1.

My thought process initially was that I needed to adjust the number of boxes/buttons that were presented and that I needed to adjust the fact that you then needed 4 in a row instead of 3 to win.

First, I adjusted the code in the GameSquare:

```
 static var reset:[GameSquare]{

       var squares=[GameSquare]()

       for index in 1...16{

          squares.append(GameSquare(id: index))

       }

       return squares

    }
```

From the index from 1-9 to have the adjusted 1-16 so that it would be adjusted for the larger game board.

However I didn't account for the fact that I would have needed to make the buttons smaller in order for them to fit on the screen. The first couple times I tried working on this, the app kept crashing when I would add the extra box and thought it was due to a larger issue. When I went back to this, I went to SquareView and adjusted the size of the height and width of the square from 100 to 65.

Lastly, I adjusted the ways in which buttons would need to be pressed in order to have a winning board. By modifying the GameModels, I was able to do this:

```
enum Moves{

    static var all = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]

    static var winningMoves = [

    [1,2,3,4],

    [5,6,7,8],

    [9,10,11,12],

    [13,14,15,16],

    [1,5,9,13],
    [2,6,10,14],
    [3,7,11,15],
    [4,8,12,16],
    [1,6,11,16],
    [4,7,10,13]
    ]
}
```

2.

To adjust the current tic tac toe code into a simple candy crush game, I would change the structure of the game, just into more buttons. First, the screen that adjusts for the number of players would no longer be needed, since you can only play candy crush games by yourself. Then, I would change the screen to have each HStack to have 4 buttons and each VStack to have 4 buttons so it makes a 4x4 table instead of the original 3x3

To change how the physical game is played, such as the enum in the GameModels grouping, it would need to be adjusted so that each button is randomly assigned a number 1, 2 or 3. These buttons would then have 3 different candy images on the buttons corresponding to their random number. I would then need to identify all the mixes of the 2 in a row, 3 in a row, or 4 in a row both vertically and horizontally, so that when the user pressed these buttons in a row, the app could then identify that the button's values need to be deleted. From there, a do-catch system could be used. The "do" would be if a button with no current value had a button vertically above it, and shift the value of the above button down, like sending button 2 down to button 6 and then potentially down to button 10 as well. The "catch" would then be any current empty button without values that would randomize a new value.

3.

To prepare the API, I wasn't able to get the apple API without a membership, so instead I tried to access one through a website called OpenWeather. Through the website I was able to get an API key including step by step instructions to include this into the Swift program. I initially started by creating a new file in Swift that I named TemperatureAPI. I followed the instructions from the website including inserting the location we were looking for, College Station. I also pulled the API key from the website along with the url. After plugging in this, I was still showing a few errors and so I compared this from a couple other Youtube videos also attempting to pull the data from the JSON file. I also tried to plug some of the code that still had errors into chat GPT and was still showing errors. In the end, I think there were errors with the JSON file and grabbing that information.

```
let city: String = "College Station"

let url = URL(string:


https://api.openweathermap.org/data/2.5/weather?q={CollegeStation}&appid={f37950cb
8dfc00451d746c942329619d
```

This was how I tried to pull the data from the API using the location and the APIKey.

```
func pullJSONData(url: URL?, forecast: Bool){

   let task = URLSession.shared.dataTask(with: url!) { data, response, error in if let error
= error {

      print("Error : \(error.localizedDescription)")

   }

      guard let response = response as? HTTPURLResponse, response.statusCode ==
200

         else{

         print("Error: HTTP Response Code Error")

         return

      }

      if (!forecast){

         decodeJSONData(JSONData: data ?? <#default value#>!)
```

```
        } else {
        }
    }
    task.resume()
}
```

This is showing what to do with the information once it is pulled and brought into the app. Whether there is information that is usable or what to do if an error occurs.