Warena weekly blog ~ Daniel Fernández

Summary

During this log I am going to snip some extracts of code, but it might be more interesting to look into the GitHub, to be able to see more important commitments made in the Unity Editor, that can not be reflected here. During the development of this project I have programmed, or help in the programming of many other scripts in collaboration with the two other programmers that I will not include in this blog.

Week 1

I did not have a group in the day of the first pitch so, after the presentations, I talked with the already formed group of DeepMind, and they told me they still needed programmers. It was an ambitious project, but we were a lot of people in the group and very motivated with the idea.

That same afternoon we kept talking about the project plan, how would we distribute the tasks, the responsibilities of each member,... I was in charge, together with Alejandro, to research AR technology and find the best technology for our project.

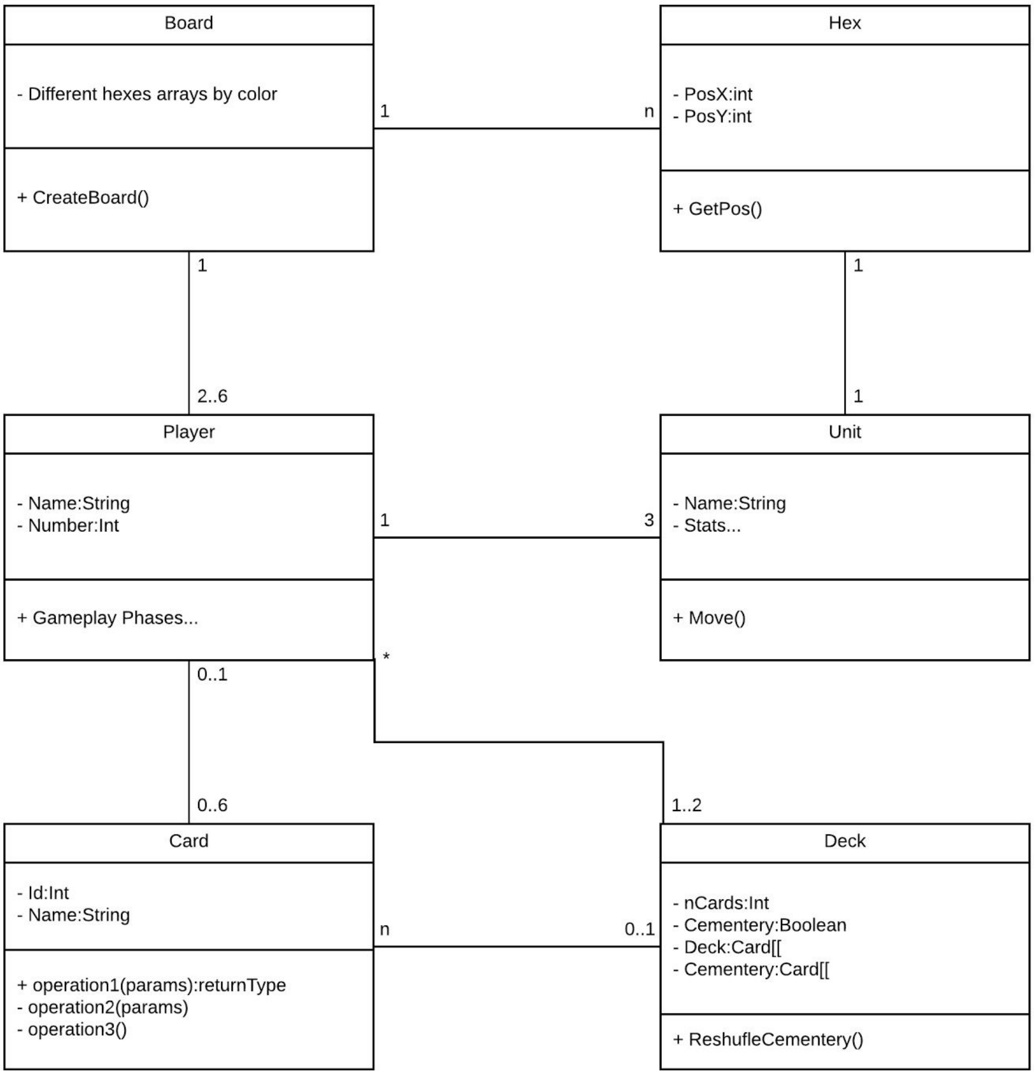
Week 2

We found out a plugin for Unity called Vuforia that pretty much filled all the requirements for out project. The decision of using Unity was already taken, as all we three programmers had some experience developing in this engine.

My task for this week was, along with Alejandro, to make a prototype in which we could see a 3D model with a phone camera using Vuforia. The documentation of the plugin is quite good, so we did not have much trouble to meet the deadline.

Week 3

For this week, I took the Game Manager task, and I started designing the classes, and the application flow.



I went back to Spain for Christmas holidays and I did not have the hardware to use Unity there, so I had to stick to the planning tasks for the upcoming weeks.

Week 4

While looking into the vuforia technology, I started to make a flowchart of the whole application, which had to be reviewed by the designers and then I applied the changes. This process was repeated several times until we had the final chart.

Week 5

As soon as I returned to Cologne, I started working on the Game Manager. To be honest, specifications from the designers and lead programmer were a bit vague, and in this point we had no idea how the networking was going to work. Because of this, my task started to take longer than expected, and I had to redo and restructure the whole Game Manager a few times.

This would be a first approach of it, where I programmed just a frame for other functions up to come, with some turn management logic.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GameManager : MonoBehaviour {

    public static GameManager instance = null;

    private int Map = 1;

    public PlayerScript Player1;
    public PlayerScript Player2;
    public BoardScript Board;

    public int nTurn;
    public int nPlayers;
    public int tTimer;
    public bool bNextPhase;
    public int nWinner;
    //      -1 -> No winner
    //       0 -> Tie
    // nPlayer -> Player "n" wins
    public bool exit;

    private Vector2 touchOrigin = -Vector2.one; //Used to store location of screen touch origin for mobile controls.

    //Awake is called before Start function

    void Awake() {
        do {
            nTurn = 1;
            nPlayers = 2;
            tTimer = 0;
            bNextPhase = false;
            nWinner = -1;

            StartCoroutine(Timer());

            Player1 = new PlayerScript();
            Player2 = new PlayerScript();

            Setup();

            do {
                for (int i = 0; i < nPlayers; i++) {
                    //Changing functions into coroutines
                    DrawPhase(i);
                    PlanningPhase(i);
                    ExecutePhase(i);

                    nWinner = CheckWinCondition();
                    if (nWinner == 0) {
                        //Display Tie
                    } else if (nWinner == 1) {
                        //Display winner 1
                    } else if (nWinner == 2) {
                        //Display winner 2
                    }
                }
            } while (nWinner == -1);
            exit = CheckPlayAgain();
        } while (!exit);
    }


public void Setup() {
    UnitScript Unit;
    UnitScript[] UnitsTemp = new UnitScript[5];
    HexScript Hex;
    //Players place the units
    for (int i = 0; i < (nPlayers * 3 - 1); i++) {
        for (int j = 1; j <= nPlayers; j++) {
            Unit = new UnitScript(0);
            if (j == 1) {
                Hex = Player1.PlanMovement(Unit);
                Player1.ApplyMovement(Unit, Hex);
                UnitsTemp[i] = Unit;
            } else if (j == 2) {
                Hex = Player2.PlanMovement(Unit);
                Player2.ApplyMovement(Unit, Hex);
                UnitsTemp[i] = Unit;
            }
            i++;
        }
    }
    //Players claim the placed units
    for (int i = 0; i < 5; i++) {
        for (int j = 1; j <= nPlayers; j++) {
            if (j == 1) {
                UnitsTemp[i].player = 1;
            } else if (j == 2) {
                UnitsTemp[i].player = 2;
            }
        }
    }
    //Last player place and claim the last unit
    Unit = new UnitScript(0);
    Hex = Player2.PlanMovement(Unit);
    Player1.ApplyMovement(Unit, Hex);
    UnitsTemp[nPlayers * 3].player = 2;
}

//Shows a message saying which player is playing this turn
public void NextPlayer(int nPlayer) {

}

public void DrawPhase(int nPlayer) {
    if (nPlayer == 1)
        Player1.DrawPhase();
    else if (nPlayer == 2)
        Player2.DrawPhase();
}

public void PlanningPhase(int nPlayer) {
    if (nPlayer == 1)
        Player1.DrawPhase();
    else if (nPlayer == 2)
        Player2.DrawPhase();
}

public void ExecutePhase(int nPlayer) {
    if (nPlayer == 1)
        Player1.DrawPhase();
    else if (nPlayer == 2)
        Player2.DrawPhase();
}

public int CheckWinCondition() {
    int win = 0;
    int unitsleft1 = 0;
    int unitsleft2 = 0;
    for (int i = 0; i < Player1.Units.Length; i++)
        if (Player1.Units[i] != null)
            unitsleft1++;
    for (int i = 0; i < Player2.Units.Length; i++)
        if (Player1.Units[i] != null)
            unitsleft2++;

    if (unitsleft1 == 0 && unitsleft1 == unitsleft2)
        win = -1;   //tie
    else if (unitsleft1 == 0)
        win = 2;
    else if (unitsleft2 == 0)
        win = 1;
    else
        win = 0;

    return win;
}

//Ask players if they want to play another game
public bool CheckPlayAgain() {
    return false;
}

IEnumerator Timer() {
    do {
        yield return new WaitForSeconds(1);
        tTimer++;
    } while (true);
}
}
```

This was my first approach for the touch control:

```
// Touch Controls
void Update() {
    int horizontal = 0;     //Used to store the horizontal move direction.
    int vertical = 0;       //Used to store the vertical move direction.
                            //Check if Input has registered more than zero touches
    if (Input.touchCount > 0) {
        //Store the first touch detected.
        Touch myTouch = Input.touches[0];

        //Check if the phase of that touch equals Began
        if (myTouch.phase == TouchPhase.Began) {
            //If so, set touchOrigin to the position of that touch
            touchOrigin = myTouch.position;
        }

        //If the touch phase is not Began, and instead is equal to Ended and the x of touchOrigin is greater or equal to zero:
        else if (myTouch.phase == TouchPhase.Ended && touchOrigin.x >= 0) {
            //Set touchEnd to equal the position of this touch
            Vector2 touchEnd = myTouch.position;

            //Calculate the difference between the beginning and end of the touch on the x axis.
            float x = touchEnd.x - touchOrigin.x;

            //Calculate the difference between the beginning and end of the touch on the y axis.
            float y = touchEnd.y - touchOrigin.y;

            //Set touchOrigin.x to -1 so that our else if statement will evaluate false and not repeat immediately.
            touchOrigin.x = -1;

            //Check if the difference along the x axis is greater than the difference along the y axis.
            if (Mathf.Abs(x) > Mathf.Abs(y))
                //If x is greater than zero, set horizontal to 1, otherwise set it to -1
                horizontal = x > 0 ? 1 : -1;
            else
                //If y is greater than zero, set horizontal to 1, otherwise set it to -1
                vertical = y > 0 ? 1 : -1;
        }
    }
}
```

In the end of the week, we had a meeting where we issued all this communication problems, so there should stop being a problem in the future.

Week 6

At the start of the week we made another meeting now that everyone was in Cologne. We talked about the communication problems once again, and more importantly, about the work of this and the upcoming weeks. Because the Intermediate presentation was in a couple of days, we decided to do a more simple scene (with a different game manager only for the presentation).

To do that I had to include code from Alejandro and Dimitri. I had no problem to implement the first's code, but I struggled for a while with Dimitri's, because of the lack of comments and no specifications in the Tech Document. He had to implement some changes with his pathfinding because it was thought to work with a static scene, and it had to support movement because of Vuforia.

I set up the whole scene and the custom Game Manager for the presentation. In this build, the player was able to select a unit from a side menu, place it on the field and move the unit around it.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class GameManagerSimplified : MonoBehaviour {

    public int nTurn;
    public int nPhase;
    public int nPlayers;
    public int nUnit;
    public int tTimer;
    public bool timerRunning;
    public bool bNextPhase;
    public int nWinner;
    //      -1 -> No winner
    //       0 -> Tie
    // nPlayer -> Player "n" wins
    public bool exit;

    public GameObject myUnit;
    public GameObject Octo;
    public GameObject Scampi;
    public GameObject Crab;

    public Text TimerText;

    //private Vector2 touchOrigin = -Vector2.one; //Used to store location of screen touch origin for mobile controls.

    void Start () {
        Debug.Log("Starting Game");

        nTurn = 1;
        nPhase = 0;
        nPlayers = 2;
        nUnit = 0;
        tTimer = 0;
        bNextPhase = false;
        timerRunning = false;
        nWinner = -1;
        Octo = GameObject.Find("Octo_base");
        Scampi = GameObject.Find("scampi_base");
        Crab = GameObject.Find("Crab_base");

        StartCoroutine(Timer());

    }

    void Update () {
        TimerText.text = tTimer.ToString();
    }

    //Change into SelectUnit(int nUnit)
    public void PlanningPhase(int nUnit) {
        Debug.Log("Planing Phase");
        this.nUnit = nUnit;
        Debug.Log("Unit selected: "+nUnit);
    }

    //Change into PlaceUnit()
    public void MovePhase() {
        if (nPhase == 0) {
            nPhase = 1;
            if (nUnit != 0) {
                Debug.Log("Move Phase");
                if (nUnit == 1) {
                    myUnit = Octo;
                }
                if (nUnit == 2) {
                    myUnit = Scampi;
                }
                if (nUnit == 3) {
                    myUnit = Crab;
                }
                myUnit.transform.parent = GameObject.Find("ImageTarget 1").transform;
                myUnit.transform.localPosition = new Vector3(0, 0, 0);
                myUnit.transform.localRotation = new Quaternion(0, 0, 0, 0);
                myUnit.transform.localScale = new Vector3(0.2f, 0.2f, 0.2f);
            }
        }
    }

    public void ResetTimer() {
        tTimer = 0;
    }

    IEnumerator Timer() {
        do {
            timerRunning = true;
            yield return new WaitForSeconds(1);
            tTimer++;
        } while (true);
    }

    public int GetTimer() {
        return tTimer;
    }
}
```

In the end, the project managers decided not to use the application for the presentation because of stability problems (which, in my point of view, were minimal and completely understandable for a very early prototype).

After the presentation, I we made another meeting in which we had to decide the future of the project. I was asked to design a plan for the game to work in a single device, so we do not need to spend time on the networking.

Here the document with the "road plan"

Week 7

In this week I continued to work in the Game Manager. The decision if the game would be running on a single device or not was still to be decided, so I started doing it as if it was going to be for two players, because then it would be easier to change it into the networking option than the other way around.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using ARTCards;
using HexMap;

public class GameManager : MonoBehaviour {

    private GameObject startMenuUI;
    private GameObject mainUI;
    private GameObject atributeCardInjectorUI;
    private GameObject askToUseCardUI;
    private GameObject finishPlanningUI;
    private GameObject mainCamera;
    private GameObject ARCamera;
    private GameObject Board;

    private Player player1;
    private Player player2;
    private GameObject imageTarget;
    private GameObject[] unclaimedUnits;
    private GameObject tempUnit;
    private RaycastHit plannedRaycast1;
    private RaycastHit plannedRaycast2;

    private TurnManager turnManager;

    private IEnumerator timer;

    private bool inSetup;
    private bool inPlacing;
    private bool inClaiming;
    private bool allowCardScanning;
    private bool wantToInjectCard;
    private bool unitSelected;
    private bool inExecute;
    private int tTimer;
    private int nTurn;
    private int playerTurn;
    private int index1;
    private int index2;
    private int selectedUnit;

    private FieldOrientationAssistant assist;

    void Start() {
        startMenuUI = GameObject.Find("canvas_start_menu");
        mainUI = GameObject.Find("canvas_main_UI");
        atributeCardInjectorUI = GameObject.Find("canvas_CardScanning");
        askToUseCardUI = GameObject.Find("canvas_AskToUseCard");
        finishPlanningUI = GameObject.Find("canvas_FinishPlanning");
        mainCamera = GameObject.Find("Main Camera");
        ARCamera = GameObject.Find("ARCamera");
        Board = GameObject.Find("Board");
        imageTarget = GameObject.Find("ImageTarget_Grid");
        assist = FindObjectOfType<FieldOrientationAssistant>();

        mainUI.SetActive(false);
        atributeCardInjectorUI.SetActive(false);
        askToUseCardUI.SetActive(false);
        finishPlanningUI.SetActive(false);
        ARCamera.SetActive(false);
        Board.SetActive(false);

        player1 = new Player();
        player2 = new Player();
        unclaimedUnits = new GameObject[6];
        tempUnit = new GameObject();

        timer = Timer();

        inExecute = false;
        inSetup = false;
        inPlacing = true;
        inClaiming = false;
        allowCardScanning = false;
        unitSelected = false;
        tTimer = 0;
        playerTurn = 1;
        index1 = 0;
        index2 = 0;
        nTurn = 1;
        selectedUnit = 0;
    }

    void Update() {
        if (Input.GetMouseButtonDown(0) && Input.touchCount < 2) {
            RaycastHit hit = new RaycastHit();
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            if (Physics.Raycast(ray, out hit)) {
                Debug.Log("hit " + hit.point + ". In object: " + hit.collider.gameObject);
                //1.- Setup
                if (inSetup) {
                    Debug.Log("In Setup");
                    if (inPlacing) {
                        Debug.Log("In placing");
                        if (playerTurn == 1) {
                            Debug.Log("Player1 has placed an unit");
                            unclaimedUnits[index1 + index2] = (GameObject)Instantiate(Resources.Load("Scampi"));
                            int hexid = HexGrid.instance.GetCellId(assist.WorldToGrid(hit.point));
                            Debug.Log("There are "+HexGrid.instance.positions.Length+" coordinates and id is "+hexid);
                            Vector3 pos = assist.GridToWorld(HexGrid.instance.positions[hexid]);
                            unclaimedUnits[index1 + index2].gameObject.transform.position = pos;
                            unclaimedUnits[index1 + index2].transform.parent = imageTarget.transform;
                            unclaimedUnits[index1 + index2].gameObject.transform.rotation = imageTarget.transform.rotation;
                            unclaimedUnits[index1 + index2].gameObject.transform.localScale = new Vector3(0.5f, 0.5f, 0.5f);
                            index1++;
                        }
                        if (playerTurn == 2) {
                            Debug.Log("Player2 has placed an unit");
                            unclaimedUnits[index1 + index2] = (GameObject)Instantiate(Resources.Load("Scampi"));
                            int hexid = HexGrid.instance.GetCellId(assist.WorldToGrid(hit.point));
                            Debug.Log("There are " + HexGrid.instance.positions.Length + " coordinates and id is " + hexid);
                            Vector3 pos = assist.GridToWorld(HexGrid.instance.positions[hexid]);
                            unclaimedUnits[index1 + index2].gameObject.transform.position = pos;
                            unclaimedUnits[index1 + index2].transform.parent = imageTarget.transform;
                            unclaimedUnits[index1 + index2].gameObject.transform.rotation = imageTarget.transform.rotation;
                            unclaimedUnits[index1 + index2].gameObject.transform.localScale = new Vector3(0.5f, 0.5f, 0.5f);
                            index2++;
                        }
                        ChangeTurn();
                    } else if (inClaiming) {
                        Debug.Log("In claiming");
                        if (hit.collider.gameObject.tag == "Unit") {
                            Debug.Log("Unit Claimed");
                            if (playerTurn == 1) {
                                //Check if unit selected has owner already, if not, claim it
                                if (unclaimedUnits[index1 + index2].GetComponent<UnitController>().unit.player == 0) {
                                    player1.units[index1].player = 1;
                                    player1.units[index1] = unclaimedUnits[index1 +
index2].GetComponent<UnitController>().unit;
                                }
                                index1++;
                            }
                            if (playerTurn == 2) {
                                    //Check if unit selected has owner already, if not, claim it
                                if (unclaimedUnits[index1 + index2].GetComponent<UnitController>().unit.player == 0) {
                                    player2.units[index2].player = 2;
                                    player1.units[index1] = unclaimedUnits[index1 +
index2].GetComponent<UnitController>().unit;
                                }
                                index2++;
                            }
                            ChangeTurn();
                        }
                    }
                    if (index1 >= 3 && index2 >= 3) {
                        index1 = index2 = 0;
                        if (inPlacing) {
                            inPlacing = false;
                            inClaiming = true;
                        } else {
                            inSetup = false;
                            finishPlanningUI.SetActive(true);
                        }
                    }
                } else {    //Normal turn
                    Debug.Log("In draw/activate card");
                    //2- Turns start: physical attribute card draw
                    //Ask the player if wants to inject. We activate the AskUseCard Canvas
                    Debug.Log("Do you want to inject an attribute card?");
                    askToUseCardUI.SetActive(true);
                    if (playerTurn == 1) {
                        //scanning of the card: the Scanned Card Activator calls
                        if (wantToInjectCard) {
                            //activate injection process: put the car in front
                            //of the camera to begin the process
                            atributeCardInjectorUI.SetActive(true);
                        }
                        //3- Planning phase
                        if (!unitSelected) {
                            if (hit.collider.gameObject.tag == "Unit" &&
hit.collider.gameObject.GetComponent<UnitController>().unit.player == 1) {
                                Debug.Log("Unit Selected");
                                tempUnit = hit.collider.gameObject;
                                unitSelected = true;
                            }
                        } else {
                            Debug.Log("Path created");
                            tempUnit.GetComponent<UnitController>().PreparePath(hit.point);
                            unitSelected = false;
                        }
                    }
                    if (playerTurn == 2) {
                        //scanning of the card: the Scanned Card Activator calls
                        if (wantToInjectCard) {
                            //activate injection process: put the car in front
                            //of the camera to begin the process
                            atributeCardInjectorUI.SetActive(true);
                        }
                        //3- Planning phase
                        if (!unitSelected) {
                            if (hit.collider.gameObject.tag == "Unit" &&
hit.collider.gameObject.GetComponent<UnitController>().unit.player == 2) {
                                Debug.Log("Unit Selected");
                                tempUnit = hit.collider.gameObject;
                                unitSelected = true;
                            }
                        } else {
                            Debug.Log("Path created");
                            tempUnit.GetComponent<UnitController>().PreparePath(hit.point);
                            unitSelected = false;
                        }
                    }
                    // Execute phase
                    if (inExecute) {
                        Debug.Log("EXECUTE PHASE");
                        finishPlanningUI.SetActive(false);
                        //Move units
                        tempUnit.GetComponent<UnitController>().StartPath();
                        tempUnit.GetComponent<UnitController>().FollowPath();

                        //Attack sub-phase. Check the initiative!!
                        for (int i = 0; i < player1.units.Length; i++) {
                            //Check here if unit is in range with other enemy units
                            //if (player1.units[i].attrs.TryGetValue("Range"))
                        }
                        for (int i = 0; i < player2.units.Length; i++) {
                            //Check here if unit is in range with other enemy units
                            //if (player2.units[i].attrs.TryGetValue("Range"))
                        }

                        finishPlanningUI.SetActive(true);

                        //Check winning condition
                        if (!inSetup && player1.units.Length <= 0) {
                            Debug.Log("Player 1 wins");
                            finishPlanningUI.SetActive(true);
                        } else if (!inSetup && player1.units.Length <= 0) {
                            Debug.Log("Player 2 wins");
                            finishPlanningUI.SetActive(true);
                        }
                        askToUseCardUI.SetActive(false);
                        nTurn++;
                        ChangeTurn();
                    }
                }//End setup-normalTurn
            }
        }
    }

    public void FindGame() {
        Debug.Log("Finding Game");
        startMenuUI.SetActive(false);
        StartGame();
    }

    public void StartGame() {
        Debug.Log("Starting Game");
        mainUI.SetActive(true);
        mainCamera.SetActive(false);
        ARCamera.SetActive(true);
        Board.SetActive(true);
        inSetup = true;
    }

    public void ReturnToMenu() {
        mainUI.SetActive(false);
        mainCamera.SetActive(true);
        ARCamera.SetActive(false);
        Board.SetActive(false);
    }

    private void ChangeTurn() {
        if (playerTurn == 1)
            playerTurn = 2;
        else
            playerTurn = 1;

        Debug.Log("Turn of player " + playerTurn);
    }

    public void SelectUnit(int nUnit) {
        Debug.Log("You have selected your unit number " + nUnit);
        selectedUnit = nUnit;
    }

    /// <summary>
    /// Called when the button "accept, i want to inject a card"
    /// is pressed. Starts the injection process, first with scanning
    /// </summary>
    public void WantToInjectCard() {
        Debug.Log("Player want to use card");
        wantToInjectCard = true;
    }

    /// <summary>
    /// Called when the player press "No, do not inject any att card"
    /// </summary>
    public void DoNotWantToInjectCard() {
        Debug.Log("Player does not want to use card");
        wantToInjectCard = false;
    }

    public void StartExecutePhase() {
        inExecute = true;
    }

    public void StartTimer() {
        if (tTimer != 0)
            StopCoroutine(timer);
        tTimer = 0;
        StartCoroutine(timer);
    }
    IEnumerator Timer() {
        do {
            yield return new WaitForSeconds(1);
            tTimer++;
        } while (true);
    }
}
```

This time, I started to include the game mechanics, such as unit placement, turn management,... Everything to make the game playable but the attack functions, because Dimitri was still working in them.

In the meantime, I helped Alejandro finish his card scanning scene and scripts. It was a very frustrating task, because we couldn’t find a way to make an event to detect when a card has been scanned, the Vuforia plugin is not thought to be used in that way and we couldn’t find any information in the internet about that at all. In the end, I realised that we could get it by checking if the mesh renderer of an object in the “image target” was active or not. It was a rather strange solution, but we decided not to waste more time in that and focus on finishing the task.

Week 8

In this week, Alex and me were able to finish the last week's task about the card scanning scripts, and we both continued to work in the game manager. In the end, we had a separate project with the card scanning feature (we did so to try to figure out the errors we had with the Vuforia plugin), that we then included into the main project, where Dimitry had been working to improve the Game Manager.

Week 9

The whole group met on Monday to give a final push to the project. In that day we could, at least, get the Round Manager working properly, meaning that we could play a whole game from start to finish.

We updated the Vuforia database with the final images of the cards and included the final assets in the game too.

The next day we put the game in a new project to get rid of the unnecessary assets and plugins. Alejandro and me included the credits menu, a placeholder for the pause menu and improved some other minor aspects like the scale of the units.

The day of the presentation and the day after we continued to make some other minor improvements and made the final build so the game could be tested by everyone at CGL.