Battleship

You’ve gained significant experience with the C# language and WPF technology. It’s time to create a fully featured application that showcases all your newfound skills. And what better way can we accomplish this than to recreate a classic family board game about naval warfare? Well, we could recreate a board game about being a property baron and slum lord, but I’d rather focus on a game where we simply blow up our friends rather than ruin those friendships for eternity.

You will create a GUI application that has all the features of the classic board game Battleship. The user (aka player) will be able to start a new game (or load an existing one), place ships, fire shots, and ultimately win or lose by their own cunning and/or luck! At any time during a game, the player can save the current state to continue play at a later time.

## Requirements

The requirements for this application are simple. You will review the standard rules and board layout of the official Battleship game and build your application to those specifications. Additionally, you will add a few features that only a digital version could possess.

1. At startup, the player may choose to start a new game or load an existing one.
2. Starting a new game presents the player with their 10 x 10 board (appropriately labeled with alphanumeric headings for ease of use). Your grid should show clear grid lines and each cell in the grid should have a color representing its current state.
   1. Blue = Water (unexplored; default state of all cells)
   2. White = Miss
   3. Red = Hit
   4. Grey = Ship
3. The player then places ships according to the standard rules. Your application must prevent illegal placement choices. Once all five ships are placed legally, the player indicates they are ready to start the game.
4. Behind the scenes, the computer (aka the AI opponent) will also place ships legally on its own grid, hidden from the player, of course. Once all ships are placed by both the player and AI, the game officially starts.
   1. Note: The computer should be placing ships in a fashion that results in a different layout each game. It should not have its ship placement hardcoded so that the layout is identical every game.
5. Once the game starts, the player is presented with two 10 x 10 grids. One grid is the player’s containing the ships as they were placed by the player. The other grid is to track their shots against their AI opponent.
6. The player is considered “Player 1” and goes first. The game continues based on the standard rules of the board game.
7. For ease of grading, you must include a “cheat mode” that allows grader to know where the AI placed their ships. This is PURELY for testing purposes.
   1. Implement a custom hotkey command so you won’t have to make a physical, easy-to-find button. Just make sure it doesn’t conflict with standard hotkeys (like Ctrl+S for Save, or Ctrl+O for Open, etc.) and include any special instructions in a comment on the LMS with your submission.
   2. The grader should be able to toggle cheat mode on and off to reveal/conceal the AI ships. By default, cheat mode is off.
8. Note that while your AI may make random shot choices, all choices must be legal (in bounds, unique, and only during the AI’s turn).
   1. You may take this opportunity to build a smarter AI that can make strategic decisions based on previous shot results. However, your AI is not allowed to “cheat” (meaning it cannot make choices based on knowing where the player placed their ships).
9. At ANY point after starting a new game (but before the game is over), the player should be able to save their current progress. This should save the state of the entire game.
   1. Save files will have the extension .bshp.
   2. The save/load dialog box must filter based on this extension.
10. Once either the player or the AI loses their final ship, the winner should be announced and the player should be given the opportunity to either start a new game or load an existing one.
11. The GUI must be designed to be intuitive, with clear and correct labels and values throughout. Ships should have their names spelled correctly and must match those designated in the standard rules.
12. You may use images and other visual/audio elements as seem appropriate to add polish to your application. However, don’t be garish or… go overboard \*Dad pun’d\*!!

# Rubric

**Automatic Zero:** Your deliverable is not a GUI, lacks a cheat mode, or is otherwise ungradable.

(10 points) – Player may start a new game

(15 points) – Save works correctly

(15 points) – Load works correctly

(15 points) – Player may only place ships legally (in bounds, acceptable orientations, prior to shots fired)

(15 points) – AI may only place ships legally (same as player)

(20 points) – Ship/Shot grids are correctly displayed (10 x 10 with correct column/row labels)

(10 points) – Grid cells display the correct state based on ship placement and shots taken

(15 points) – Shots can only be made by the player or opponent when it is respectively their turn

(15 points) – Player can only take legal shots

(15 points) – AI can only take legal shots

(10 points) – Turn taking logic works correctly

(10 points) – Winner is correctly determined

(5 points) – After declaring a winner, player may start or load another game

(30 points) – GUI is well designed and intuitive to use (proper labeling, visual flow, etc)

Bonus Points:

(5 points) – Add a 1-second splash screen

(5 points) – Add a custom icon to the window (instead of the default icon)

(5 points) – Add a custom icon to be displayed in the task bar (instead of the default icon)

(10 points) – Add appropriate sound effects

(10 points) – Add appropriate music

(15 points) – Add at least 3 different and appropriate animations

(25 points) – Make an intelligent AI that uses previous shot results to make future shot choices