Criterion C: Product Development

WinPattern()
It may not be the most complex one but it is somewhat complex and is one of the most important algorithms needed in this game, Triplets. The purpose of the algorithm 'WinPattern()' checks for the entire buttons and see if 3 buttons are in a row horizontally, vertically or diagonally then return true. In order to make the return work, I made the algorithm in Boolean. The return true will be used to end the game.

```
public boolean WinPattern()
{ // need loop for buttons for ex. button[1][1] for convienience

    String WinPly = Winner();
```

This algorithm first implements a string from Winner() to a WinPly. The Winner() is a method that checks for who's turn it is and save it as a string for each turns. Then, this string is later used in WinPattern() algorithm to output who the winner is by looking at whose turn it was.

```
public String Winner()
{
    int CBP = StartPlayer();
    String WinPly = null;
    if (CBP == 1)
    {
        WinPly = "Player 1";
    }
    else
    {
        WinPly = "Player 2";
    }
    return WinPly;

}
```

In order for WinPattern() to check who the winner is, it first has to have a loop for all the rows and columns of the buttons. Then, it makes an 'if' command with 'if a button with a specific row and column is color black'. The color black represents when button is clicked. This is because in the button action code, I made the button to be colored black if it is clicked. This 'if' command will be applied to all 4 ways of winning horizontally, vertically, diagonally from Northwest to Southeast and diagonally from Northeast to Southwest.

```
for (int row=0; row<AllButtons[0].length; row=row+1)
    for (int col=0; col<AllButtons[0].length; col=col+1)
    {

        if (AllButtons[col][row].getBackground() == Color.black)
        {
```

First, for horizontal, I made an 'if' command that 'if column, which is horizontally arranged, is smaller than AllButtons[0].length-2 which is 4, and AllButtons[row][col+1] is black and AllButtons[row][col+2] is black' then return true. This means that 3-in-a-row is completed horizontally. The column had to be smaller than 4 but since this is what we call a magic number in computer science, I used the length of all buttons which is 6 and subtracted it by 2. Anyhow, if it column was 4 or higher, the coding would have been broken since it could ask for AllButtons[5][7] which does not exist. The same use of coding was used for vertical, rows. Yet, this time, the +1 and +2 must be applied to row not col and row should be the one smaller than 4.

```
// Vertical
    if (row<AllButtons[0].length-2
        && AllButtons[col][row+1].getBackground() == Color.black
        && AllButtons[col][row+2].getBackground() == Color.black)
    {
        textField1.setText("Winner is " + WinPly + "!");
        return true;
    }

// Horizontal
    if(col<AllButtons[0].length-2
        && AllButtons[col+1][row].getBackground() == Color.black
        && AllButtons[col+2][row].getBackground() == Color.black)
    {
        textField1.setText("Winner is " + WinPly + "!");
        return true;
    }
```

Then, it needs to check for diagonal from Northwest to Southeast. Since I am adding 2 spots further than original button, AllButtons[col][row], I must claim that chosen button must be smaller than 4 in terms of rows and columns. For example, if the chosen button was AllButtons[5][5], the next buttons will be AllButtons[6][6] and AllButtons[7][7] but they do not exist. Therefore, when the 3-in-a-row is made when it is diagonal from Northwest to Southeast, it will return true.

```
//DiagonalToSouthEast
    if (col<AllButtons[0].length-2 && row<AllButtons[0].length-2
        && AllButtons[col+1][row+1].getBackground() == Color.black
        && AllButtons[col+2][row+2].getBackground() == Color.black)
    {
        textField1.setText("Winner is " + WinPly + "!");
        return true;
    }
```

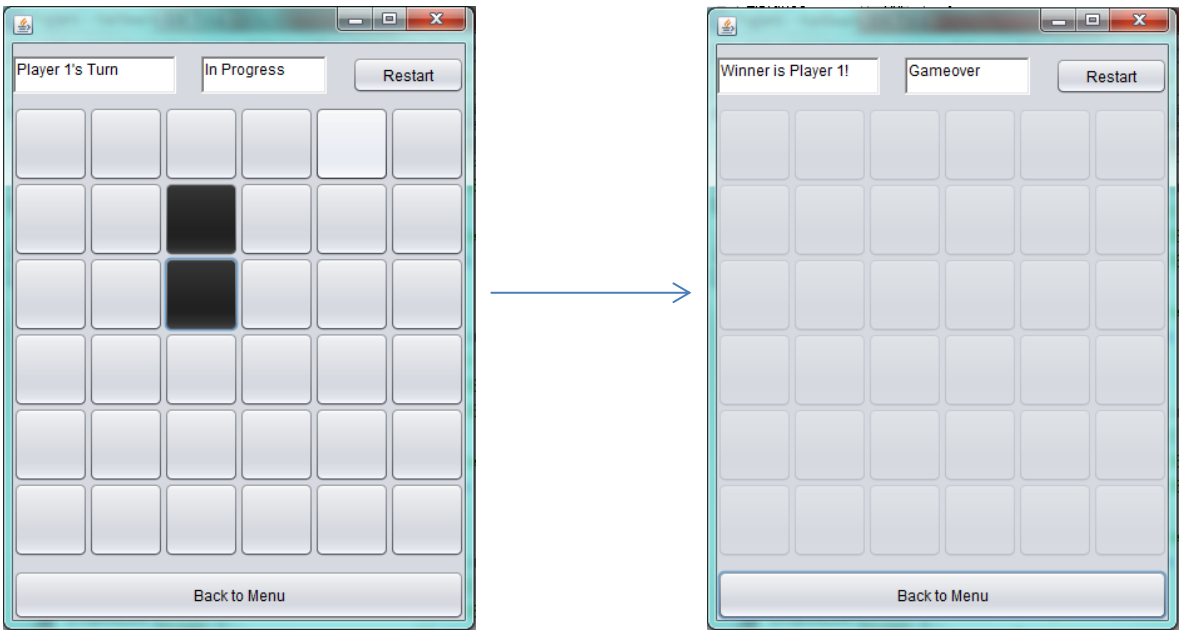
Lastly, it should check for diagonal from Northeast to Southwest. This works just like diagonal from Northwest to Southeast but this time, the column increases while row decreases. For this one, I said that column must be smaller than 4 while row is bigger than 1 since if this boundary was not made, it will look for button that does not exist and thus, resulting in 'IllegalBoundaryException'.

```
//DiagonalToSouthWest
    if (col<AllButtons[0].length-2 && row>1
        && AllButtons[col+1][row-1].getBackground() == Color.black
        && AllButtons[col+2][row-2].getBackground() == Color.black)
    {
        textField1.setText("Winner is " + WinPly + "!");
        return true;
    }
```

Then, it should return false at last since if none of these happen, it should return false that WinPattern() is not yet completed.

```
                }
            }
        }
        return false;
}
```

For example) when checking for horizontal, if I press button at row=0 and col=2 or row=3 and col=2, the game should end since triplets has been completed.



It says that the game is over meaning triplets has been made.

Another complex algorithm in my coding was Artificial Intelligence in intermediate mode but this is quite a repetition from WinPattern() so I will talk about another coding which is Artificial Intelligence in beginner mode. The purpose of AI aka artificial intelligence in beginner mode is so that players can understand how the game goes. Therefore, AI will be very bad at this game and hence, will click on any random button.

```
public void ComputerPlay()
{
    {int PlayerNum = StartPlayer();
    int colRan = (int)(Math.random() * 6);
    int rowRan = (int)(Math.random() * 6);
    int colRan2 = (int)(Math.random() * 6);
    int rowRan2 = (int)(Math.random() * 6);
    int colRan3 = (int)(Math.random() * 6);
    int rowRan3 = (int)(Math.random() * 6);
    for (int row=0; row<AllButtons[0].length; row=row+1)
        for (int col=0; col<AllButtons[0].length; col=col+1)
            if (PlayerNum % 2 != 0)
                if (AllButtons[col][row] != AllButtons[colRan][rowRan])
                {
                    AllButtons[colRan][rowRan].setBackground(Color.black);
                    return;
                }
                else if(AllButtons[col][row] == AllButtons[colRan][rowRan] && AllButtons[col][row] != AllButtons[colRan2][rowRan2])
                {
                    AllButtons[colRan2][rowRan2].setBackground(Color.black);
                    return;
                }
                else if(AllButtons[col][row] == AllButtons[colRan2][rowRan2] && AllButtons[col][row] != AllButtons[colRan3][rowRan3
                {
                    AllButtons[colRan3][rowRan3].setBackground(Color.black);
                    return;
                }
    }
}
```

In this method, I first implemented StartPlayer(), casted it into an int value and called the variable PlayerNum. This will be in use later. Then, I made three random numbers for both column and row. This will be further explained when we go deep into the methods.

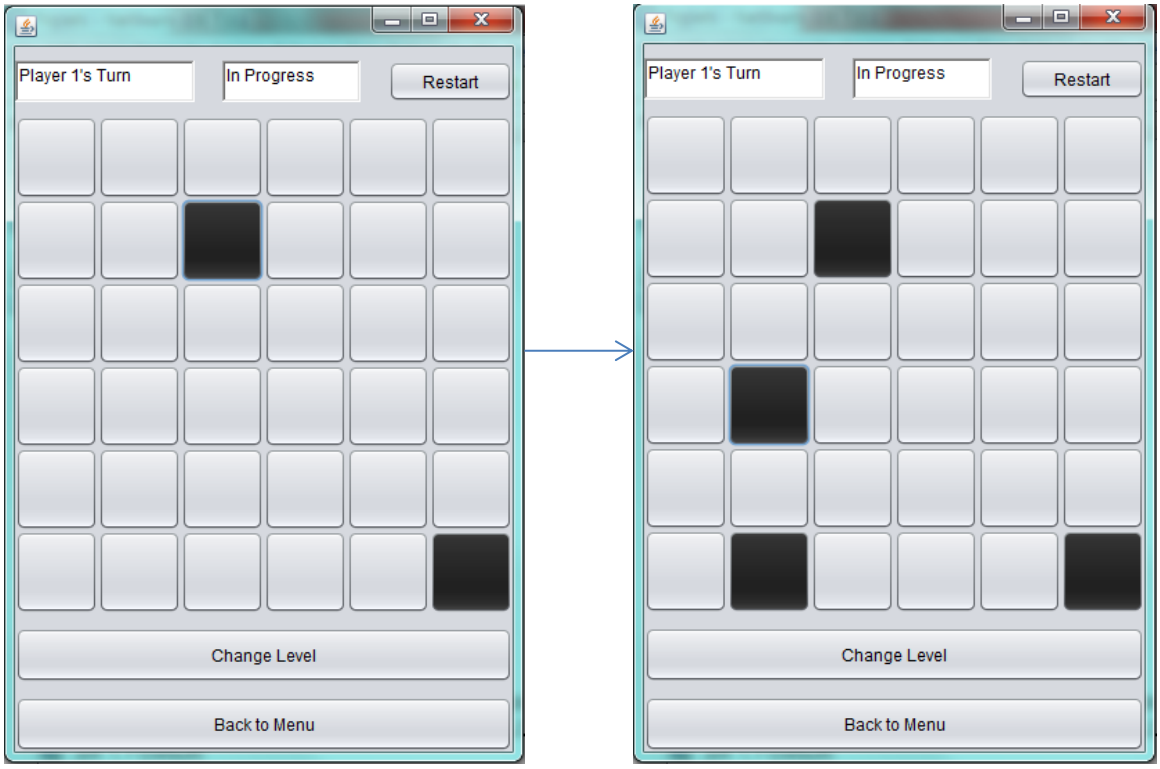
First I need a loop for row and column to check for all board. Then I used an ‘if ’ command and said if PlayerNum is divisible by 0. What StartPlayer() does is

```
public int StartPlayer()
{
    int CBP = CountButtonPerformed();

    if ((CBP % 2 ) == 0)
        {
            CBP = 2;
            textField1.setText("Player 1's Turn");
        }
        else
        {
            CBP = 1;
            textField1.setText("Computer's Turn");
        }

    return CBP;
}
```

```
public int CountButtonPerformed() // Start Player is 1.
{
    int CBP = 0; // Count Button Performed
    for (int row=0; row<AllButtons[0].length; row=row+1)
        for (int col=0; col<AllButtons[0].length; col=col+1)
        {
            if (AllButtons[col][row].getBackground() == Color.bl
            {
                CBP = CBP + 1;
            }
        }
    return CBP;
}
```

it counts number of buttons pressed and if the number of button pressed is divisible by 2, then CBP which is number of buttons pressed will be 2 and if not, it will be 1. Then, The StartPlayer() returns CBP which is either 2 or 1. Then in the ComputerPlay() method, I said if PlayerNum is divisible by 2. This is what checks for whether it is player's turn or computer's turn. Computer should not be pressing button if it is player's turn. Thus, I said if PlayerNum is not divisible by 2, which is when PlayerNum is 1 meaning it is computer's turn, press a random button on a board. However, the random button cannot be same as already pressed button so I said if button does not equal random button. Then, if it does not equal, it will press the random button and return it so that the method does not move on to else{}. For else{}, since button was equal to random button, I made another random button with colRan2 and rowRan2 so that it would be different to originally pressed button. Lastly, I have colRan3 and rowRan3 since original button equals to colRan, colRan2, rowRan and rowRan2. This will lessen the chance of same button being pressed.

For example)
As I pressed the button at row=1 and column=2, the computer pressed random button at row=5 and column=5. Then, this time I pressed button at row=3 and col=1, the computer pressed random button at row=5 and col=1.



Words: 840