



Battleship!

Many computer scientists create software agents that are able to play “games” in an automated fashion: no human interaction is required. These agents can be part of a video games where a human player competes against a computer player as a form of entertainment. But in other applications, the “game” is a model or simulation that is used by scientists to solve problems: everything from analyzing traffic patterns to microbes in a petri dish to herd behavior. We are going to create a

simple agent that play the classic board game Battleship and then let all our agents compete against each other. This project will help us to become more proficient with:

- Creating projects with MS Visual Studio
- Designing and implementing software
- Practicing with C# syntax
- Debugging programs
- Creating simple classes
- Creating a simple “AI-like” algorithm
- Interacting with external programs
- Keeping track of long-term state

Program Description:

For this project, you are not going to be writing a program. The program has already been written and provided to you—it is the game Battleship. Rather, you are writing a library that the Battleship Program will import and use. Your library will create a Battleship Agent that is able to play the game. It plays the game by implementing these *five* methods:

public override string GetNickname() – This function returns a nickname that identifies your agent. The nickname should be unique but does not need to identify you personally (e.g., Fuzzy Wuzzy).

public override string SetOpponentsName(string opponent) – a function that tells your agent the nickname of the agent you are playing against. You can ignore this information or

public override BattleshipFleet PositionFleet() – This function is called once at the beginning of the game and allows your agent to position its own fleet on the board. Your agent will return an object that lists the top-left coordinate of each ship and whether the ship is to be oriented vertically or horizontally. Keep in mind that the top-left corner of the board is (0, 0), the bottom-left (0, 9), and the top-right (9, 0).

public override GridSquare LaunchAttack() – Returns a GridSquare object which contains the x- and y-coordinates of your attack. The boardgame is 10-by-10 squares, and so your function should return two numbers that are both between 0 and 9.

public override void DamageReport(char report) – After launching an attack, the Battleship Program will tell your agent whether the attack was a miss or whether it hit one of your opponent’s ships. The value of report will be one of the values from the ShipType struct: it will either be Carrier “C”, Battleship “B”, Destroyer “D”, Submarine “S”, or PatrolBoat “P”. To make use of this information, you will need to match it up to x- and y-coordinates of your last attack.

We will have a game tournament the week before finals in order to let all of the agents from class compete against each other. *This means there is a hard deadline for when this project is due and that late submissions will not be accepted.*

General Hints:

Your agent should keep track of the grid squares that it has already attacked and not waste any turns by sending multiple shots to the same coordinate.

A higher-quality agent will take note of whether a particular attack has hit an opponent's ship and then repeatedly fire in the general area until that ship has been sunk.

Your agent should position your fleet in a way that is difficult for your opponents to guess. If all your ships are clumped together, then your agent may be easier to defeat.

Rubric

The 10 points for this project will be earned according to the following rubric:

- 1 point – Source code uploaded to Blackboard according to the following naming format: `2019_Fall_FirstLast_Battleship.cs` (where "FirstLast" is your first and last name).
- 1 point – Library compiles without any modifications by the instructor.
- 1 point – Library implements the `GetNickname()` function (which provides a unique name!)
- 1 point – Library implements the `LaunchAttack()` function
- 1 point – Library implements the `DamageReport()` function
- 1 point – Library implements the `PositionFleet()` function
- 1 point – Your agent successfully plays multiple games of Battleship
- 1 point – Instructor's discretionary points to assign based on the quality of the code
- 1 point – Your agent is able to defeat the instructor's simple agent, Bozo the Clown, at least 7 out of 10 times. Bozo is a relatively basic agent who fires randomly.
- 1 point – You present your agent's algorithm to the class

Extra credit will be available for agents that perform particularly well:

Professional -Level (+1 point extra credit):

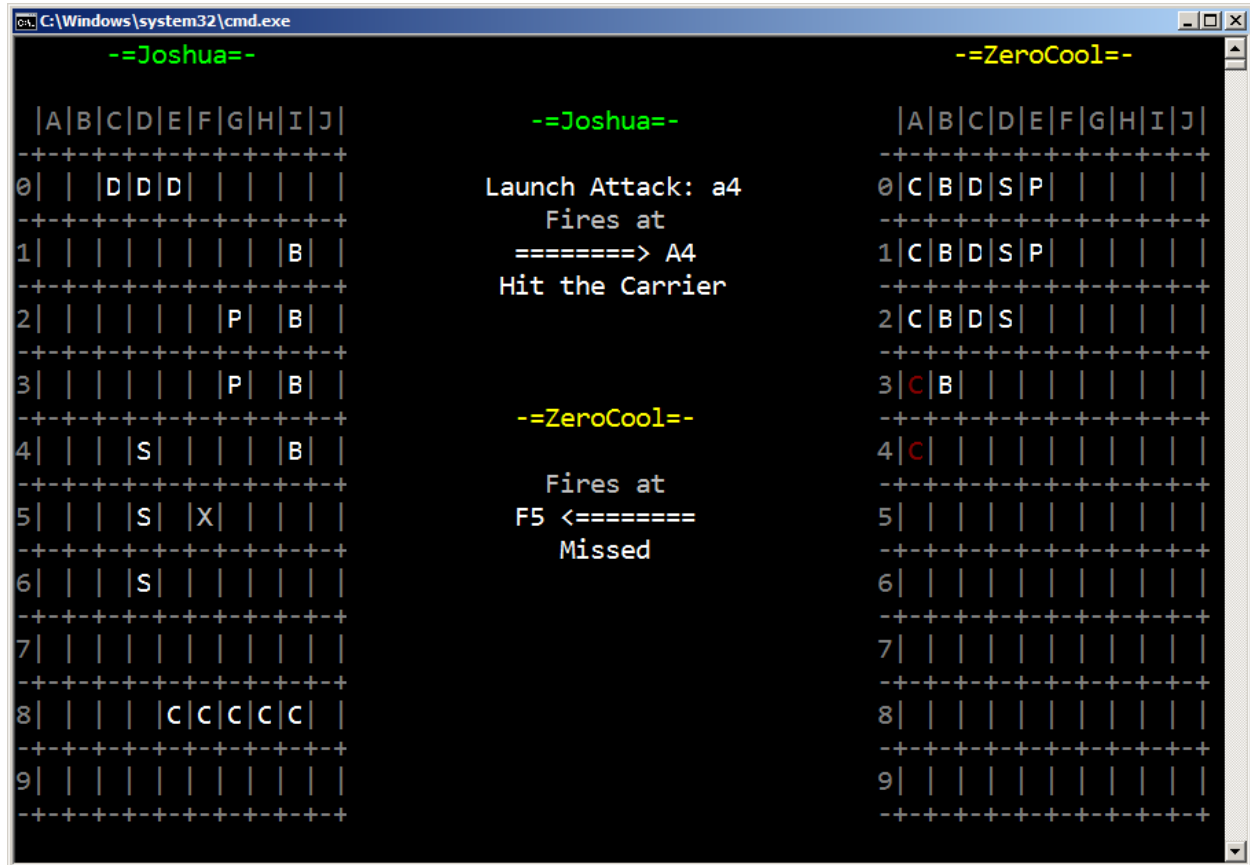
Your agent wins more games of Battleship than it loses when playing against your classmates. Your agent is likely to play upwards of 3000 games total (approximately half of the class should earn this extra credit point).

L33t Hacker-Level (+1 point extra credit):

Your agent finishes the game tournament in the top-three (e.g., three students will earn this point).

Example Output

When run, the existing Battleship Program will display the human's board on the left side and your agent's board on the right side. It will prompt the human player to enter the attack coordinates. The agent that you have written will automatically provide the coordinates for the counterattack.



```
C:\Windows\system32\cmd.exe

--Joshua--

|A|B|C|D|E|F|G|H|I|J|
-+-+---+
0| |D|D|D| | | | |
-+-+---+
1| | | | | | |B| |
-+-+---+
2| | | | |P|B| |
-+-+---+
3| | | | |P|B| |
-+-+---+
4| | |S| | |B| |
-+-+---+
5| | |S|X| | | |
-+-+---+
6| | |S| | | | |
-+-+---+
7| | | | | | | |
-+-+---+
8| | |C|C|C|C|C| |
-+-+---+
9| | | | | | | |
-+-+---+

--ZeroCool--

|A|B|C|D|E|F|G|H|I|J|
-+-+---+
0|C|B|D|S|P| | | |
-+-+---+
1|C|B|D|S|P| | | |
-+-+---+
2|C|B|D|S| | | | |
-+-+---+
3|C|B| | | | | | |
-+-+---+
4|C| | | | | | | |
-+-+---+
5| | | | | | | | |
-+-+---+
6| | | | | | | | |
-+-+---+
7| | | | | | | | |
-+-+---+
8| | | | | | | | |
-+-+---+
9| | | | | | | | |
-+-+---+

Launch Attack: a4
Fires at
=====> A4
Hit the Carrier

--ZeroCool--

Fires at
F5 <=====
Missed
```

The game will keep running until one player sinks the other player's fleet or until 100 turns have been completed (which finishes the game in a tie).

Note: This Battleship Program does not allow you, the human player, to choose the position of your fleet. It always places the human ships in the same location. It would be naïve for your agent to assume that your fellow classmates will place their ships in the exact same location.

Note: The position of both fleets will be visible to anyone who is watching the program. This allows you, as the human player, to verify that your agent is positioning its fleet correctly. It also allows you to quickly sink your agent's ships, should you so desire. It does not mean that your agent is able to see the screen. Your agent is playing "blind" like a real player would be.

Programming Style:

This course does not have a required coding style (at least not yet). Your code should be nicely arranged with proper whitespace, along with appropriate and consistently named variables/functions. Any code that is executed repeatedly should be put into its own function. This makes it easier for you and others to maintain the code.

Homework Submissions:

Please upload your source code to Blackboard. There should only be a single .cs file for this project. You'll find the upload page by clicking "Project #3: Battleship" on the Assignments page. I will cut-and-paste your code into a scratch project in order to review it, compile it, and test it.

Class Presentations:

You will need to present your code to the class. You must describe the algorithm that your agent uses to place ships and choose attack coordinates. You will have a maximum of 2-3 minutes to present.

Important Parts of the Battleship Engine

Your agent can make use of the following definitions, which are part of the BattleshipAgent library:

```
namespace Battleship
{
    public abstract class BattleshipAgent
    {
        /// <summary>
        /// The Battleship Game Engine will call this function to get the nickname
        /// of your Battleship agent.
        /// </summary>
        /// <returns>a string representing the name of this agent</returns>
        public abstract string GetNickname();

        /// <summary>
        /// Tells your agent the name of your opponent.
        /// </summary>
        /// <param name="opponent"></param>
        public abstract void SetOpponent(string opponent);

        /// <summary>
        /// The Battleship Game Engine invokes this method at the beginning of the
        /// game in order to get the position of
        /// all five ships in the sea. You provide the upper-left coordinate of
        /// each ship and whether it should be
        /// oriented vertically or horizontally. You must make sure that the ships
        /// do not overlap with each other.
```

```

    /// </summary>
    /// <returns>an object that holds the location of all five ships</returns>
    public abstract BattleshipFleet PositionFleet();

    /// <summary>
    /// The Battleship Game Engine runs this subroutine to learn where this
    /// agent would like to direct its next shot.
    /// The game board is a 0-based 10x10 grid.
    /// </summary>
    /// <returns>the x- and y-coordinates of the square being
    /// attacked</returns>
    public abstract GridSquare LaunchAttack();

    /// <summary>
    /// The Battleship Game Engine calls this function to tell the agent
    /// whether their shot hit their opponents fleet,
    /// and if so, which ship it hit.
    /// </summary>
    /// <param name="report">the ship ('C', 'B', 'D', ...) that was hit or
    /// '\0' for a miss</param>
    public abstract void DamageReport(char report);
}

public class BattleshipFleet
{
    public ShipPosition PatrolBoat;
    public ShipPosition Submarine;
    public ShipPosition Destroyer;
    public ShipPosition Battleship;
    public ShipPosition Carrier;
}

public struct GridSquare
{
    public int x;
    public int y;

    public GridSquare(int x = 0, int y = 0)
    {
        this.x = x;
        this.y = y;
    }

    public override string ToString()
    {
        return $"({x}, {y})";
    }
}

public struct ShipType
{
    public const char None = '\0';
    public const char Carrier = 'C';
    public const char Battleship = 'B';
    public const char Destroyer = 'D';
    public const char Submarine = 'S';
    public const char PatrolBoat = 'P';
}

```

```

public static string Name(char shipType)
{
    switch (char.ToUpper(shipType))
    {
        case ShipType.Carrier:
            return "Carrier";
        case ShipType.Battleship:
            return "Battleship";
        case ShipType.Destroyer:
            return "Destroyer";
        case ShipType.Submarine:
            return "Submarine";
        case ShipType.PatrolBoat:
            return "Patrol Boat";
    }

    return "Unknown/Error";
}

public struct ShipRotation
{
    public const char Horizontal = 'H';
    public const char Vertical = 'V';
}

public struct ShipPosition
{
    public char rotation;
    public int left;
    public int top;

    public ShipPosition(int left=0, int top=0, char
                        rotation=ShipRotation.Horizontal)
    {
        this.left = left;
        this.top = top;
        this.rotation = rotation;
    }

    public override string ToString()
    {
        return $"({left}, {top}, {rotation})";
    }
}
}

```