRUCTURES*

*//-----------------------------------------------------------------------------*

typedef struct {

    int row;

    int col;

} Coordinate;

typedef struct {

    char name\_long[MAX\_SHIP\_NAME\_LEN];

    char letter;

    int size;

} ShipTypeInfo;

const ShipTypeInfo SHIP\_TYPES[MAX\_SHIPS] = {

    {"Seminole State Ship", 'S', 3},

    {"Air Force Academy",   'A', 5},

    {"Valencia Destroyer",  'V', 4},

    {"Eskimo University",   'E', 3},

    {"Deland High School",  'D', 2}

};

typedef struct {

    char name\_long[MAX\_SHIP\_NAME\_LEN];

    char letter;

    int size;

    int hits\_taken;

    bool is\_sunk;

    Coordinate segments[5]; *// Max ship size is 5*

} Ship;

typedef struct {

    char computer\_ocean\_grid[GRID\_SIZE][GRID\_SIZE]; *// Stores ship letters (uppercase if intact, lowercase if hit)*

    char player\_target\_grid[GRID\_SIZE][GRID\_SIZE]; *// Stores EMPTY\_CELL, MISS\_CELL, HIT\_CELL, or sunk ship letter*

    Ship computer\_fleet[MAX\_SHIPS];

    int missiles\_fired\_count;

    int ships\_remaining\_count;

    bool game\_in\_progress;

    Coordinate last\_shot\_coord;

    bool last\_shot\_valid; *// To know if last\_shot\_coord is meaningful for highlighting*

} GameState;

typedef struct {

    char player\_name[MAX\_PLAYER\_NAME\_LEN];

    int score\_value;

    char date\_time\_achieved[DATETIME\_STR\_LEN];

} ScoreEntry;

*//-----------------------------------------------------------------------------*

*// III. FUNCTION PROTOTYPES*

*//-----------------------------------------------------------------------------*

*// Menu Functions*

void **displayMainMenu**();

int **getMenuChoice**();

void **displayHelpScreen**();

*// Game Setup Functions*

void **initializeNewGame**(GameState \**game*);

void **setupComputerShips**(GameState \**game*);

bool **isValidShipPlacement**(const char *grid*[GRID\_SIZE][GRID\_SIZE], const ShipTypeInfo\* *ship\_type*, int *r*, int *c*, int *orientation*);

*// Gameplay Loop Function*

void **playGame**(GameState \**game*);

*// Gameplay Helper Functions*

void **displayPlayerTargetGrid**(const char *grid*[GRID\_SIZE][GRID\_SIZE], Coordinate *last\_shot*, bool *highlight\_last\_shot*);

void **displayShipStatusAndStats**(const GameState \**game*);

void **displayComputerOceanGrid\_Revealed**(const char *grid*[GRID\_SIZE][GRID\_SIZE]);

void **getPlayerShotInput**(char\* *buffer*, int *buffer\_size*, const char\* *prompt*);

ShotParseError **parseShotCoordinates**(const char\* *shot\_str*, int\* *r*, int\* *c*);

ShotProcessResult **processPlayerShot**(GameState \**game*, int *r\_shot*, int *c\_shot*);

void **updateTargetGridForSunkShip**(GameState \**game*, const Ship\* *sunk\_ship*);

char **numberToLetter**(int *num*);

int **letterToNumber**(char *val*);

*// Game State Persistence Functions*

bool **saveGameState**(const GameState \**game*);

bool **loadGameState**(GameState \**game*);

*// Scoring Functions*

void **viewTopScores**();

void **updateTopScores**(int *new\_score\_value*);

int **readScoresFromFile**(ScoreEntry *scores\_array*[], int *max\_scores*);

void **writeScoresToFile**(const ScoreEntry *scores\_array*[], int *count*);

void **sortScores**(ScoreEntry *scores\_array*[], int *count*);

void **getCurrentDateTimeString**(char\* *buffer*, int *buffer\_size*);

*// Utility Functions*

void **clearScreen**();

void **pauseForKey**(const char\* *message*);

void **safeGets**(char \**buffer*, int *size*);

*//-----------------------------------------------------------------------------*

*// IV. MAIN FUNCTION*

*//-----------------------------------------------------------------------------*

int **main**() {

    GameState current\_game;

    current\_game.game\_in\_progress = false;

    current\_game.last\_shot\_valid = false;

    bool running = true;

    int choice;

**srand**(**time**(NULL)); *// Seed random number generator once*

**printf**("Welcome to Battleship!\n");

**pauseForKey**("Press Enter to continue to the Main Menu...");

    while (running) {

**displayMainMenu**();

        choice = **getMenuChoice**();

        switch (choice) {

            case 1: *// Start New Game*

**initializeNewGame**(&*current\_game*);

**playGame**(&*current\_game*);

                break;

            case 2: *// Resume Game*

                if (**loadGameState**(&*current\_game*)) {

**printf**("Game resumed.\n");

**pauseForKey**("Press Enter to start playing...");

**playGame**(&current\_game);

                } else {

**printf**("No saved game found or error loading.\n");

**pauseForKey**("Press Enter to start a new game instead...");

**initializeNewGame**(&current\_game);

**playGame**(&current\_game);

                }

                break;

            case 3: *// View Top 10 Scores*

**viewTopScores**();

                break;

            case 4: *// How to Play*

**displayHelpScreen**();

                break;

            case 5: *// Quit*

                if (current\_game.game\_in\_progress) {

                    char save\_prompt[10];

**printf**("A game is currently in progress.\n");

**printf**("Save current game before quitting? (Y/N): ");

**safeGets**(save\_prompt, sizeof(save\_prompt));

                    if (**toupper**(save\_prompt[0]) == 'Y') {

                        if (**saveGameState**(&current\_game)) {

**printf**("Game saved.\n");

                        } else {

**printf**("Error saving game.\n");

                        }

                    }

                }

                running = false;

**printf**("Exiting game. Goodbye!\n");

                break;

            default:

**printf**("Invalid choice. Please try again.\n");

**pauseForKey**(NULL);

                break;

        }

    }

    return 0;

}

*//-----------------------------------------------------------------------------*

*// V. MENU FUNCTIONS*

*//-----------------------------------------------------------------------------*

void **displayMainMenu**() {

**clearScreen**();

**printf**("=======================================\n");

**printf**("    B A T T L E S H I P    \n");

**printf**("=======================================\n\n");

**printf**("MAIN MENU\n");

**printf**("---------------------------------------\n");

**printf**("1. Start New Game\n");

**printf**("2. Resume Game\n");

**printf**("3. View Top 10 Scores\n");

**printf**("4. How to Play\n");

**printf**("5. Quit Game\n");

**printf**("---------------------------------------\n");

}

int **getMenuChoice**() {

    char input[10];

    int choice = 0;

**printf**("Enter your choice (1-5): ");

**safeGets**(input, sizeof(input));

    if (**sscanf**(input, "%d", &choice) == 1 && choice >= 1 && choice <= 5) {

        return choice;

    }

    return 0; *// Invalid choice*

}

void **displayHelpScreen**() {

**clearScreen**();

**printf**("-----------------------------------------------------------------\n");

**printf**("                       HOW TO PLAY BATTLESHIP                    \n");

**printf**("-----------------------------------------------------------------\n");

**printf**("OBJECTIVE:\n");

**printf**("  Be the first to sink all 5 of the computer's hidden ships.\n\n");

**printf**("THE FLEET (Name, Letter on Grid when Sunk, Size):\n");

    for (int i = 0; i < MAX\_SHIPS; ++i) {

**printf**("  - %-20s (%c) - %d holes\n", SHIP\_TYPES[i].name\_long, SHIP\_TYPES[i].letter, SHIP\_TYPES[i].size);

    }

**printf**("\nGAMEPLAY:\n");

**printf**("  1. On your turn, call out a shot by entering coordinates (e.g., A5, J10).\n");

**printf**("  2. The grid will update with the result of your shot:\n");

**printf**("     '%c' : Empty water (already shot or initial state)\n", EMPTY\_CELL);

**printf**("     '%c' : Miss\n", MISS\_CELL);

**printf**("     '%c' : Hit on a ship (that is not yet sunk)\n", HIT\_CELL);

**printf**("     'S,A,V,E,D': Indicates a segment of that specific sunk ship.\n");

**printf**("  3. A ship is sunk when all its segments have been hit.\n");

**printf**("  4. The game ends when all 5 ships are sunk.\n\n");

**printf**("SCORING:\n");

**printf**("  Try to use the fewest missiles possible. A perfect game uses 17 missiles.\n");

**printf**("  Your score (missiles fired) might make the Top 10 list!\n\n");

**printf**("SAVING/LOADING:\n");

**printf**("  You can save your game progress if you need to quit and resume later.\n");

**printf**("-----------------------------------------------------------------\n");

**pauseForKey**(NULL);

}

*//-----------------------------------------------------------------------------*

*// VI. GAME SETUP FUNCTIONS*

*//-----------------------------------------------------------------------------*

void **initializeNewGame**(GameState \**game*) {

    for (int r = 0; r < GRID\_SIZE; ++r) {

        for (int c = 0; c < GRID\_SIZE; ++c) {

*game*->player\_target\_grid[r][c] = EMPTY\_CELL;

*game*->computer\_ocean\_grid[r][c] = EMPTY\_CELL;

        }

    }

    for (int i = 0; i < MAX\_SHIPS; ++i) {

**strcpy**(*game*->computer\_fleet[i].name\_long, SHIP\_TYPES[i].name\_long);

*game*->computer\_fleet[i].letter = SHIP\_TYPES[i].letter;

*game*->computer\_fleet[i].size = SHIP\_TYPES[i].size;

*game*->computer\_fleet[i].hits\_taken = 0;

*game*->computer\_fleet[i].is\_sunk = false;

    }

*game*->missiles\_fired\_count = 0;

*game*->ships\_remaining\_count = MAX\_SHIPS;

*game*->game\_in\_progress = true;

*game*->last\_shot\_valid = false;

**setupComputerShips**(*game*);

**printf**("New game initialized. The computer has secretly placed its ships.\n");

**pauseForKey**("Press Enter to begin...");

}

void **setupComputerShips**(GameState \**game*) {

    for (int i = 0; i < MAX\_SHIPS; ++i) {

        const ShipTypeInfo\* current\_ship\_type = &SHIP\_TYPES[i];

        bool placed\_successfully = false;

        int attempts = 0;

        while (!placed\_successfully && attempts < 1000) {

            int start\_row = **rand**() % GRID\_SIZE;

            int start\_col = **rand**() % GRID\_SIZE;

            int orientation = **rand**() % 2;

            if (**isValidShipPlacement**(*game*->computer\_ocean\_grid, current\_ship\_type, start\_row, start\_col, orientation)) {

                for (int j = 0; j < current\_ship\_type->size; ++j) {

                    int r = start\_row;

                    int c = start\_col;

                    if (orientation == 0) {

                        c += j;

*game*->computer\_ocean\_grid[r][c] = current\_ship\_type->letter;

*game*->computer\_fleet[i].segments[j].row = r;

*game*->computer\_fleet[i].segments[j].col = c;

                    } else {

                        r += j;

*game*->computer\_ocean\_grid[r][c] = current\_ship\_type->letter;

*game*->computer\_fleet[i].segments[j].row = r;

*game*->computer\_fleet[i].segments[j].col = c;

                    }

                }

                placed\_successfully = true;

            }

            attempts++;

        }

        if (!placed\_successfully) {

**fprintf**(stderr, "Warning: Could not place ship %s optimally after %d attempts. Game might be unplayable.\n", current\_ship\_type->name\_long, attempts);

        }

    }

}

bool **isValidShipPlacement**(const char *grid*[GRID\_SIZE][GRID\_SIZE], const ShipTypeInfo\* *ship\_type*, int *r\_start*, int *c\_start*, int *orientation*) {

    if (*orientation* == 0) {

        if (*c\_start* < 0 || *r\_start* < 0 || *c\_start* + *ship\_type*->size > GRID\_SIZE || *r\_start* >= GRID\_SIZE) return false;

    } else {

        if (*r\_start* < 0 || *c\_start* < 0 || *r\_start* + *ship\_type*->size > GRID\_SIZE || *c\_start* >= GRID\_SIZE) return false;

    }

    for (int i = 0; i < *ship\_type*->size; ++i) {

        int r = *r\_start*;

        int c = *c\_start*;

        if (*orientation* == 0) c += i; else r += i;

        if (*grid*[r][c] != EMPTY\_CELL) return false;

    }

    return true;

}

*//-----------------------------------------------------------------------------*

*// VII. GAMEPLAY LOOP FUNCTION*

*//-----------------------------------------------------------------------------*

void **playGame**(GameState \**game*) {

    char shot\_input\_str[10];

    int shot\_row, shot\_col;

    ShotParseError parse\_err;

    while (*game*->ships\_remaining\_count > 0 && *game*->game\_in\_progress) {

**clearScreen**();

**displayPlayerTargetGrid**(*game*->player\_target\_grid, *game*->last\_shot\_coord, *game*->last\_shot\_valid);

**displayShipStatusAndStats**(*game*);

**printf**("Enter 'quit' to return to main menu.\n");

**getPlayerShotInput**(shot\_input\_str, sizeof(shot\_input\_str), "Your command (e.g., A5 or quit): ");

        if (**strcmp**(shot\_input\_str, "quit") == 0 || **strcmp**(shot\_input\_str, "QUIT") == 0) {

            char save\_prompt[10];

**printf**("Are you sure you want to quit this game session?\n");

**printf**("Save current game before returning to menu? (Y/N): ");

**safeGets**(save\_prompt, sizeof(save\_prompt));

            if (**toupper**(save\_prompt[0]) == 'Y') {

                 if(**saveGameState**(*game*)) **printf**("Game saved.\n"); else **printf**("Error saving game.\n");

            }

*game*->game\_in\_progress = false;

**pauseForKey**("Returning to Main Menu...");

            return;

        }

        parse\_err = **parseShotCoordinates**(shot\_input\_str, &shot\_row, &shot\_col);

*game*->last\_shot\_valid = false;

        if (parse\_err != PARSE\_OK) {

            switch (parse\_err) {

                case PARSE\_ERROR\_FORMAT: **printf**("Error: Invalid coordinate format. Use LetterNumber (e.g., A5, J10).\n"); break;

                case PARSE\_ERROR\_COL\_RANGE: **printf**("Error: Column out of range. Must be A-J.\n"); break;

                case PARSE\_ERROR\_ROW\_NAN: **printf**("Error: Row must be a number.\n"); break;

                case PARSE\_ERROR\_ROW\_RANGE: **printf**("Error: Row out of range. Must be 1-10.\n"); break;

                default: **printf**("Error: Unknown coordinate parsing error.\n"); break;

            }

**pauseForKey**(NULL);

            continue;

        }

*game*->last\_shot\_coord.row = shot\_row;

*game*->last\_shot\_coord.col = shot\_col;

*game*->last\_shot\_valid = true;

        if (*game*->player\_target\_grid[shot\_row][shot\_col] != EMPTY\_CELL &&

*game*->player\_target\_grid[shot\_row][shot\_col] != HIT\_CELL) {

**printf**("You've already conclusively fired at %s (%c). Try a different spot.\n", shot\_input\_str, *game*->player\_target\_grid[shot\_row][shot\_col]);

**pauseForKey**(NULL);

            continue;

        }

*game*->missiles\_fired\_count++;

        ShotProcessResult result = **processPlayerShot**(*game*, shot\_row, shot\_col);

        char result\_message[100] = "";

        switch (result) {

            case SHOT\_MISS:

**sprintf**(result\_message, "\*\*\*\*\* M I S S \*\*\*\*\*");

*game*->player\_target\_grid[shot\_row][shot\_col] = MISS\_CELL;

                break;

            case SHOT\_HIT:

**sprintf**(result\_message, "\*\*\*\*\* H I T ! \*\*\*\*\*");

*game*->player\_target\_grid[shot\_row][shot\_col] = HIT\_CELL;

                break;

            case SHOT\_SUNK:

                for (int i = 0; i < MAX\_SHIPS; ++i) {

                    if (*game*->computer\_fleet[i].is\_sunk) {

                        bool this\_ship\_hit = false;

                        for(int k=0; k < *game*->computer\_fleet[i].size; ++k) {

                            if(*game*->computer\_fleet[i].segments[k].row == shot\_row && *game*->computer\_fleet[i].segments[k].col == shot\_col) {

                                this\_ship\_hit = true; break;

                            }

                        }

                        if(this\_ship\_hit) {

**sprintf**(result\_message, "\*\*\*\*\* YOU SUNK THE %s! (%c) \*\*\*\*\*", *game*->computer\_fleet[i].name\_long, *game*->computer\_fleet[i].letter);

**updateTargetGridForSunkShip**(*game*, &*game*->computer\_fleet[i]);

                            break;

                        }

                    }

                }

                break;

            case SHOT\_ALREADY\_PROCESSED:

**sprintf**(result\_message, "You already hit that spot. It's part of a ship (%c).", *game*->player\_target\_grid[shot\_row][shot\_col]);

                break;

            case SHOT\_ERROR:

**sprintf**(result\_message, "Error processing shot. Please report this.");

                break;

        }

**printf**("\n%s\n", result\_message);

**pauseForKey**("Press Enter for next turn or results...");

    }

    if (*game*->ships\_remaining\_count == 0) {

**clearScreen**();

**displayPlayerTargetGrid**(*game*->player\_target\_grid, *game*->last\_shot\_coord, false);

**displayShipStatusAndStats**(*game*);

**printf**("\n====================================================\n");

**printf**("    CONGRATULATIONS! You sunk all enemy ships!    \n");

**printf**("====================================================\n");

**printf**("Total missiles fired: %d\n", *game*->missiles\_fired\_count);

        if (*game*->missiles\_fired\_count == 17) {

**printf**("A PERFECT GAME! You used the minimum possible missiles!\n");

        }

**updateTopScores**(*game*->missiles\_fired\_count);

*game*->game\_in\_progress = false;

        char view\_prompt[10];

**printf**("\nWould you like to see the computer's ship placements? (Y/N): ");

**safeGets**(view\_prompt, sizeof(view\_prompt));

        if (**toupper**(view\_prompt[0]) == 'Y') {

**displayComputerOceanGrid\_Revealed**(*game*->computer\_ocean\_grid);

        }

    }

**pauseForKey**("Press Enter to return to the Main Menu...");

}

*//-----------------------------------------------------------------------------*

*// VIII. GAMEPLAY HELPER FUNCTIONS*

*//-----------------------------------------------------------------------------*

void **displayPlayerTargetGrid**(const char *grid*[GRID\_SIZE][GRID\_SIZE], Coordinate *last\_shot*, bool *highlight\_last\_shot*) {

**printf**("\nYOUR TARGET GRID:\n");

**printf**("  |");

    for (int c = 0; c < GRID\_SIZE; ++c) {

**printf**(" %c |", **numberToLetter**(c));

    }

**printf**("\n");

**printf**("  +");

    for (int c = 0; c < GRID\_SIZE; ++c) **printf**("---+"); **printf**("\n");

    for (int r = 0; r < GRID\_SIZE; ++r) {

**printf**("%2d|", r + 1);

        for (int c = 0; c < GRID\_SIZE; ++c) {

            char display\_char = *grid*[r][c];

            bool is\_last\_shot = *highlight\_last\_shot* && r == *last\_shot*.row && c == *last\_shot*.col;

            if (is\_last\_shot) **printf**("[%c]", display\_char);

            else **printf**(" %c ", display\_char);

**printf**("|");

        }

**printf**("\n");

**printf**("  +");

        for (int c = 0; c < GRID\_SIZE; ++c) **printf**("---+"); **printf**("\n");

    }

**printf**("---------------------------------------\n");

}

void **displayShipStatusAndStats**(const GameState \**game*) {

**printf**("\nGAME STATUS:\n");

**printf**("---------------------------------------\n");

**printf**("Missiles Fired: %d\n", *game*->missiles\_fired\_count);

**printf**("Enemy Fleet Status:\n");

    for (int i = 0; i < MAX\_SHIPS; ++i) {

        const Ship\* ship = &*game*->computer\_fleet[i];

        char status\_str[30];

        if (ship->is\_sunk) {

**sprintf**(status\_str, "SUNK");

        } else if (ship->hits\_taken > 0) {

**sprintf**(status\_str, "HIT (%d/%d)", ship->hits\_taken, ship->size);

        } else {

**sprintf**(status\_str, "Undamaged");

        }

**printf**("  (%c) %-20s : %s\n", ship->letter, ship->name\_long, status\_str);

    }

**printf**("---------------------------------------\n");

}

void **displayComputerOceanGrid\_Revealed**(const char *grid*[GRID\_SIZE][GRID\_SIZE]) {

**clearScreen**();

**printf**("\nCOMPUTER'S SECRET OCEAN GRID (Revealed):\n");

**printf**("  |");

    for (int c = 0; c < GRID\_SIZE; ++c) {

**printf**(" %c |", **numberToLetter**(c));

    }

**printf**("\n");

**printf**("  +");

    for (int c = 0; c < GRID\_SIZE; ++c) **printf**("---+"); **printf**("\n");

    for (int r = 0; r < GRID\_SIZE; ++r) {

**printf**("%2d|", r + 1);

        for (int c = 0; c < GRID\_SIZE; ++c) {

            char cell\_content = *grid*[r][c];

**printf**(" %c ", cell\_content);

**printf**("|");

        }

**printf**("\n");

**printf**("  +");

        for (int c = 0; c < GRID\_SIZE; ++c) **printf**("---+"); **printf**("\n");

    }

**printf**("---------------------------------------\n");

**pauseForKey**("This was the computer's setup. Press Enter to continue...");

}

void **getPlayerShotInput**(char\* *buffer*, int *buffer\_size*, const char\* *prompt*) {

**printf**("%s", *prompt*);

**safeGets**(*buffer*, *buffer\_size*);

    if (**strcmp**(*buffer*, "quit") != 0 && **strcmp**(*buffer*, "QUIT") != 0) {

        for (int i = 0; *buffer*[i]; i++) {

*buffer*[i] = **toupper**(*buffer*[i]);

        }

    }

}

ShotParseError **parseShotCoordinates**(const char\* *shot\_str*, int\* *r*, int\* *c*) {

    if (*shot\_str* == NULL || **strlen**(*shot\_str*) < 2 || **strlen**(*shot\_str*) > 3) return PARSE\_ERROR\_FORMAT;

    char char\_col\_upper = **toupper**(*shot\_str*[0]);

    if (!**isalpha**(char\_col\_upper)) return PARSE\_ERROR\_FORMAT;

    if (char\_col\_upper < 'A' || char\_col\_upper > ('A' + GRID\_SIZE - 1)) return PARSE\_ERROR\_COL\_RANGE;

    \**c* = **letterToNumber**(char\_col\_upper);

    int parsed\_row\_val = 0;

    int num\_parsed\_items = 0;

    if (**strlen**(*shot\_str*) == 2) {

        if (!**isdigit**(*shot\_str*[1])) return PARSE\_ERROR\_ROW\_NAN;

        num\_parsed\_items = **sscanf**(*shot\_str* + 1, "%1d", &parsed\_row\_val);

    } else if (**strlen**(*shot\_str*) == 3) {

         if (!**isdigit**(*shot\_str*[1]) || !**isdigit**(*shot\_str*[2])) return PARSE\_ERROR\_ROW\_NAN;

        num\_parsed\_items = **sscanf**(*shot\_str* + 1, "%2d", &parsed\_row\_val);

    } else {

        return PARSE\_ERROR\_FORMAT;

    }

    if (num\_parsed\_items != 1) return PARSE\_ERROR\_ROW\_NAN;

    if (parsed\_row\_val < 1 || parsed\_row\_val > GRID\_SIZE) return PARSE\_ERROR\_ROW\_RANGE;

    \**r* = parsed\_row\_val - 1;

    return PARSE\_OK;

}

ShotProcessResult **processPlayerShot**(GameState \**game*, int *r\_shot*, int *c\_shot*) {

    if (*game*->player\_target\_grid[*r\_shot*][*c\_shot*] == MISS\_CELL ||

        (**isupper**(*game*->player\_target\_grid[*r\_shot*][*c\_shot*]) && *game*->player\_target\_grid[*r\_shot*][*c\_shot*] != EMPTY\_CELL && *game*->player\_target\_grid[*r\_shot*][*c\_shot*] != HIT\_CELL) ) {

        return SHOT\_ALREADY\_PROCESSED;

    }

    char target\_on\_computer\_grid = *game*->computer\_ocean\_grid[*r\_shot*][*c\_shot*];

    if (target\_on\_computer\_grid == EMPTY\_CELL) {

        return SHOT\_MISS;

    } else if (**islower**(target\_on\_computer\_grid)) {

        return SHOT\_ALREADY\_PROCESSED;

    } else if (**isupper**(target\_on\_computer\_grid)) {

        char ship\_hit\_letter = target\_on\_computer\_grid;

        int found\_ship\_index = -1;

        for (int i = 0; i < MAX\_SHIPS; ++i) {

            if (*game*->computer\_fleet[i].letter == ship\_hit\_letter) {

                found\_ship\_index = i;

                break;

            }

        }

        if (found\_ship\_index != -1) {

*game*->computer\_fleet[found\_ship\_index].hits\_taken++;

*game*->computer\_ocean\_grid[*r\_shot*][*c\_shot*] = **tolower**(ship\_hit\_letter);

            if (*game*->computer\_fleet[found\_ship\_index].hits\_taken >= *game*->computer\_fleet[found\_ship\_index].size) {

                if (!*game*->computer\_fleet[found\_ship\_index].is\_sunk) {

*game*->computer\_fleet[found\_ship\_index].is\_sunk = true;

*game*->ships\_remaining\_count--;

                    return SHOT\_SUNK;

                } else {

                    return SHOT\_ALREADY\_PROCESSED;

                }

            } else {

                return SHOT\_HIT;

            }

        } else {

**fprintf**(stderr, "Error: Hit a non-empty, non-lowercase cell ('%c') on computer grid that doesn't map to a known ship letter.\n", target\_on\_computer\_grid);

            return SHOT\_ERROR;

        }

    }

**fprintf**(stderr, "Error: Unhandled case in processPlayerShot for cell '%c'.\n", target\_on\_computer\_grid);

    return SHOT\_ERROR;

}

void **updateTargetGridForSunkShip**(GameState \**game*, const Ship\* *sunk\_ship*) {

    for (int i = 0; i < *sunk\_ship*->size; ++i) {

        int r = *sunk\_ship*->segments[i].row;

        int c = *sunk\_ship*->segments[i].col;

*game*->player\_target\_grid[r][c] = *sunk\_ship*->letter;

    }

}

char **numberToLetter**(int *num*) {

    if (*num* >= 0 && *num* < 26) return 'A' + *num*;

    return '?';

}

int **letterToNumber**(char *val*) {

    return **toupper**(*val*) - 'A';

}

*//-----------------------------------------------------------------------------*

*// IX. GAME STATE PERSISTENCE FUNCTIONS*

*//-----------------------------------------------------------------------------*

bool **saveGameState**(const GameState \**game*) {

    FILE \*file = **fopen**(SAVE\_FILE\_NAME, "wb");

    if (file == NULL) {

**perror**("Error opening save file for writing");

        return false;

    }

    size\_t written = **fwrite**(*game*, sizeof(GameState), 1, file);

**fclose**(file);

    if (written != 1) {

**fprintf**(stderr, "Error: Failed to write complete game state to save file.\n");

        return false;

    }

    return true;

}

bool **loadGameState**(GameState \**game*) {

    FILE \*file = **fopen**(SAVE\_FILE\_NAME, "rb");

    if (file == NULL) {

        return false;

    }

    size\_t read\_count = **fread**(*game*, sizeof(GameState), 1, file);

**fclose**(file);

    if (read\_count != 1) {

**fprintf**(stderr, "Error: Failed to read complete game state from save file or file corrupted.\n");

        return false;

    }

*game*->game\_in\_progress = true;

    return true;

}

*//-----------------------------------------------------------------------------*

*// X. SCORING FUNCTIONS*

*//-----------------------------------------------------------------------------*

void **viewTopScores**() {

**clearScreen**();

**printf**("--- TOP 10 SCORES ---\n");

    ScoreEntry scores[10];

    int count = **readScoresFromFile**(scores, 10);

    if (count == 0) {

**printf**("No scores recorded yet. Be the first!\n");

    } else {

**printf**("Rank | Name | Score (Missiles) | Date Achieved\n");

**printf**("-----|------|------------------|--------------------\n");

        for (int i = 0; i < count; ++i) {

**printf**("%-4d | %-4s | %-16d | %s\n", i + 1, scores[i].player\_name, scores[i].score\_value, scores[i].date\_time\_achieved);

        }

    }

**printf**("------------------------------------------------------\n");

**pauseForKey**(NULL);

}

void **updateTopScores**(int *new\_score\_value*) {

    ScoreEntry scores[11];

    int count = **readScoresFromFile**(scores, 10);

    bool qualifies = false;

    if (count < 10) {

        qualifies = true;

    } else if (*new\_score\_value* < scores[9].score\_value) {

        qualifies = true;

    }

    if (qualifies) {

        char player\_name\_input[MAX\_PLAYER\_NAME\_LEN + 10];

**printf**("\nCongratulations! You've made the Top 10 high scores!\n");

        do {

**printf**("Enter your initials (3 characters, e.g., ACE): ");

**safeGets**(player\_name\_input, sizeof(player\_name\_input));

            if (**strlen**(player\_name\_input) != 3) {

**printf**("Error: Initials must be exactly 3 characters. Please try again.\n");

            }

        } while (**strlen**(player\_name\_input) != 3);

        ScoreEntry new\_entry;

**strncpy**(new\_entry.player\_name, player\_name\_input, MAX\_PLAYER\_NAME\_LEN -1);

        new\_entry.player\_name[MAX\_PLAYER\_NAME\_LEN - 1] = '\0';

        new\_entry.score\_value = *new\_score\_value*;

**getCurrentDateTimeString**(new\_entry.date\_time\_achieved, DATETIME\_STR\_LEN);

        if (count < 10) {

            scores[count] = new\_entry;

            count++;

        } else {

            scores[9] = new\_entry;

        }

**sortScores**(scores, count);

**writeScoresToFile**(scores, count);

**printf**("Your score has been recorded!\n");

**viewTopScores**();

    } else {

**printf**("Good game! Your score of %d missiles was not quite enough for the Top 10 this time.\n", *new\_score\_value*);

    }

}

int **readScoresFromFile**(ScoreEntry *scores\_array*[], int *max\_scores\_to\_read*) {

    FILE \*file = **fopen**(SCORE\_FILE\_NAME, "r");

    if (file == NULL) {

        return 0;

    }

    int num\_scores = 0;

    while (num\_scores < *max\_scores\_to\_read* &&

**fscanf**(file, "%3s %d %19[^\n]",

*scores\_array*[num\_scores].player\_name,

                  &*scores\_array*[num\_scores].score\_value,

*scores\_array*[num\_scores].date\_time\_achieved) == 3) {

*scores\_array*[num\_scores].player\_name[MAX\_PLAYER\_NAME\_LEN - 1] = '\0';

*scores\_array*[num\_scores].date\_time\_achieved[DATETIME\_STR\_LEN - 1] = '\0';

        num\_scores++;

    }

**fclose**(file);

**sortScores**(*scores\_array*, num\_scores);

    return num\_scores;

}

void **writeScoresToFile**(const ScoreEntry *scores\_array*[], int *count*) {

    FILE \*file = **fopen**(SCORE\_FILE\_NAME, "w");

    if (file == NULL) {

**perror**("Error: Could not write scores to file");

        return;

    }

    for (int i = 0; i < *count*; ++i) {

**fprintf**(file, "%s %d %s\n", *scores\_array*[i].player\_name, *scores\_array*[i].score\_value, *scores\_array*[i].date\_time\_achieved);

    }

**fclose**(file);

}

void **sortScores**(ScoreEntry *scores\_array*[], int *count*) {

    ScoreEntry temp;

    for (int i = 0; i < *count* - 1; ++i) {

        for (int j = 0; j < *count* - i - 1; ++j) {

            if (*scores\_array*[j].score\_value > *scores\_array*[j + 1].score\_value) {

                temp = *scores\_array*[j];

*scores\_array*[j] = *scores\_array*[j + 1];

*scores\_array*[j + 1] = temp;

            }

        }

    }

}

void **getCurrentDateTimeString**(char\* *buffer*, int *buffer\_size*) {

    time\_t now = **time**(NULL);

    struct tm \*tm\_info = **localtime**(&now);

**strftime**(*buffer*, *buffer\_size*, "%Y-%m-%d %H:%M", tm\_info);

}

*//-----------------------------------------------------------------------------*

*// XI. UTILITY FUNCTIONS*

*//-----------------------------------------------------------------------------*

void **clearScreen**() {

    #ifdef \_WIN32

**system**("cls");

    #else

        system("clear");

    #endif

}

void **pauseForKey**(const char\* *message*) {

    if (*message* != NULL && **strlen**(*message*) > 0) {

**printf**("%s\n", *message*);

    } else {

**printf**("Press Enter to continue...");

    }

    int c;

*// Try to consume any leftover characters from previous input line*

*// This is particularly important after a safeGets that might not have read the newline*

*// if the input was exactly buffer\_size-1 characters long.*

    bool newline\_found\_in\_buffer = false;

    while ((c = **getchar**()) != '\n' && c != EOF) {

        newline\_found\_in\_buffer = true; *// Consumed something other than just a newline*

    }

*// If the buffer was empty and only a newline was pending from a previous Enter,*

*// the above loop might not execute, and 'c' would be '\n'.*

*// If getchar() returned EOF, something is wrong, but we proceed.*

}

void **safeGets**(char \**buffer*, int *size*) {

    if (**fgets**(*buffer*, *size*, stdin) != NULL) {

        size\_t len = **strlen**(*buffer*);

        if (len > 0 && *buffer*[len - 1] == '\n') {

*buffer*[len - 1] = '\0';

        } else {

            if (len == *size* -1 && *buffer*[len-1] != '\n') { *// Buffer was filled without a newline*

                int c;

                while ((c = **getchar**()) != '\n' && c != EOF); *// Clear rest of stdin*

            }

        }

    } else {

*buffer*[0] = '\0';

**clearerr**(stdin);

    }

}

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer menu

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.