# Introduction

This document describes some of the logic and settings used in the application. The solution is created using the .Net core 5.0.

The final API is deployed in the below URL.

API - <https://battleshipstatetracker.azurewebsites.net/BattleShip/0/0/Player1>

Swagger - <https://battleshipstatetracker.azurewebsites.net/swagger/index.html>

## Application Settings

* xCordinate / yCordinate – The board size in x and y direction. 9 to 9 is the default value. This means a board with 100 units will be prepared. 0 to 9 in x direction and 0 to 9 in y direction.
* BoardName – Default name for the board "Board 1".
* DefaultPlayerName – “Player1”. This is the default player name for the single player game.
* ShipSize – 5. This means that a ship will occupy 5 units in the board.
* ShipDirection – Horizontal. Which direction the ship will be allocated in the board. This can be either Horizontal/Vertical. Currently logic is return for horizontal but the code is extendable for vertical as well.
* NumberOfShips - 10. Number of ships that will be allocated to the board during start-up.

## Board Specifications

A 10x10 board will have the below coordinates when initialised.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0,0 | 1,0 | 2,0 | 3,0 | 4,0 | 5,0 | 6,0 | 7,0 | 8,0 | 9,0 |
| 0,1 | 1,1 | 2,1 | 3,1 | 4,1 | 5,1 | 6,1 | 7,1 | 8,1 | 9,1 |
| 0,2 | 1,2 | 2,2 | 3,2 | 4,2 | 5,2 | 6,2 | 7,2 | 8,2 | 9,2 |
| 0,3 | 1,3 | 2,3 | 3,3 | 4,3 | 5,3 | 6,3 | 7,3 | 8,3 | 9,3 |
| 0,4 | 1,4 | 2,4 | 3,4 | 4,4 | 5,4 | 6,4 | 7,4 | 8,4 | 9,4 |
| 0,5 | 1,5 | 2,5 | 3,5 | 4,5 | 5,5 | 6,5 | 7,5 | 8,5 | 9,5 |
| 0,6 | 1,6 | 2,6 | 3,6 | 4,6 | 5,6 | 6,6 | 7,6 | 8,6 | 9,6 |
| 0,7 | 1,7 | 2,7 | 3,7 | 4,7 | 5,7 | 6,7 | 7,7 | 8,7 | 9,7 |
| 0,8 | 1,8 | 2,8 | 3,8 | 4,8 | 5,8 | 6,8 | 7,8 | 8,8 | 9,8 |
| 0,9 | 1,9 | 2,9 | 3,9 | 4,9 | 5,9 | 6,9 | 7,9 | 8,9 | 9,9 |

The board with initial 10 ships allocated to the default player with

* Ship size 5 (1 ship occupies 5 units).
* Allocation direction is horizontal.

Will look like below. 10 x 5 = 50 units are occupied.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0,0 | 1,0 | 2,0 | 3,0 | 4,0 | 5,0 | 6,0 | 7,0 | 8,0 | 9,0 |
| 0,1 | 1,1 | 2,1 | 3,1 | 4,1 | 5,1 | 6,1 | 7,1 | 8,1 | 9,1 |
| 0,2 | 1,2 | 2,2 | 3,2 | 4,2 | 5,2 | 6,2 | 7,2 | 8,2 | 9,2 |
| 0,3 | 1,3 | 2,3 | 3,3 | 4,3 | 5,3 | 6,3 | 7,3 | 8,3 | 9,3 |
| 0,4 | 1,4 | 2,4 | 3,4 | 4,4 | 5,4 | 6,4 | 7,4 | 8,4 | 9,4 |
| 0,5 | 1,5 | 2,5 | 3,5 | 4,5 | 5,5 | 6,5 | 7,5 | 8,5 | 9,5 |
| 0,6 | 1,6 | 2,6 | 3,6 | 4,6 | 5,6 | 6,6 | 7,6 | 8,6 | 9,6 |
| 0,7 | 1,7 | 2,7 | 3,7 | 4,7 | 5,7 | 6,7 | 7,7 | 8,7 | 9,7 |
| 0,8 | 1,8 | 2,8 | 3,8 | 4,8 | 5,8 | 6,8 | 7,8 | 8,8 | 9,8 |
| 0,9 | 1,9 | 2,9 | 3,9 | 4,9 | 5,9 | 6,9 | 7,9 | 8,9 | 9,9 |

The logic lets to allocate any number of ships of any size, provided the maximum of size and number is not more than the total unit count. The logic checks for space in a raw for allocation etc already.

## Start-up Logic

Below are the main sections of code responsible for preloading all objects.

1. Startup.cs

services.AddSingleton<IPlayerProvider, PlayerProvider>(provider =>{

UnitProvider unitProvider = new(Configuration);

BoardProvider boardProvider = new(unitProvider);

ShipProvider shipProvider = new(Configuration);

Board board = boardProvider.CreateBoard(1, Configuration["GameSettings:BoardName"].ToString());

board = shipProvider.CreateShips(int.Parse(Configuration["GameSettings:ShipSettings:NumberOfShips"].ToString()), board);

return new PlayerProvider(board, Configuration);

});

* *boardProvider.CreateBoard creates a board with 10x10 units and with other default values.*
* shipProvider.CreateShips creates and allocates ships to the board.

1. Constructor of PlayerProvider.cs

public PlayerProvider(Board board, IConfiguration configuration)

{

Board = board;

Configuration = configuration;

//Initialize Primary player.

//This is only since we are initializing on startup.

Name = Configuration["GameSettings:DefaultPlayerName"].ToString();

CreatePlayer(Name);

}

* Creates a default player while starting up and adds it to the List<Player> which is an in-memory list.

The in-memory implementation can be easily replaced with a persistent one with ease because of the use of interfaces.