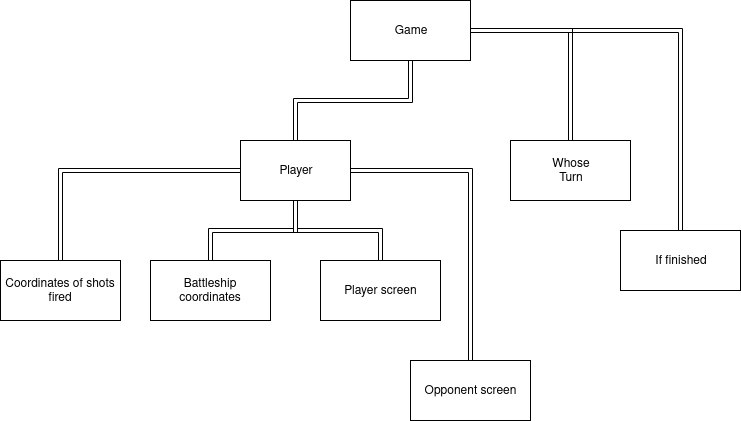
# Design



This diagram shows a high-level design of the game. The game will be represented using a Game object, which keeps track of the player data. It also tracks whose turn it is to determine which player’s screens to draw. At the end of each turn, the Game object checks if the game is finished i.e., if all the battleships of a player have been sunk.

There are two Player objects, each representing a player. The Player object keeps track of the coordinates of where the player fired shots, and the coordinates of the player’s battleships. The battleship coordinates are used with the coordinates of the opponent’s shots fired in order to draw the player screen. The coordinates of the shots fired combined with the coordinates of the opponent’s battleships are used to draw the opponent screen (obviously the opponent’s battleships aren’t drawn, but shots that hit the battleships are drawn in red).

The current player is then allowed to click on the screen, and these coordinates are added to the co-ordinates of this player’s shots fired. The Game then switches to the next player’s turn.

## Game

play = True  
while play   
  
 player1 = new Player()  
 player2 = new Player()  
  
 gameOver = False  
 player1Turn = True  
  
 player1.placeShips()  
 player2.placeShips()  
  
 while not gameOver  
  
 var currentPlayer  
 var otherPlayer  
 if player1Turn then  
 currentPlayer = player1  
 otherPlayer = player2  
 else  
 currentPlayer = player2  
 otherPlayer = player1  
 endif  
 player1Turn = not player1Turn  
  
 currentPlayer.displayPlayerScreen(otherPlayer.getFiredCoordinates())  
  
 gameOver = True  
 for coords in currentPlayer.getShipCoordinates()  
 if not otherPlayer.getFiredCoordinates().contains(coords) then  
 gameOver = False  
 endif  
 endfor  
  
 if not gameOver then  
 currentPlayer.displayOpponentScreen(otherPlayer.getShipCoordinates())  
 endif  
 endwhile  
  
 dialog = new TwoButtonDialog("Game over!", "Exit", "New game")  
 if dialog.firstButtonClicked() then  
 play = False  
 endif  
  
endwhile

At the start, the algorithm draws the first player’s screen, allowing the player to choose where to place ships. This is then done for the second player. After that the algorithm enters a loop with the player turns.

Within the loop, the current player is first determined. Then the current player’s player screen is shown. This screen shows the player’s ships and the shots that the opponent fired into these ships. This means that the shots that the opponent fired are passed as an argument into the procedure used to display this screen.

After that, the algorithm checks if the game is finished. This is done by checking if the fired coordinates of the opponent contain all of the ship coordinates of the current player. First, the algorithm sets gameOver to True. Then it iterates through all of the ship coordinates of the current player and sets gameOver to False if the other player’s fired coordinates don’t contain these coordinates.

If the game isn’t finished that the opponent screen is displayed, allowing the player to pick a square in which to fire a short. Once the player has done that, the fired coordinates list of this player is modified and the next player’s turn begins. If the game is finished, then a dialog appears, allowing the player to exit or to start a new game. If the player chooses to exit, then play is set to False, so the top-level loop is exited and the program closes. Otherwise, the top-level loop is iterated through again, starting a new game.

## Player

class Player  
   
 private shipCoordinates  
 private firedCoordinates  
 private squares  
  
 public procedure new()  
 shipCoordinates = []  
 firedCoordinates = []  
 for x = 0 to 9  
 for y = 0 to 9  
 square = new Button()  
 getScreen().add(square)  
 square.setCoordinates(x, y)  
 square.setColor("blue")  
 squares.append(button)  
 endfor  
 endfor  
 endprocedure  
  
 public function getShipCoordinates()  
 return shipCoordinates  
 endfunction  
  
 public function getFiredCoordinates()  
 return firedCoordinates  
 endfunction  
  
 private function findSquare(x, y)  
 for square in squares  
 if square.getXCoordinates() == x and square.getYCOordinates == y then  
 return square  
 endif  
 endfor  
 endfunction  
  
 public procedure placeShips()  
 ...  
 endprocedure  
  
 public procedure displayPlayerScreen(opponentFiredCoordinates)  
 ...   
 endprocedure  
  
 public procedure displayOpponentScreen(opponentShipCoordinates)  
 ...  
 endprocedure  
endclass

We use a Player class in the game. This class contains a list of the coordinates of the player’s ship and a list of the fired coordinates, which are set to empty lists in the Player constructor. It has a getter for shipCoordinates and firedCoordinates, but no setter, as only the Player class should be able to modify them. Additionally, Player has the three procedures which we saw above and will discuss below. It also has a private function findSquare, which is used to find a square by its coordinates. This function is used within the the three procedures. It isn’t needed outside of them so it is set to private.

### Ship placement

private function findShipCoordinates(x, y, size, vertical)  
 result = []  
 for i = 0 to size - 1  
 if vertical then  
 result.append((x, y + i))  
 else  
 result.append((x + i, y))  
 endif  
 endfor  
 return result  
endfunction  
  
public procedure placeShips()  
 vertical = True  
 const SHIP\_SIZES = [5, 4, 3, 3, 2]  
 for size in SHIP\_SIZES  
 shipPlaced = False  
 while not shipPlaced  
 action = getScreen().getMouseAction()  
 if action.isRightClick() then  
 vertical = not vertical   
 else if action.isLeftClick() then  
 shipPlaced = True  
 for coordinates in findShipCoordinates(action.getMouseX(), action.getMouseY(), size, vertical)  
 shipCoordinates.append(coordinates)  
 endfor  
 else if action.isMouseMoved() then  
 for square in squares  
 if square.getCoordinates() in findShipCoordinates(action.getMouseX(), action.getMouseY(), size, vertical)  
 square.setColor("grey")   
 else if not shipCoordinates.contains(square.getCoordinates())  
 square.setColor("blue")  
 endif  
 endfor  
 endif  
 endwhile  
 endfor   
endprocedure

We have the helper function findShipCoordinates, which finds the squares of the ship corresponding to the current mouse location: it selects multiple squares based on the single mouse location. If the orientation is vertical then all of the squares have the mouse’s x coordinates but the y coordinates vary from 0 to size - 1, so that the total size is size. Note that higher y coordinates correspond to lower positions on the screen so the top of the mouse corresponds to the top of the ship. The opposite is true from the horizontal orientation: the y coordinates are the same as the mouse’s, while the x coordinates vary from 0 to size - 1. Higher x coordinates correspond to positions further to the right, so the mouse is on the left of the ship. The findShipSquares function is only used within placeShips so findShipSquares is set to private.

In placeShips, we iterate through the list of ship sizes, which is a constant so that it can be changed easily. we start with an initial orientation of vertical. For each size, we wait until the ship is placed by waiting for a left click. Meanwhile, we observe other mouse actions. On a right click, we change the ship orientation, by flipping the value of vertical. If the mouse is moved, then we find the ship selection corresponding to the mouse, and color these squares grey. For other squares, we color them blue if they don’t correspond to any ship coordinates, in case they’ve been set to grey by a previous mouse selection (without the ship actually being placed). Finally, on a left click, we add all of the squares corresponding to the mouse to shipCoordinates, and set shipPlaced to True so that the while loop is exited and the next ship size is iterated over in the for loop. Note that because the mouse must have entered a square corresponding to this ship in order to click it, all of the ship’s squares have been colored grey, and they will not be colored blue when the mouse is moved again, as these squares are now in shipCoordinates.

### Player screens

public procedure displayPlayerScreen(opponentFiredCoordinates)  
 opponentHit = []  
 for coordinates in shipCoordinates  
 square = findSquare(coordinates)  
 if opponentFiredCoordinates.contains(square) then  
 square.setColor("red")   
 opponentHit.append(coordinates)  
 else  
 square.setColor("grey")  
 endif  
 endfor  
 for coordinates in opponentFiredCoordinates  
 if not opponentHit.contains(coordinates) then  
 findSquare(coordinates).setColor("dark blue")  
 endif  
 endfor  
endprocedure

This procedure display the player screen, showing the player’s ships and how the opponent has tried to fire at them. First it iterates through all of the ship coordinates. If these coordinates are in the opponentFiredCoordinates, then the square has been hit so it is set to red. Otherwise the square hasn’t been hit so it is set to grey, the non-hit ship color. Additionally, a list is kept of the coordinates that the opponent has hit. After the first for loop, we iterate through the opponentFiredCoordinates. We iterated through all of the shipCoordinates, so opponentHit contains all of the hit coordinates. This means that if the coordinates are not in opponentHit, then they must have been a miss so they are set to the miss color: dark blue.

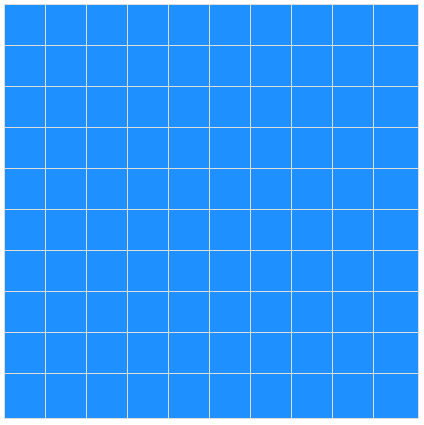
### Opponent screen

private procedure setHitOrMissColor((x, y), opponentShipCoordinates)  
 square = findSquare(x, y)  
 if opponentShipCoordinates.contains((x, y)) then  
 square.setColor("red")  
 else  
 square.setColor("dark blue")  
 endif  
endprocedure  
  
public procedure displayOpponentScreen(opponentShipCoordinates)  
 for coordinates in firedCoordinates  
 setHitOrMissColor(coordinates, opponentShipCoordinates)  
 endfor  
 fired = False  
 while not Fired  
 action = getScreen().getMouseAction()  
 if action.isLeftClick() then  
 setHitOrMissColor(action.getMouseCoordinates(), opponentShipCoordinates)  
 firedCoordinates.append(action.getMouseCoordinates())  
 fired = True  
 else if action.isMouseMoved()  
 for square in squares  
 if square.getColor() == "white" then  
 square.setColorToPrevious()  
 endif  
 endfor  
 square = findSquare(action.getMouseX(), action.getMouseY())  
 square.setColor("white")  
 endif  
 endwhile  
endprocedure

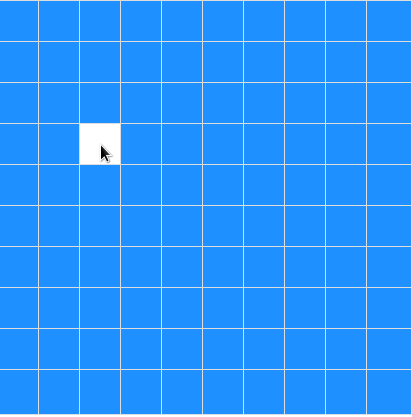
The helper procedure setHitOrMissColor is used to set the color of a button based on its coordinates and the opponentShipCoordinates list. It first uses the findSquare function to find the corresponding square. It then checks if the coordinates are in the opponentShipCoordinates list. If they are then it must have been a hit so the square is set to the hit color: red. Otherwise it must have been a miss so it set to the miss color: dark blue. setHitOrMissColor is only used within the displayOpponentScreen procedure, so setHitOrMissColor is set to private.

In the displayOpponentScreen procedure, we first iterate through all of the fired coordinates and use setHitOrMissColor set the color to the corresponding button depending on whether it was a hit or miss. We then enter a while loop where we wait for the player to fire. If the player moves the mouse, then we set color of the square at the location of the mouse to white. Additionally, we loop through all of the white squares: these are the squares that have previously been selected. As they are no longer selected, we set them to the previous color. We also loop through these squares before setting the color of the newly selected square: otherwise the the newly selected woud be immediately set to the previous color, which isn’t desirable. Once the player has made a left click, we consider the player to have fired. Then we set the color of the button, to tell the player if they hit or missed. Finally, we add the selected coordinates to the firedCoordinates list and then set fired to True so that the while loop is exited and we move to the next part of the program.

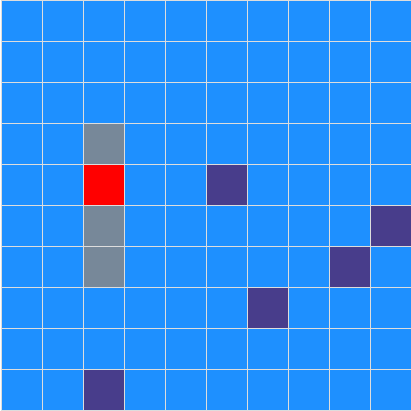
## Visual design



The UI will be a 10 by 10 grid. When empty, it will have a nice light blue color, which imitates the color of water. For the sake of simplicity and to reduce distraction, there will be no menu surrounding the grid, and all communication will be done using dialogues.



The selected square is colored white to provide the user with visual feedback that they have selected a square and their selection has been taken into account.



The ships are colored with a metalic grey, which is similary to the color of an actual ship. The shots that hit a ship are colored red, because this makes it look like fire, a result of the ship being hit. The missed shots are colored in dark blue, which makes the water look deeper, as if the shot has penetrated the surface level of the water.

The above image is what is used on the player screen. On the opponent screen, everything will be the same except that there will be no ships. On the initial placement screen, there will be ships (of the same color) but no shots.