The game starts with the option to choose between the two different rules: Salvo or Basic. The buttons are just grid locations, so if there is a click, the location is recorded and checked to see if it is either of the two locations. Depending on which is clicked a Boolean type variable becomes true. Then the ships must be placed. First an intermediate screen is placed saying “Player 1 click to continue” this intermediate screen is presented every time the screen switches players. After clicking, a grid is presented, with a black bar on top which tells you what ship you are placing. A rotate button is in the top right that allows you to change the orientation of the ship. If the mouse is not clicked, then a ship will follow your mouse, and will be placed when clicked on the grid. A check is made to make sure the ships do not overlap or go out of bounds with a method called testOverlapAndBounds.

After all five ships have been placed, the player value is switched, the ship count is reset, and the loop counter is increased. The intermediate screen is shown again as well. After the other player places all of their ships, the loop quits. Also as the ships were being placed, their values were being stored into locations within the Player class. As the ship became permanently placed on the grid, a method called permaDraw was called within the drawing method. The drawing method for this part was called setUp and drew the grid, rotate button, the String for what ship it was, and called permaDraw. The permaDraw class searched through the Player’s grid and drew ships, according to what ship type was invoked. This method was put through a loop so that all ship types were drawn on the grid.

After the placement part, the StdDraw panel is redrawn with new dimensions. A new drawing method is called as well. This one is called drawGrid and it draws the grid, the ships, misses, hits, and writes how many shots are remaining. Depending on which case was chosen a loop is started, if it is the basic case, there is either a hit or a miss. If there is a hit, there is a check to see if a ship was sunk. Either way a message is given out saying what was hit/sunk. If it is a miss, the value is stored and the player is switched. Each time a ship is sunk, it checks to...
see how many ships are remaining. If they are down to zero, the game is over and that person that sank the ship wins.

For Salvo rules, when shots are placed, a Salvo object is created and placed into an array of salvos. After all the shots have been taken, the array is run through to check to see if they were hits or not. If there is a hit, the sinkShip check is made. Again if the ship count is down to zero, the game is over and the player that sank the last ship wins.

Difficulties:

Creating the physical ships was difficult. I decided to make a rectangle that was slightly smaller than the grid and then add circles at the ends. This presented difficulty when there was a hit on the ship, as the value would change, so I allowed hits to be drawn as ships if adjacent to ship values. This poses as small problem because if there are two ships next to each other and there is a hit on the connecting spot, the ships will be drawn as overlapping.

Another big problem was flicker and lag, StdDraw.show() is used everywhere in my code to reduce on screen draw time.

Code-wise the biggest difficulty was figuring out how to limit the salvo array after a ship has been sunk. I had to figure out a way to reduce the checks as per each player, depending upon their shot value. I eventually figured out how to relate ships to the index of the array.

Notes:

Below are clipped selections of code. The first is the basic game loop, and the second is the salvo game loop. I also made a very short Salvo class to store the location of and value of the temporary shots. Also I changed the StdDraw code to include a new JMenuItem for a Rules text. To include that turned out to be harder than I thought, because I had to learn how to use the try and catch syntax. Overall this turned out to be a fun game, both to make and to play. It mimics the old game almost exactly. If there was anything I would have liked to have added, it would have been a computer player. It would have started as the computer taking random shots, but maybe I could refine it so as to make more educated guesses later.
while(casel) { // basic rules
    if(StdDraw.mousePressed()) {
        int c = (int) (StdDraw.mouseX());
        int r = (int) (StdDraw.mouseY());

        // the shot hits
        if (r > 10 && list[1-player].getValue(c, r-11) > 2) {
            int value = list[1-player].getValue(c, r-11);
            list[1-player].setMine(c, r-11, 2);
            list[player].setYours(c, r-11, 2);

            boolean sank = sinkShip(list[1-player], value);
            if (sank) {
                list[1-player].ship--;  
                System.out.println("You sank my " + Player.getString(value));
                if(list[1-player].ships == 0) {
                    drawGrid(list[player], shots);
                    System.out.println("Player " + (player+1) + " wins!");
                    break;
                }
            } else System.out.println("You hit my " + Player.getString(value));
        }

        // the shot misses
        else if (r > 10 && list[1-player].getValue(c, r-11) == 0) {
            ...}

Salvo[] salvo = new Salvo[5];
while(casel) { // salvo rules
    if(StdDraw.mousePressed()) {
        int c = (int) (StdDraw.mouseX());
        int r = (int) (StdDraw.mouseY());

        if (r > 10 && list[1-player].getValue(c, r-11) > 2 || list[1-player].getValue(c, r-11) == 0) {
            int val = list[1-player].getValue(c, r-11);
            list[player].setYours(c, r-11, 1);
            salvo[list[player].ships-shots] = new Salvo(r, c, val);
            shots--;

            if (shots == 0) {
                for(int x = 0; x < list[player].ships; x++) {
                    if(salvo[x].val > 2) {
                        list[1-player].setMine(salvo[x].c, salvo[x].r-11, 2);
                        list[player].setYours(salvo[x].c, salvo[x].r-11, 2);

                        boolean sank = sinkShip(list[1-player], salvo[x].val);
                        if (sank) {
                            list[1-player].ships--;
                        }
                    }
                }

                else System.out.println("You hit my " + Player.getString(salvo[x].val));
            }

            if(salvo[x].val == 0) {
                list[1-player].setMine(salvo[x].c, salvo[x].r-11, 1);
                list[player].setYours(salvo[x].c, salvo[x].r-11, 1);
            }
        } else System.out.println("You missed!");
    }