Game description:

1.

We have two fields(seas). The first belongs to the user and the second is computer.

When game starts, computer randomly locates ships on the field, so that they were either horizontal, or vertical positioned.

The size of fields are 7x7(vertically from A to G and horizontally, from 0 to 6).

We have overall 6 ships for each player, from which 3 are small(1 square), 2 are middle (2 squares) and 1 is big (3 squares).

We have also one field for location input for user and one field for messages related to game.

2.

User can see the locations of his ships, but he can't see the locations of computer's seas and vice versa.

Тhe logic is shooting each other by guessing the ship locations of opponent.

First starts User and then is the turn of computer (3 seconds later). If the shot was true, on the field of opponent appears "Ship" image, and if it was false on the field of opponent appears "Miss" image. If player truly guesses all the locations of opponent's ship, that ship becomes SUNK. Who first sinks all the ships of opponent, that player wins.

3. When anyone wins, the game updates itself and starts again.

Game Pseudocode:

1. We create an object (variable-view) that is responsible for updating display corresponding to the logic of game.
   1. We add to the object view a Method – updateMessage, that receives a message and updates the message area in HTML.
      1. updateMessage receives as a parameter a message(msg),
      2. gets from DOM the element, in which must be the message, by it's Id and saves it in variable messageElement.
      3. sets the property innerHTML of the messageElement - received message(msg).
   2. We add to the object view a Method - displayHit, that receives a location (that is equal to field Id) and updates the hitted field (the field with received location(id)) in HTML, adding a background image (ship.png).
      1. showHit receives as a parameter a location (that is equal to field Id),
      2. gets from DOM the element, in which must be the image by it's Id and saves it in variable hitElement.
      3. sets for hitElement a new attribute(class = "hit"). (in css file we have a selector by class (.hit), that sets the background image of it's element (ship.png).
   3. We add to the object view a Method - showMiss, that receives a location (that is equal to field Id) and updates the missed field (the field with received location(id)) in HTML, adding a background image (miss.png).
      1. showMiss receives as a parameter a location (that is equal to field Id),,
      2. gets from DOM the element, in which must be the image by it's Id and saves it in variable missElement.
      3. sets for missElement a new attribute(class = "miss"). (in css file we have a selector by class (.miss), that sets the background image of it's element (miss.png).
2. We create an object (variable-model) that is responsible for the logic of the game.
   1. Object model contains 2 properties(userField and computerField) ,that, in turn, contain objects.
      1. We create the following properties in the object userField :

1. fieldSize: 7 (it means 7 rows and 7 columns),

2. shipsCount: 6 ( count of ships ),

3. ships: [{location:[""], hits:[""]},

{location:[""], hits:[""]},

{location:[""], hits:[""]},

{location:["",""], hits:["",""]},

{location:["",""], hits:["",""]},

{location:["","",""], hits:["","",""]},

],(property ships contains all ships of object userField. In fact it contains an Array, that in turn contains objects(ships), that, in turn, contains 2 properties: location, that contains an Array with locations of a ship and hits, that contains an Array with empty strings, that changes to "hit" when corresponding location is hitted.

We have overall 6 ships for each player, from which 3 are small(1 square), 2 are middle (2 squares) and 1 is big (3 squares).)

4. sunkShips: 0; (the count of sunk ships),

5. shipLength: { small:1,

medium:2,

big:3,} ( length of different kind of ships),

* + 1. We create the following methods in object userField :
       1. userField can receive the location of shooting and check, whether it is hit or not.(method fireCheck).

If it is hit, method updates ships[corresponding ship].hits and sets the corresponding element, that have the same index as shooted element, to “hit”. And then it checks if the ship sunk by calling the method isSunk, if isSunk() is true, sunkShips+=1.

And finnaly it returns true, otherwise it returns false.

* + - 1. Method isSunk - it checks a ship, if all the elements are “hit”, it returns true, otherwise it returns false.
      2. direction – the method accepts nothing ․
         1. We randomly pick a number in diapason 0 - 2 and save that number in variable dir,
         2. If dir = 0, method returns “horizontal”,
         3. Else if dir = 1 method returns “vertical”,
      3. collision – The method accepts parameter location, and checks, if there is a match with the ships’ locations or not.
         1. It picks each element of ships,

Then it picks each element of ship.location ,

If element of ship.locations = location,

Method returns true,

* + - * 1. Else it returns false.
      1. userField locates the ships on the field randomly (shipsLocation). It accepts nothing.
         1. To do that at first shipsLocation calls the method direction, to decide, whether the ship direction is vertical or horizontal, and saves the result in variable shipDirection.
         2. Then it picks the property ships and picks each of it's elements(ship):

if length of ship = shipLength.small ,

it generates randomly two different numbers in diapason 0- fieldSize,

transform them to string and concatenate them,

Then it calls the method collision with argument(above result) and saves the result in new variable check.

while check = true, it does the 3 instructions above again,

Then sets the result as the ship.location.

else if length of ship = shipLength.medium and shipDirection = vertical,

it creates an empty Array and saves it in a new variable checkLocation.

it generates randomly two different numbers, first in diapason 0- fieldSize-1 and the second in diapason 0- fieldSize, and adds as the ship's location's start position the concatenation of transformation this numbers to string to checkLocation.

For the rest positions it first add to the first number 1 and then does the same.

Then it picks each element of checkLocation and for each of them calls the method collision and save the result in new variable check.

while check = true, it does the 4 instructions above again,

Then it sets the result as the ship.location

Else if length of ship = shipLength.medium and shipDirection = horizontal,

It creates an empty Array and save it in new variable checkLocation.

it generates randomly two different numbers, first in diapason 0- fieldSize and the second in diapason 0- fieldSize-1, and adds as the ship's location's start position the concatenation of transformation this numbers to string to checkLocation.

For the rest positions it first adds to the second number 1 and then does the same.

Then it picks each element of checkLocation and for each of them it calls the method collision and saves the result in new variable check.

while check = true, it does the 4 instructions above again,

Then it sets the result as the ship.location,

Else if length of ship = shipLength.big and shipDirection = vertical,

It creates an empty Array and saves it in new variable checkLocation.

it generates randomly two different numbers, first in diapason 0- fieldSize-2 and the second in diapason 0- fieldSize, and adds as the ship's location's start position the concatenation of transformation this numbers to string to checkLocation.

For the rest positions it first adds to the first number 1 and then does the same.

Then it picks each element of checkLocation and for each of them it calls the method collision and saves the result in new variable check.

while check = true, it does the 4 instructions above again,

Then it sets the result as the ship.location,

Else if length of ship = shipLength.big and shipDirection = horizontal,

It creates an empty Array and saves it in new variable checkLocation.

it generates randomly two different numbers, first in diapason 0- fieldSize and the second in diapason 0- fieldSize-2, and adds as the ship's location's start position the concatenation of transformation this numbers to string to It creates an empty Array and save it in new variable checkLocation.

For the rest positions it first add to the second number 1 and then do the same.

Then it picks each element of checkLocation and for each of them it calls the method collision and saves the result in new variable check.

while check = true, it does the 4 instructions above again,

Then it sets the result as the ship.location,

* + - * 1. Method returns nothing (we can return false).
    1. We create the following properties in the object computerField :

1. fieldSize: 7 (it means 7 rows and 7 columns),

2. shipsCount: 6 ( count of ships ),

3. ships: [{location:[""], hits:[""]},

{location:[""], hits:[""]},

{location:[""], hits:[""]},

{location:["",""], hits:["",""]},

{location:["",""], hits:["",""]},

{location:["","",""], hits:["","",""]},

],(property ships contains all ships of object userField. In fact it contains an Array, that in turn contains objects(ships), that, in turn, contains 2 properties: location, that contains an Array with locations of a ship and hits, that contains an Array with empty strings, that changes to "hit" when corresponding location is hitted.

We have overall 6 ships for each player, from which 3 are small(1 square), 2 are middle (2 squares) and 1 is big (3 squares).)

4. sunkShips: 0; (the count of sunk ships),

5. shipLength: { small:1,

medium:2,

big:3,} ( length of different kind of ships),

* + 1. We create the following methods in object computerField :
       1. computerField can receive the location of shooting and check, whether it is hit or not.(method fireCheck).

If it is hit, method updates ships[corresponding ship].hits and sets the corresponding element, that have the same index as shooted element, to “hit”. And then it checks if the ship sunk by calling the method isSunk, if isSunk() is true, sunkShips+=1.

And finnaly it returns true, otherwise it returns false.

* + - 1. Method isSunk - it checks a ship, if all the elements are “hit”, it returns true, otherwise it returns false.
      2. direction – the method accepts nothing ․
         1. We randomly pick a number in diapason 0 - 2 and save that number in variable dir,
         2. If dir = 0, method returns “horizontal”,
         3. Else if dir = 1 method returns “vertical”,
      3. collision – The method accepts parameter location, and checks, if there is a match with the ships’ locations or not.
         1. It picks each element of ships,

Then it picks each element of ship.location ,

If element of ship.locations = location,

Method returns true,

* + - * 1. Else it returns false.
      1. computerField locates the ships on the field randomly (shipsLocation). It accepts nothing.
         1. To do that at first shipsLocation calls the method direction, to decide, whether the ship direction is vertical or horizontal, and saves the result in variable shipDirection.
         2. Then it picks the property ships and picks each of it's elements(ship):

if length of ship = shipLength.small ,

it generates randomly two different numbers in diapason 0- fieldSize,

transform them to string and concatenate them,

Then it calls method collision and saves the result in variable check.

while check = true, it does the 3 instructions above again,

Then sets the result as the ship.location.

else if length of ship = shipLength.medium and shipDirection = vertical,

it creates an empty Array and saves it in new variable checkLocation.

it generates randomly two different numbers, first in diapason 0- fieldSize-1 and the second in diapason 0- fieldSize, and adds as the ship's location's start position the concatenation of transformation this numbers to string to checkLocation.

For the rest positions it first add to the first number 1 and then does the same.

Then it picks each element of checkLocation and for each of them calls the method collision and save the result in new variable check.

while check = true, it does the 4 instructions above again,

Then it sets the result as the ship.location

Else if length of ship = shipLength.medium and shipDirection = horizontal,

It creates an empty Array and save it in new variable checkLocation.

it generates randomly two different numbers, first in diapason 0- fieldSize and the second in diapason 0- fieldSize-1, and adds as the ship's location's start position the concatenation of transformation this numbers to string to checkLocation.

For the rest positions it first adds to the second number 1 and then does the same.

Then it picks each element of checkLocation and for each of them it calls the method collision and saves the result in new variable check.

while check = true, it does the 4 instructions above again,

Then it sets the result as the ship.location,

Else if length of ship = shipLength.big and shipDirection = vertical,

It creates an empty Array and saves it in new variable checkLocation.

it generates randomly two different numbers, first in diapason 0- fieldSize-2 and the second in diapason 0- fieldSize, and adds as the ship's location's start position the concatenation of transformation this numbers to string to checkLocation.

For the rest positions it first adds to the first number 1 and then does the same.

Then it picks each element of checkLocation and for each of them it calls the method collision and saves the result in new variable check.

while check = true, it does the 4 instructions above again,

Then it sets the result as the ship.location,

Else if length of ship = shipLength.big and shipDirection = horizontal,

It creates an empty Array and saves it in new variable checkLocation.

it generates randomly two different numbers, first in diapason 0- fieldSize and the second in diapason 0- fieldSize-2, and adds as the ship's location's start position the concatenation of transformation this numbers to string to It creates an empty Array and save it in new variable checkLocation.

For the rest positions it first add to the second number 1 and then do the same.

Then it picks each element of checkLocation and for each of them it calls the method collision and saves the result in new variable check.

while check = true, it does the 4 instructions above again,

Then it sets the result as the ship.location,

* + - * 1. Method returns nothing (we can return false).
  1. Object model contains the following methods.
     1. Method userFire accepts as a parameter a location.
        1. It calls the method fireCheck of object computerField,
        2. It retutns nothing(or false).
     2. Method computerFire accepts as a parameter a location.
        1. It calls the method fireCheck of object userField,
        2. It retutns nothing(or false).
     3. Method startGame accepts as a parameter nothing.
        1. It calls the method shipsLocation of the object computerField,
        2. It calls the method shipsLocation of the object userField,
        3. It retutns nothing(or false).

1. We create a function parseGuess that gets as a parameter a text and checks and edits it and returns a format of location that must be sent to object model or returns null.
   1. It accepts as a parameter guess,
   2. It creats an Array, with Alphabet elements from A – G in a new variable alphabet,
   3. If guess === null, or guess.length!==2, alert("Oops, please enter a letter and a number on the board"),
   4. Else it gets the first symbol of the parameter guess and saves it in a new variable firstChar, gets from the alphabet index of firstChar and saves it in a new variable row, gets from the guess the second element, and saves it in a new variable column,
   5. If row or column is not a number, alert("Oops, that isn`t on the board");
   6. Else if row < 0 or row >= the boardsize of the computerField, or column < 0 or column >= the boardsize of the computerField, alert("Oops,that`s off the board");
   7. Else return row + column,
   8. If we don’t have the field 3.7, then returns null,
2. We create a function in randomLocation, that generates a random location for computer.
   1. It gets no parameter;
   2. It creates two different random numbers in diapason 0 - model.userField.fieldSize and saves them in new variables row and col.
   3. It returns “u”+row+col
3. We create an object (variable- controller) that is responsible for the connection between model and veiw.
   1. It has the following properties:
      1. Property guesses that holds how many times has the player shooed.(entered a correct form of location and pressed on mouse or pressed enter).(start number is 0);
      2. Property computerGuessedLocs, that holds all the guesses that has computer made,(firstly an empty Array)
      3. Property userGuessedLocs, that holds all the guesses that has user made,(firstly an empty Array)
   2. It has the following methods:
      1. Method computerGuess that is described from line 5.2.5.
      2. Method processGuess – it gets from parseGuess it’s return and checks if it is true, adds the value of property guesses by 1, and then sends the location to the object model.
         1. It accepts a parameter guess
         2. It calls the function parseGuess with argument guess, and saves the result in a new variable location.
         3. If location is true
            1. Property guesses ++,
            2. It calls the method fireCheck of the computerField with an argument location and saves the result in a new variable hit.
            3. If hit is true it sends a message “hit” to the object view and updates the computerField screen.
            4. If location is true and sunkShips of computerField === numShips of computerField

It sends a message “You sank all my battleships, in " + this.guesses + "guesses", and ends the game( returns false).

* + - * 1. Else if hit is false, it sends a message “you missed” to the object view and updates the computerField screen.
      1. After three seconds it calls the method computerGuess of controller.

1. We create a function handleFirebutton – it gets from HTML document the text that has user written, makes it uppercase, then calls the method processGuess of controller and then sets the value of HTML element to an empty string (“”).
2. We create a function init(without parameters) that starts the Game when HTML document is loaded.
   1. Function init randomly locates ships on two fields ( userField and computerField) by calling the method startGame of the object model.
   2. It waits when Player enters his guess(location).
      1. After Player have done this, and clicked the button, or pessed enter, it sends the guess to the controller.
      2. The controller in it’s turn checks the guess, if the form is true it sends the transformed guess to computerField and adds to the property of controller.guesses 1,
         1. Object model returns whether or not was the hit correct,
         2. And controller sends the answer to the object view, and the object view updates itself and returns that all was successful.
      3. Else while the form is false, it warns the Player and asks to input once more and does instructions from 5.2.2 once more.
      4. After this controller checks whether or not were all the ships of computerField sunk;
         1. If it is true, it sends a message to object view that Player wins in controller.guesses steps and stops the function init.
         2. Else continius playing
      5. Three seconds after this was done the controller randomly creats a location(guess),(We must add a new global function ).
      6. If the location was not created before, it sends the guess to the userField;
      7. Object model returns whether or not was the hit correct,
      8. And controller sends the answer to the object view, and the object view updates itself and returns that all was successful.
      9. After this the controller checks whether or not were all the ships of the userField sunk;
         1. If it is true, it sends a message to the object view that Computer wins in controller.guesses steps and ends the game( returns false).
         2. Else continius playing

We can improve the Game by adding a logic which is:

When computer guesses and it is hit and the ship is not sunk, then next time computer guesses not randomly, but it hits the neighbor locations and continues until the ship is sunk.