DATA LAYER

namespace DataLayer

{

public partial class Players

{

public Players(string username, string password)

{

this.username = username;

this.password = password;

}

}

public partial class Games

{

public Games(string title, bool complete, string creatorFK, string opponentFK)

{

this.title = title;

this.complete = complete;

this.creatorFK = creatorFK;

this.opponentFK = opponentFK;

}

}

public partial class Attacks

{

public Attacks(string coordinate, bool hit, int gameFK)

{

this.coordinate = coordinate;

this.hit = hit;

this.gameFK = gameFK;

}

}

public partial class Ships

{

public Ships(string title, int size)

{

this.title = title;

this.size = size;

}

}

public partial class GameShipConfigurations

{

public GameShipConfigurations(int gameFK, string playerFK, string coordinate)

{

this.gameFK = gameFK;

this.playerFK = playerFK;

this.coordinate = coordinate;

}

}

}

BUSINESS LOGIC LAYER

using DataLayer;

namespace BusinessLogicLayer

{

public class BusinessLogic

{

public static BattleshipsDBEntities db = new BattleshipsDBEntities();

public void AddPlayer(string username, string password) // add player to database

{

Players player = new Players(username, password); // create new player object and add to database

db.Players.Add(player);

db.SaveChanges();

}

public int GetPlayerUsernameCount(string username) // get count of players with username

{

var playerCount = (from player in db.Players

where player.username == username

select player).Count();

return playerCount;

}

public string GetPlayerPassword(string username) // get password of player

{

var playerPassword = (from player in db.Players

where player.username == username

select player.password).First();

return playerPassword;

}

public void AddGame(string title, bool complete, string creatorFK, string opponentFK) // add game to database

{

Games game = new Games(title, complete, creatorFK, opponentFK); // create new game object and add to database

db.Games.Add(game);

db.SaveChanges();

}

public void UpdateGameComplete(int gameFK) // update game complete

{

var game = (from g in db.Games

where g.id == gameFK

select g).First();

game.complete = true;

db.SaveChanges();

}

public int GetLatestGameId() // get id of game

{

var gameID = (from game in db.Games

select game.id).Max();

return gameID;

}

public int GetShipSize(int id) // get size of ship

{

var shipSize = (from ship in db.Ships

where ship.id == id

select ship.size).First();

return (int)shipSize;

}

public void AddGameShipConfiguration(string coordinate, string playerFK, int gameFK) // add game ship configuration to database

{

GameShipConfigurations gameShipConfiguration = new GameShipConfigurations(gameFK, playerFK, coordinate); // create new game ship configuration object and add to database

db.GameShipConfigurations.Add(gameShipConfiguration);

db.SaveChanges();

}

public List<string> GetGameShipConfigurationCoordinates(int gameFK, string playerFK) // get coordinates of game ship configuration

{

var gameShipConfigurationCoordinates = (from gameShipConfiguration in db.GameShipConfigurations

where gameShipConfiguration.gameFK == gameFK && gameShipConfiguration.playerFK == playerFK

select gameShipConfiguration.coordinate).ToList();

return gameShipConfigurationCoordinates;

}

public void AddAttack(string coordinate, bool hit, int gameFK) // add attack to database

{

Attacks attack = new Attacks(coordinate, hit, gameFK); // create new attack object and add to database

db.Attacks.Add(attack);

db.SaveChanges();

}

}

}

PRESENTATION LAYER

using BusinessLogicLayer;

namespace PresentationLayer

{

public class Presentation

{

BusinessLogic bl = new BusinessLogic();

bool activeGame = false;

bool shipsConfigured = false;

string[] users = new string[2];

public void ShowMenu() // show the menu

{

try

{

Console.WriteLine("1.Add Player details");

Console.WriteLine("2.Configure Ships");

Console.WriteLine("3.Launch Attack");

Console.WriteLine("4.Quit");

Console.Write("\nChoice: ");

string choice = Console.ReadLine().Trim();

switch (choice)

{

case "1":

if (!activeGame)

AddPlayer();

else

Console.WriteLine("Game already active\n");

break;

case "2":

if (activeGame && !shipsConfigured)

ConfigureShips();

else if (activeGame && shipsConfigured)

Console.WriteLine("Ships already configured\n");

else

Console.WriteLine("No active game\n");

/\*

\* For testing purposes:

\*

\* User 1

\* a1,a2

\* b1,b2,b3

\* c1,c2,c3

\* d1,d2,d3,d4

\* e1,e2,e3,e4,e5

\*

\* User 2

\* h7,g7

\* h6,g6,f6

\* h5,g5,f5

\* h4,g4,f4,e4

\* h3,g3,f3,e3,d3

\*/

break;

case "3":

if (shipsConfigured)

LaunchAttack();

else

Console.WriteLine("Ships not configured\n");

break;

case "4":

Console.WriteLine("Thanks For Playing");

Environment.Exit(1);

break;

default:

Console.WriteLine("Invalid choice\n");

break;

}

}

catch (Exception e)

{

Console.WriteLine($"\nAn unexpected error occurred: {e.Message}");

Console.WriteLine("Returning to the main menu and clearing entered values...");

resetGame();

Console.ReadLine(); // Pause before showing the menu again

Console.Clear();

}

}

public void AddPlayer() // add player details

{

Console.Clear();

for (int i = 1; i < 3; i++)

{

Console.WriteLine("Add Player {0}'s details", i);

Console.Write("Username: ");

string username = Console.ReadLine();

if (bl.GetPlayerUsernameCount(username) == 0) // if username doesn't exist

{

Console.Write("Password: ");

string password = PasswordMask();

bl.AddPlayer(username, password); // add player to database

Console.WriteLine("\nUser Created");

users[i - 1] = username;

Console.WriteLine();

}

else // if username exists

{

while (true)

{

Console.Write("Enter password for {0}: ", username);

string password = PasswordMask();

if (bl.GetPlayerPassword(username) == password) // if password is correct

{

Console.WriteLine("\nLogin Successful");

users[i - 1] = username;

break;

}

else // if password is incorrect

{

Console.WriteLine("\nLogin Failed. Try Again");

}

}

Console.WriteLine();

}

}

Console.Write("Enter Game Title: ");

string gameTitle = Console.ReadLine();

AddGame(gameTitle, users[0], users[1]);

}

public static string PasswordMask() // hides the password input

{

string password = "";

while (true)

{

ConsoleKeyInfo keyInfo = Console.ReadKey(true); // true hides the input

if (keyInfo.Key != ConsoleKey.Enter && keyInfo.Key != ConsoleKey.Spacebar && keyInfo.Key != ConsoleKey.Backspace)

{

password += keyInfo.KeyChar; // add the key to the password string

Console.Write("\*");

}

else if (keyInfo.Key == ConsoleKey.Spacebar || keyInfo.Key == ConsoleKey.Backspace) // if spacebar

continue; // ignore the spacebar and backspace

else

break;

}

return password;

}

public void AddGame(string title, string user1, string user2) // add game details

{

bl.AddGame(title, false, user1, user2);

Console.WriteLine("Game Created. Press enter to continue.\n");

Console.ReadLine();

Console.Clear();

activeGame = true;

}

public void ConfigureShips() // configure ships

{

int destroyerSize = bl.GetShipSize(1);

int submarineSize = bl.GetShipSize(2);

int cruiserSize = bl.GetShipSize(3);

int battleshipSize = bl.GetShipSize(4);

int carrierSize = bl.GetShipSize(5);

for (int i = 1; i < 3; i++) // for each player

{

Console.Clear();

List<int> placedShips = new List<int>();

List<string> placedShipCoordinates = new List<string>();

while (true)

{

Console.WriteLine($"Configure Ships for {users[i - 1]} \n");

PrintGrid(placedShipCoordinates);

Console.WriteLine("\nSelect Ship (Enter Number):\n");

Console.WriteLine($"1.Destroyer ({destroyerSize} spaces)");

Console.WriteLine($"2.Submarine ({submarineSize} spaces)");

Console.WriteLine($"3.Cruiser ({cruiserSize} spaces)");

Console.WriteLine($"4.Battleship ({battleshipSize} spaces)");

Console.WriteLine($"5.Carrier ({carrierSize} spaces)");

Console.Write("\nChoice: ");

int choice;

try

{

choice = Convert.ToInt32(Console.ReadLine().Trim());

}

catch (Exception e)

{

Console.WriteLine("\nInvalid choice. Try again\n");

continue;

}

if (choice != 1 && choice != 2 && choice != 3 && choice != 4 && choice != 5) // if ship doesn't exist

{

Console.WriteLine("\nInvalid choice. Ship does not exist. Try again\n");

continue;

}

if (placedShips.Contains(choice)) // if ship is already placed

{

Console.WriteLine("\nInvalid choice. Ship may have been already placed. Try again\n");

continue;

}

while (true) // get coordinates

{

Console.Write("Enter Coordinate (ex: B3,C3,D3 or G2,G3,G4,G5): ");

string coordinates = Console.ReadLine().ToUpper();

string[] coordinateArray = coordinates.Split(',');

if ((choice == 1 && coordinateArray.Length != destroyerSize) // if coordinates do not match the ship size

|| (choice == 2 && coordinateArray.Length != submarineSize)

|| (choice == 3 && coordinateArray.Length != cruiserSize)

|| (choice == 4 && coordinateArray.Length != battleshipSize)

|| (choice == 5 && coordinateArray.Length != carrierSize))

{

Console.WriteLine("\nInvalid coordinates. Try again");

continue;

}

bool validCoordinates = true;

foreach (string coordinate in coordinateArray) // check if coordinates are valid

{

if (coordinate.Length != 2 || coordinate[0] < 'A' || coordinate[0] > 'H' || coordinate[1] < '1' || coordinate[1] > '7') // if coordinate is invalid

{

Console.WriteLine("\nInvalid coordinates. Try again");

validCoordinates = false;

break; // Exit the foreach loop if any coordinate is invalid

}

if (placedShipCoordinates.Contains(coordinate)) // if coordinate is already used

{

Console.WriteLine("\nOne or more coordinate points are already used. Try again");

validCoordinates = false;

break; // Exit the foreach loop if any coordinate is already used

}

placedShipCoordinates.Add(coordinate); // add coordinate to list

bl.AddGameShipConfiguration(coordinate, users[i - 1], bl.GetLatestGameId());

}

if (validCoordinates)

break; // Break out of the while loop if all coordinates are valid

}

Console.WriteLine("\nShip added\n");

placedShips.Add(choice); // add ship to list

if (placedShips.Count() == 5) // if all ships are placed

{

Console.WriteLine($"All ships placed, press enter to continue. {users[i - 1]}'s Ships: \n");

PrintGrid(placedShipCoordinates);

Console.ReadLine();

Console.Clear();

break;

}

}

}

shipsConfigured = true;

}

public void LaunchAttack() // launch attack

{

List<string> user1AttackCoordinates = new List<string>();

int user1HitCount = 0;

List<string> user2AttackCoordinates = new List<string>();

int user2HitCount = 0;

while (true)

{

user1HitCount = LaunchAttackOnPlayer(user2AttackCoordinates, users[1], user1HitCount); // launch attack on player 2

if (user1HitCount == 17) // if player 1 wins

{

bl.UpdateGameComplete(bl.GetLatestGameId());

Console.WriteLine("Player 1 Wins. Press enter to go back to the menu.");

Console.ReadLine();

resetGame();

Console.Clear();

break;

}

user2HitCount = LaunchAttackOnPlayer(user1AttackCoordinates, users[0], user2HitCount); // launch attack on player 1

if (user2HitCount == 17) // if player 2 wins

{

bl.UpdateGameComplete(bl.GetLatestGameId());

Console.WriteLine("Player 2 Wins. Press enter to go back to the menu.");

Console.ReadLine();

resetGame();

Console.Clear();

break;

}

}

}

public void PrintGrid(List<string> Coordinates)

{

Console.WriteLine(" A B C D E F G H");

for (int row = 0; row < 7; row++)

{

Console.Write(row + 1 + " ");

for (int col = 0; col < 8; col++)

{

string currentCoordinate = $"{(char)('A' + col)}{row + 1}"; // get the current coordinates, adds 1 to the ascii value of A to get the correct letter

if (Coordinates.Contains(currentCoordinate) && (shipsConfigured == false))

{

Console.Write("[#]");

}

else if (Coordinates.Contains(currentCoordinate + "H") && (shipsConfigured == true))

{

Console.Write("[X]");

}

else if (Coordinates.Contains(currentCoordinate + "M") && (shipsConfigured == true))

{

Console.Write("[O]");

}

else

{

Console.Write("[ ]");

}

}

Console.WriteLine();

}

}

public int LaunchAttackOnPlayer(List<string> userAttackCoordinates, string user, int userHitCount) // launch attack on player

{

Console.Clear();

Console.WriteLine($"Launch Attack on {user}\n");

PrintGrid(userAttackCoordinates);

while (true) // get coordinates

{

Console.Write("\nEnter Coordinate (ex: B3): ");

string coordinate = Console.ReadLine().ToUpper();

if (coordinate.Length != 2 || coordinate[0] < 'A' || coordinate[0] > 'H' || coordinate[1] < '1' || coordinate[1] > '7') // if coordinate is invalid

{

Console.WriteLine("\nInvalid coordinates. Try again");

continue;

}

if (userAttackCoordinates.Contains(coordinate + "H") || userAttackCoordinates.Contains(coordinate + "M")) // if coordinate is already used

{

Console.WriteLine("\nCoordinate is already used. Try again");

continue;

}

List<string> userPlacedShips = bl.GetGameShipConfigurationCoordinates(bl.GetLatestGameId(), user);

if (userPlacedShips.Contains(coordinate)) // if coordinate is a ship

{

Console.WriteLine("\nHit. Press enter to continue.");

userAttackCoordinates.Add(coordinate + "H");

bl.AddAttack(coordinate, true, bl.GetLatestGameId());

userHitCount++;

Console.ReadLine();

}

else // if coordinate is not a ship

{

Console.WriteLine("\nMiss. Press enter to continue");

userAttackCoordinates.Add(coordinate + "M");

bl.AddAttack(coordinate, false, bl.GetLatestGameId());

Console.ReadLine();

}

break;

}

return userHitCount;

}

public void resetGame() // reset game

{

activeGame = false;

shipsConfigured = false;

users[0] = "";

users[1] = "";

}

}

#region AA1.4 - Classes

public class GameScreen

{

private List<Cell> Cells;

public GameScreen(List<Cell> cells)

{

Cells = cells;

}

public void PrintGrid()

{

Console.WriteLine(" A B C D E F G H");

for (int row = 0; row < 7; row++)

{

Console.Write(row + 1 + " ");

for (int col = 0; col < 8; col++)

{

}

Console.WriteLine();

}

}

}

public abstract class Cell

{

//common properties for both a cell showing a ship or a successful/failed attack

//e.g. public int GridCellNo;

public int Coordinate { get; set; }

//this has to be implemented differently in the next two classes

//for a cell representing a ship or cell representing a successful/failed attack

//e.g if it is a ship, on screen print S, if it is a successful attack print X, if it is a failed attack print O

public abstract void PrintCell();

}

public class ShipCell : Cell

{

public override void PrintCell()

{

Console.Write("[#]"); // Representing a ship

}

}

public class AttackCell : Cell

{

public bool IsHit { get; set; } // Indicates if the attack was successful

public AttackCell(bool isHit)

{

IsHit = isHit;

}

public override void PrintCell()

{

Console.Write(IsHit ? "[X]" : "[O]"); // 'X' for hit, 'O' for miss

}

}

#endregion AA1.4 - Classes

}

MAIN

using PresentationLayer;

namespace Main

{

public class Program

{

static void Main(string[] args)

{

Presentation pt = new Presentation();

while (true)

{

pt.ShowMenu();

}

}

}

}