Programmer documentation

Grid system

Each player gets assigned two grids, a secret grid and a public grid. The secret grid holds information about placements of their own ships, while the public grid is used when the players are targeting eachother's ships. These grids are objects which take a parameter of type Windows.Forms.GroupBox. The GroupBox objects are simply lists and are treated as such throughout the program. The GameWindow Form has 4 GroupBoxes with 100 buttons each. These GroupBoxes are assigned to four different Grid objects which are then assigned to two Player objects, as shown:

Top of GameWindow.cs

```
16          private Grid player1SecretGrid;
17          private Grid player2SecretGrid;
18          private Grid player1PublicGrid;
19          private Grid player2PublicGrid;
20          private Player player1;
21          private Player player2;
```

Start of newGameBtn_Click event handler in GameWindow.cs

```
94              // Create Grid object with GroupBox control
95              player1SecretGrid = new Grid(player1SecretGridGroupBox);
96              player2SecretGrid = new Grid(player2SecretGridGroupBox);
97              player1PublicGrid = new Grid(player1PublicGridGroupBox);
98              player2PublicGrid = new Grid(player2PublicGridGroupBox);
```

In newGameBtn_Click event handler in GameWindow.cs

```
103             // Instantiate Players
104             player1 = new Player();
105             player2 = new Player();
106
107             player1.secretGrid = player1SecretGrid;
108             player2.secretGrid = player2SecretGrid;
109             player1.publicGrid = player1PublicGrid;
110             player2.publicGrid = player2PublicGrid;
```

Inside Grid.cs in the constructor the groupBox variable is assigned to the Grid objects groupBox variable:

```
21          public GroupBox playerGroupBox;
22          public int occupiedTilesCount;
            4 references | Morten Oleander Nielsen, 17 hours ago | 1 author, 2 changes
23          public Grid(GroupBox groupBox)
24          {
25              this.playerGroupBox = groupBox;
26              Populate();
27          }
```

The function Populate is called which populates the tiles list within the Grid object with Tile objects that take in the Grid's GroupBox's Button objects:

```
288         // Populate the tiles array to make up the grid
            1 reference | Morten Oleander Nielsen, 17 hours ago | 2 authors, 3 changes
289         private void Populate()
290         {
291             // Populate tiles with buttons from the GroupBox associated with this grid object
292             foreach (Control control in this.playerGroupBox.Controls)
293             {
294                 tiles.Add(new Tile((Button)control));
295             }
296
297             // Reverse the order of tiles list to get around weird indexing in GroupBox
298             tiles.Reverse();
299         }
```

The tiles list is reversed to get around weird ordering done in the GroupBox object.

The Tile class contains variables to check if the Tile is occupied and to get the button associated with the tile. It also includes functions Hit, Miss, Highlight, Place, Disable and DetermineHitOrMiss, which are used to style the buttons associated with the Tile objects and setting the isOccupied variable.

Main game flow

Every action the player can take is determined by button event handlers in the GameWindow.cs class.

When the program starts the GroupBoxes parents are set to the GameWindow object, because they are grouped on top of eachother in the designer.

The User pressed the 'New Game' button which has an event handler associated with the buttons onclick event. At the start of the event handler the reset function is called which simply calls the Window_Load function. Afterwards the 4 Grids are instantiated. Next up the gameState is set to creation, which is used further down to decide what should happen when the user presses the end turn button. The two players are then instantiated. The grids of the two player objects are set to the grids created. Next up a variable to determine the current player is set to the player1 object. The PlaceShips function is then called with a parameter that is the current player variable.

The PlaceShips function takes a parameter of type Player. Essentially the PlaceShips function checks to see if the Player object has any ships left and places them on their secret grid if they do.

The next step is to enable all the Controls and attach event handlers to the button on click event in the players' public GroupBoxes. This is done like so:

```
116             // Attach event handler to each button (tile) in the GroupBox (grid)
117             foreach (Control control in this.player1PublicGridGroupBox.Controls)
118             {
119                 if (control is Button)
120                 {
121                     // Make every button clickable
122                     control.Enabled = true;
123
124                     // Attach event handler
125                     control.Click += new EventHandler(tileBtn_Click);
126                 }
127             }
128
129             // Attach event handler to each button (tile) in the GroupBox (grid)
130             foreach (Control control in this.player2PublicGridGroupBox.Controls)
131             {
132                 if (control is Button)
133                 {
134                     // Make every button clickable
135                     control.Enabled = true;
136
137                     // Attach event handler
138                     control.Click += new EventHandler(tileBtn_Click);
139                 }
140             }
```

When the players hit the End Turn button the endTurnBtn_Click event handler is called. The Event handler contains a switch on the gameState variable, because it does different things depending on if the game is currently in Creation, i.e. ship placement on the Players' secret grids, or if the game is currently InPlay, i.e. the targeting phase.

The tileBtn_Click Event handler gets the button that has been clicked on and finds the tile that corresponds to that button within the current player's public grid.

A switch on the gameState that isn't really needed is used and if the gameState is in play it determines through if statements which player's turn it is. When this is determined another if statement checks if the other player's grid has any occupied tiles left. If they do the function DetermineHitOrMiss, which compares the current player's public grid to the other player's secret grid and determines whether or not the tile that has been clicked on is occupied and therefore either hit or missed, is called. The turn is then ended automatically by calling Click on the end turn button.

PlaceShip function

This is the meat of what I've spent my time on in the project. I wanted to make a function that would work for placement of all different types of ships. The PlaceShip function is placed in Grid.cs and takes a parameter of type int called shipLength. First of all the function finds a list of tiles in the current Grid object that are not occupied. Then it chooses a random tile from the non-occupied tiles list to start the placement of the ship. It then calls the place function on that starting tile. Then it checks if the starting tile can be

found within the whole tiles list of Grid object. If this is true it finds the index of the starting tile within the Grid object's tiles list. It then makes four lists of possible placements for all directions starting from the starting tile. The next step is to run a for loop to find out how many tiles should be filled based on the shipLength parameter. The function then checks for all four lists the starting tile is not on the edge of the grid. For example before the rightTiles list is populated it checks to see if the starting tile's placement is not on the right edge of the board, because if it is then placement to the right of the starting tile is not legal and the rightTiles list shouldn't be populated.

If the starting tile is not on the right edge of the board the rightTiles list is populated with the four tiles to the right of the starting tile. This is done by adding the increment from the for loop to the index of the starting tile and finding those tiles within the tiles list. The reason for adding the increment is because every time the index of the tiles list increments by one we get the tile to the right in the grid. If the tiles to the left are to be found the increment is instead subtracted from the starting tile index.

The next step is to ensure any of the tiles in the rightTiles list do not go over the edge of the right side of the grid because if they do they are not valid placements and the rightTiles list should be cleared.

The next step is to find the tiles adjacent to the rightTiles and set them to occupied since they are not valid placements for the next ship. To do this four if statements are made to determine whether any of the tiles in the rightTiles list are on any of the four edges of the board. For example if any of the tiles are on the right side of the board the tiles adjacent to the right are not actually on the right but instead on the left edge of the board because of how the lists are indexed. If none of the tiles are on the right edge of the board the tiles to the right are set to occupied to indicate that they are not valid placements for the next ship.

A bit further down, out of the for loop, four if statements are made that check each of the four tile direction lists to ensure that they aren't empty. The corresponding direction tile list is then looped through and each tile has the Place function called on it. The variable occupiedTilesCount, which is used to determine when a player has no ships left on their secret grid and therefore lose, is incremented for each tile.