How to create a Chess App using Unity

Workshop 2

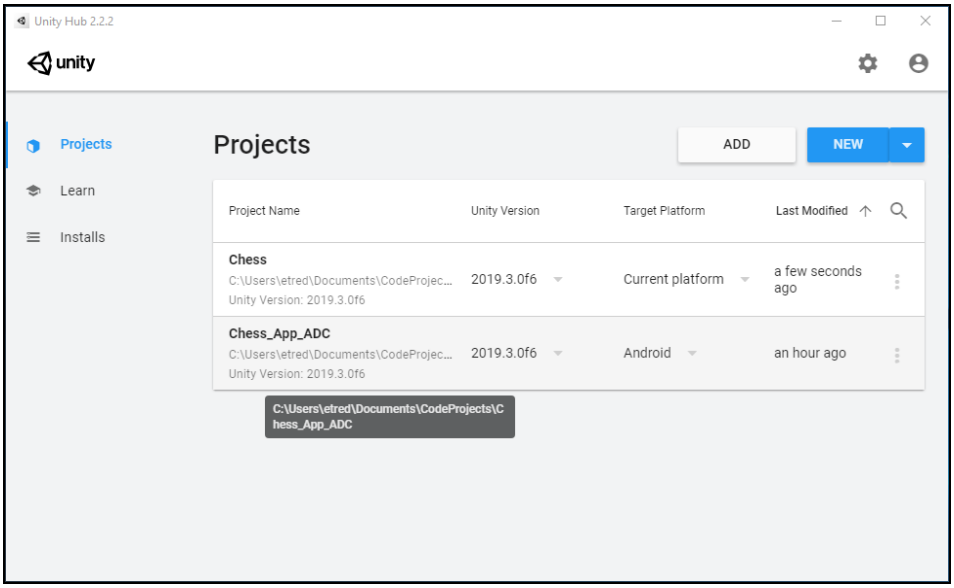
Prerequisites

-Unity installed (preferably version 2019.3 or newer)
-An IDE (code editor) that is attached to Unity
    -When installing Unity also install Visual Studio with it for it to work automatically
-The completed version: https://github.umn.edu/app-developers-club/Chess_App

Let's Get Started

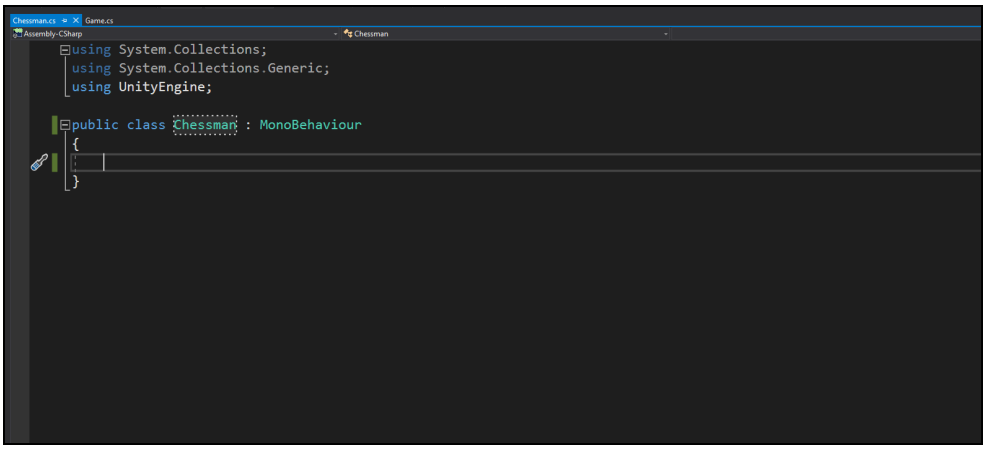
Open Unity Hub and open the Chess project:

- Set the Unity Version to the version of Unity you use
- Set the Target Platform to Android
    - Set to "Current platform" if you do not have an Android Phone
- Open Chess_App_ADC



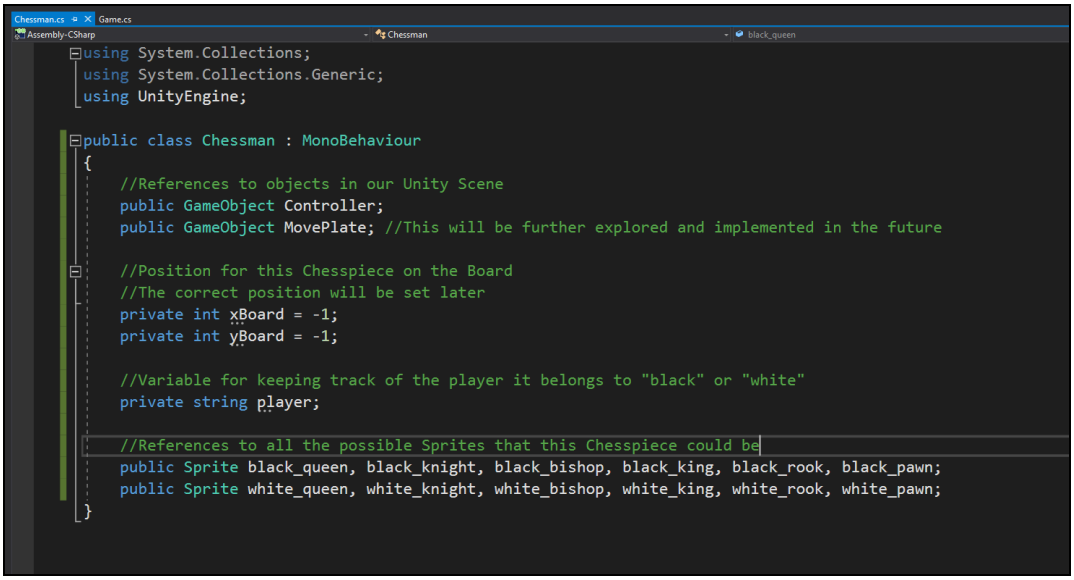
Create a new script and name it Chessman:

- Navigate to the “Scripts” folder
- Right click and select “Create” and then “C# Script”
- Rename it Chessman
- Click on it and it should open in Visual Studio
    - If this does not happen you can try installing Unity with Visual Studio or attaching your own preferred IDE to Unity
- Once it is opened, make sure it says “public class Chessman : MonoBehaviour” on line 5
- Delete the code inside the class



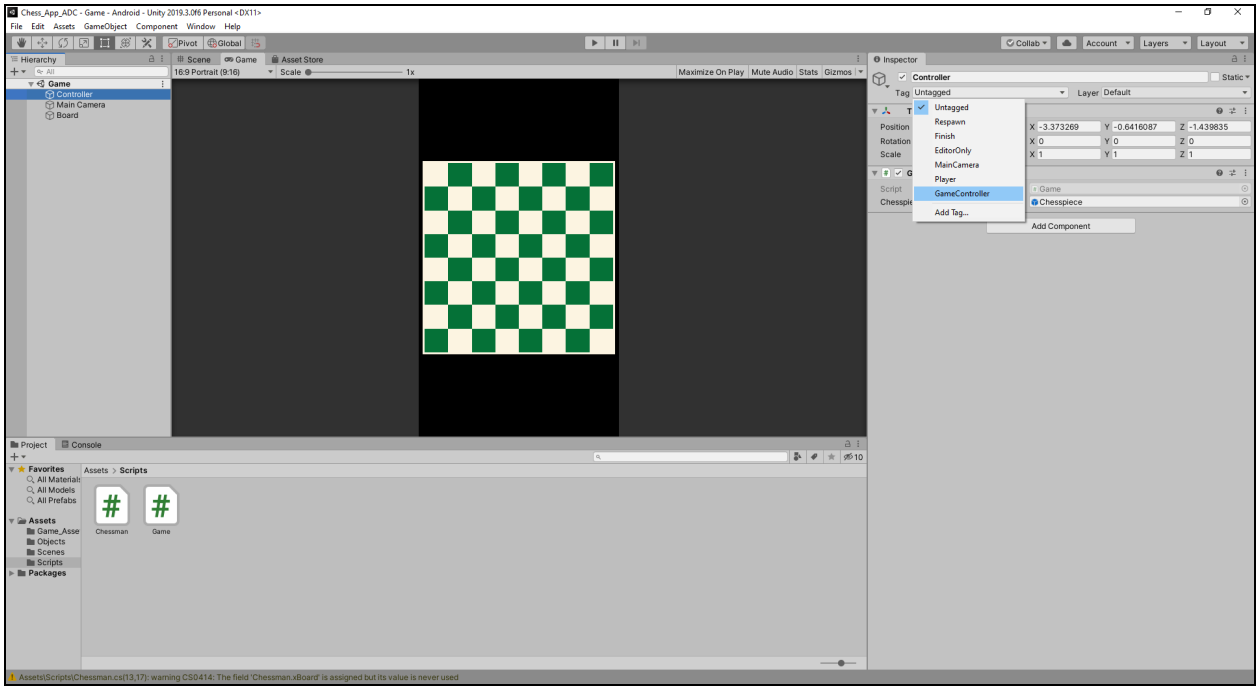
Now create the needed references and variables for the Chesspiece script:

- Keep in mind that this will later be attached as a component to the Chesspiece object
    - So we need to keep track of its position, which player it is associated with, etc.
    - We also need a way to access the different possible sprites it could use. This is generic to any Chesspiece so it could be a queen, a rook, a pawn, etc.
    - *Please Note:* ***Controller*** *and* ***MovePlate*** *should be* ***controller*** *and* ***movePlate***
    - *This is for standard coding conventions*



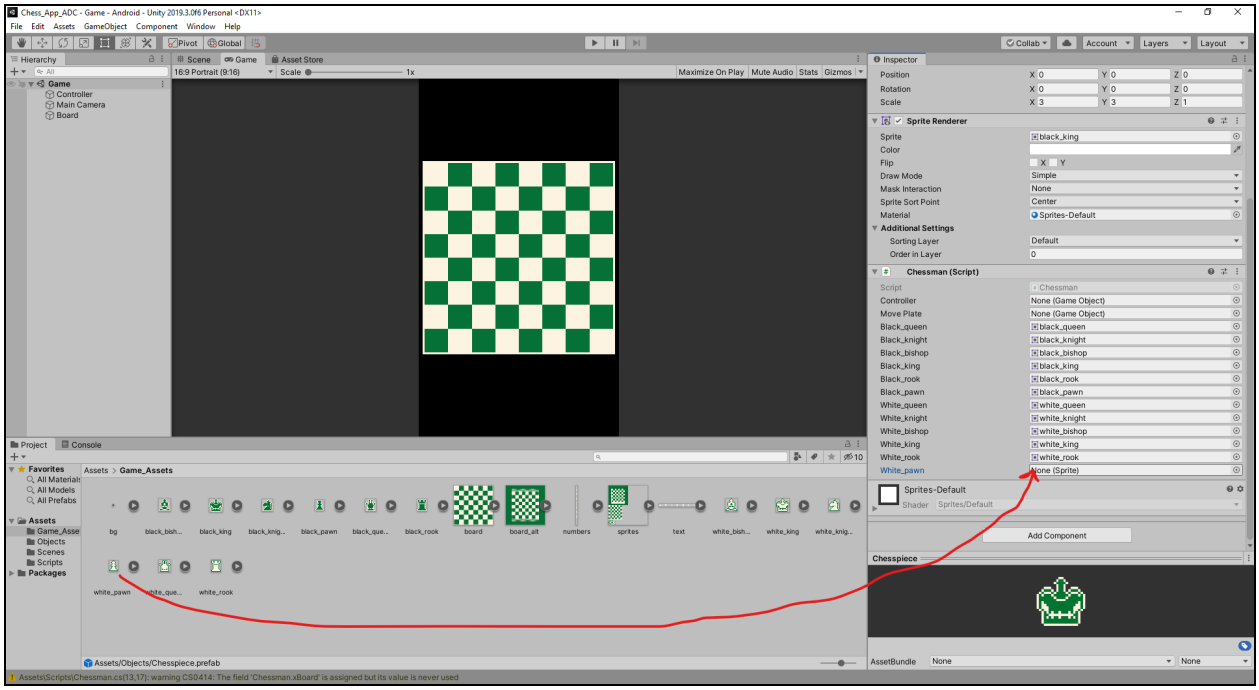
Return to Unity and add a tag to Controller:

- Since an instance of the Chesspiece does not exist when the game starts, we cannot simply drag the Controller to the Chessman script as we did before
- We have to make a reference for Controller in the form of a tag
- Click on Controller in the top left
- Select the “Untagged” drop down menu
- Select “GameController”



Add the Chessman script to the Chesspiece and fill in the sprites:

- Now navigate to the “Objects” folder and click on Chesspiece
- Click on “Add Component” near the bottom right corner
- Add the Chessman script (which you can simply search for in the text entry)
- Now there are a bunch of boxes that say None (Sprite)
    - We need to add the references for each of the Sprites
    - Above these you will see “Controller” and “Move Plate,” remember we will set up the Controller using a tag and we will add the Moveplate in the future
        - So we do not need to do anything about these two for now
- Click on the “Game_Assets” folder in the bottom left
- Drag the correct .png files into the boxes
    - Do not click on the assets or else the inspector will show information on them
    - You can always go back to Objects > Chesspiece



Add Activate() and SetCoords() functions to Chessman:

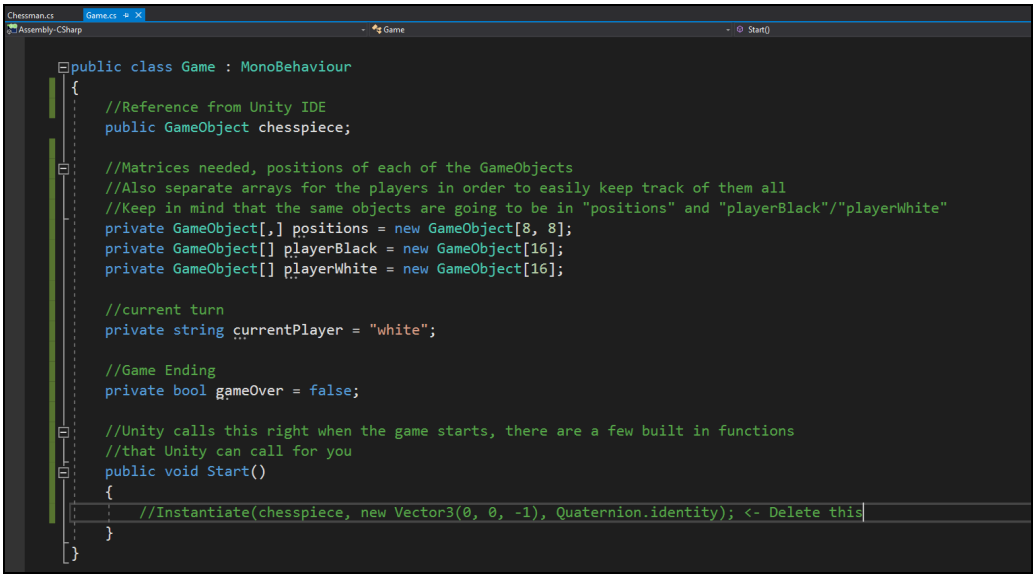
- Open the Chessman script again in Visual Studio and we will add two functions
- Activate()
    - This will set up the Chesspiece for us
    - It needs to get the reference for the Controller
    - Set the coordinates in world space (this will be done in another function implemented next)
    - Choose the correct sprite to use depending on which piece it is (i.e. black knight or white pawn)
        - As this object will be spawned by the controller, the controller will give the Chesspiece its name
        - We just need to take the given name and choose the correct sprite
- SetCoords()
    - We know the position on the board for this Chesspiece, it is xBoard and yBoard
        - However, this is based on the board
        - Position (0,0) on the board is not the same as position (0,0) in the Unity Scene
        - The position on the board is useful for us, but not for Unity
        - So we need to convert to what Unity calls “World Space”
        - The given float values and constants for x and y should work but change the numbers a little if it does not look right after running the project
    - SetCoords() needs to do some transformations in order to take the given position on the board and set it correctly on the screen when the game run
- If the following code looks confusing to you, I recommend reading about switch statements and generics (for “this.GetComponent<SpriteRenderer>().sprite = ...”)

```
public Sprite white_queen, white_knight, white_bishop, white_king, white_rook, white_pawn;

public void Activate()
{
    //Get the game controller
    Controller = GameObject.FindGameObjectWithTag("GameController");

    //Take the instantiated location and adjust transform
    SetCoords();

    //Choose correct sprite based on piece's name
    switch (this.name)
    {
        case "black_queen": this.GetComponent<SpriteRenderer>().sprite = black_queen; player = "black"; break;
        case "black_knight": this.GetComponent<SpriteRenderer>().sprite = black_knight; player = "black"; break;
        case "black_bishop": this.GetComponent<SpriteRenderer>().sprite = black_bishop; player = "black"; break;
        case "black_king": this.GetComponent<SpriteRenderer>().sprite = black_king; player = "black"; break;
        case "black_rook": this.GetComponent<SpriteRenderer>().sprite = black_rook; player = "black"; break;
        case "black_pawn": this.GetComponent<SpriteRenderer>().sprite = black_pawn; player = "black"; break;
        case "white_queen": this.GetComponent<SpriteRenderer>().sprite = white_queen; player = "white"; break;
        case "white_knight": this.GetComponent<SpriteRenderer>().sprite = white_knight; player = "white"; break;
        case "white_bishop": this.GetComponent<SpriteRenderer>().sprite = white_bishop; player = "white"; break;
        case "white_king": this.GetComponent<SpriteRenderer>().sprite = white_king; player = "white"; break;
        case "white_rook": this.GetComponent<SpriteRenderer>().sprite = white_rook; player = "white"; break;
        case "white_pawn": this.GetComponent<SpriteRenderer>().sprite = white_pawn; player = "white"; break;
    }
}
```

```
        case "white_bishop": this.GetComponent<SpriteRenderer>().sprite = white_bishop; player = "white"; break;
        case "white_king": this.GetComponent<SpriteRenderer>().sprite = white_king; player = "white"; break;
        case "white_rook": this.GetComponent<SpriteRenderer>().sprite = white_rook; player = "white"; break;
        case "white_pawn": this.GetComponent<SpriteRenderer>().sprite = white_pawn; player = "white"; break;
    }
}

public void SetCoords()
{
    //Get the board value in order to convert to xy coords
    float x = xBoard;
    float y = yBoard;

    //Adjust by variable offset
    x *= 0.66f;
    y *= 0.66f;

    //Add constants (pos 0,0)
    x += -2.3f;
    y += -2.3f;

    //Set actual unity values
    this.transform.position = new Vector3(x, y, -1.0f);
}
}
```

Let's spawn in all Chesspieces in their correct positions:

- We are going to need to add some arrays and variables to make this all work
- First let's add a 2D array to keep track of the board itself (this is used in conjunction with the xBoard and yBoard from the Chessman script)
- We also want two more arrays for each of the players as this will make it easy to spawn in the chesspieces
- When we start the game up, we want to spawn in all the Chesspieces through the two player arrays
    - This is for our convenience, we can then make a reusable function that sets the position in the actual board 2D array
    - First let's create the initial attributes needed to run the game

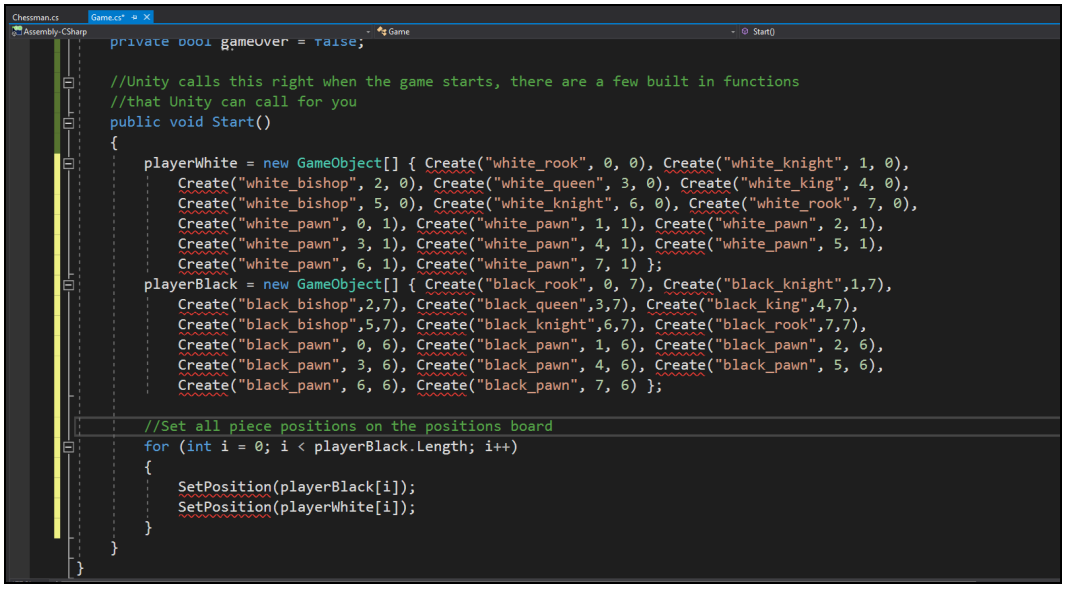


Let's add all of the starting positions to the arrays:

- We want to add all of the Chesspieces to the playerWhite and playerBlack arrays, however there are a bunch of things we need to take care of before we create the objects
    - The xBoard and yBoard need to be given real values
    - It needs to be named correctly so it can choose the correct sprite to use
- In order to do this we will call Create(...) which will return a GameObject, so we can still put this function inside of the array
- The long initialization is available here, Create takes in a name and the xBoard and yBoard location

```
playerWhite = new GameObject[] { Create("white_rook", 0, 0), Create("white_knight", 1, 0),
        Create("white_bishop", 2, 0), Create("white_queen", 3, 0), Create("white_king", 4, 0),
        Create("white_bishop", 5, 0), Create("white_knight", 6, 0), Create("white_rook", 7, 0),
        Create("white_pawn", 0, 1), Create("white_pawn", 1, 1), Create("white_pawn", 2, 1),
        Create("white_pawn", 3, 1), Create("white_pawn", 4, 1), Create("white_pawn", 5, 1),
        Create("white_pawn", 6, 1), Create("white_pawn", 7, 1) };
playerBlack = new GameObject[] { Create("black_rook", 0, 7), Create("black_knight",1,7),
        Create("black_bishop",2,7), Create("black_queen",3,7), Create("black_king",4,7),
        Create("black_bishop",5,7), Create("black_knight",6,7), Create("black_rook",7,7),
        Create("black_pawn", 0, 6), Create("black_pawn", 1, 6), Create("black_pawn", 2, 6),
        Create("black_pawn", 3, 6), Create("black_pawn", 4, 6), Create("black_pawn", 5, 6),
        Create("black_pawn", 6, 6), Create("black_pawn", 7, 6) };
```

- After this we also have to loop through all of these Chesspieces again in order to set their positions on the "positions" 2D array which holds their location on the Board
    - Each of the Chesspieces individually know where they are, however we will need to know where other Chesspieces are in order to know if they can move or capture
        - Using a matrix that holds all of the positions of each piece is useful for this even if it makes extra work for us to code it in
- Please type the code as-is, we will add Create() and SetPosition() next so the errors will go away



Implement a function to set up the Chesspiece correctly:

- Now we will make Create(), remember it takes in a name, x position, and y position
- All we have to do is create the Chesspiece and use Chesspiece functions in order to set it up (remember we already created Activate() to do this)
    - We will have to create setter functions for the Chesspiece
    - Navigate back to the Chesspiece script and add getters and setters (since xBoard and yBoard are private)

```
        this.transform.position = new Vector3(x, y, -1.0f);
    }

    public int GetXBoard()
    {
        return xBoard;
    }

    public int GetYBoard()
    {
        return yBoard;
    }

    public void SetXBoard(int x)
    {
        xBoard = x;
    }

    public void SetYBoard(int y)
    {
        yBoard = y;
    }
}
```

- Now time to implement Create() and SetPosition()

```
    }

    public GameObject Create(string name, int x, int y)
    {
        GameObject obj = Instantiate(chesspiece, new Vector3(0, 0, -1), Quaternion.identity);
        Chessman cm = obj.GetComponent<Chessman>(); //We have access to the GameObject, we need the script
        cm.name = name; //This is a built in variable that Unity has, so we did not have to declare it before
        cm.SetXBoard(x);
        cm.SetYBoard(y);
        cm.Activate(); //It has everything set up so it can now Activate()
        return obj;
    }

    public void SetPosition(GameObject obj)
    {
        Chessman cm = obj.GetComponent<Chessman>();

        //Overwrites either empty space or whatever was there
        positions[cm.GetXBoard(), cm.GetYBoard()] = obj;
    }
}
```

Now let's try it out:

- Press CTRL+S on both of the scripts to save them
- Return to Unity and click the play button in the top middle
- Wow look at that!
    - There are now Chesspieces on the board
    - We will be adding movement next time using MovePlates