SGMY Group Assignment

Date: 20 May, 2017

Marek Dvoracek, 239926

Supervisor: Kasper Knop Rasmussen

Table of Contents

1. Concept 3
2. Implementation 4
   - UI 4
   - Displaying star information 5
   - Connecting stars 6
3. Conclusion 8

1. Concept

This project is a simple game in space. Player basically sees star constellations from Earth's point of view and is able to rotate and zoom in/out the camera. Player can also tap on stars he can see, and he has two types of taps available. First one is just tap, when tapped two or more stars of one constellation, they will connect with bright line, showing they belong to the same constellation. Once player has found all of the stars in one constellation, its name will pop up. The other type of tap is tap-and-hold, where player can hold his finger on a star and information about the star will pop up in simple UI. In this UI, there can be seen information like temperature of the star, it's distance from our Solar System and, of course, brief description of the star.

There were also some conditions for us to fulfill. Since we needed this game to teach children something, we chose something really interesting and maybe, for children now, more important than for us. The field of this project, as can be deducted from description above, is Astronomy. Learning goal here is pretty simple, this game aims to educate children about stars and constellations, how humans are connecting stars in constellations for thousands of years and how they are seen from Earth and even how big or small some other stars are, compared to Sun. Game also implements real star colors based on their temperature, which vary from blue to red. The thing here is, that from Earth, these colors are distorted by Earth's atmosphere and more, we couldn't see these with just human eye.

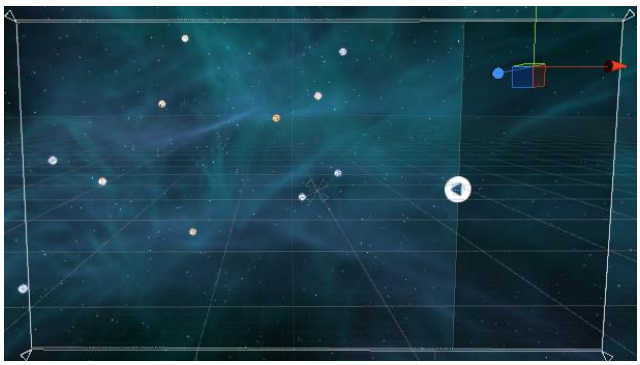
2. Implementation

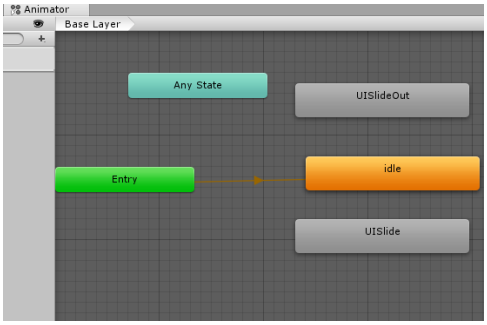
In the implementation part, I would like to say, that I have something common with basically everything that happened during period we were developing the game, but my main responsibility was connecting stars into constellations, drawing line between connected stars if they are from the same constellation and selecting a star to see more information about it.

In this part, Unity Collaborate was very helpful tool for sharing all the changes every single one of us made. I would say it worked even better than coping with Git, because Collaborate is implemented in Unity and after the simplest change in code it was just two clicks away from being usable by everyone.

- **UI**

When we started implementing, we knew we need two things. These were to set up and populate our scene with at least one constellation and some simple UI. While doing UI, we have first done very simple panel, that just slides in and out using animations triggered by tapping on a UI button. Then, we added some text fields and GameObjects into the actual UI, which was first problem we experienced during implementation. This problem was in render settings of UI Canvas. The only thing we needed to do was just set render settings to "Screen Space – Camera" instead of "Screen Space – Overlay". This allowed us to attach 3D GameObjects to UI and display them in there. The difference before and after can be seen on pictures below respectively.



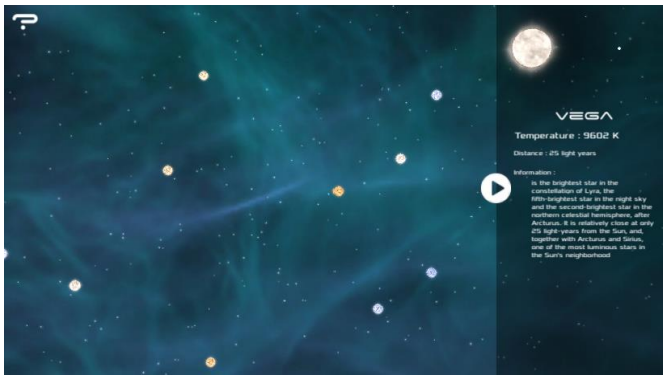


With basic UI and populated scene, I went straight for implementing selecting stars and connecting them with line, when they are from the same constellation, while others were working on camera controls.

- **Displaying star information**

First, I would analyze how I coped with selecting a star to display more info for as selecting all the stars is much more complicated. To view information about a star as well as it's model is done using TouchScript, more in depth with Long Press Gesture, where every star in the scene has this component inside. This script works the way, that player taps and holds his/her finger on a star for one second, another script called Star Taps then causes all the action in method LongPress, which takes two arguments in cooperation with TouchScript scripts. So when LongPress method is called inside Star Taps script, it selects tapped star and retrieves all the information, like name, distance, temperature, description and even its model, from the star and replaces it with current information in the UI. This piece of code can be seen in picture below.

```
void LongPress(object s, EventArgs arg0) {
    GameObject holder = GameObject.Find("Star Holder");
    Destroy(holder.transform.GetChild(0).gameObject);
    GameObject newOne = Instantiate(gameObject, holder.transform);
    newOne.transform.localPosition = new Vector3(0,0,0);
    newOne.transform.localScale = new Vector3(scaleToSun, scaleToSun, scaleToSun);
    GameObject.Find("Star Name").GetComponent<Text>().text = gameObject.name;
    GameObject.Find("Star Distance").GetComponent<Text>().text = "Distance : " + distance + " light years";
    GameObject.Find("Star Temperature").GetComponent<Text>().text = "Temperature : " + GetComponent<Star>().temperatureKelvin + " K";
    GameObject.Find("Star Description").GetComponent<Text>().text = description;
    GameObject.Find("Roll Button").GetComponent<UIRoll>().Show();
}
```



Two pictures above show what actually happens, when star Vega is selected to show information.

- **Connecting stars**

So basically, script StarTaps is redirecting the work to StarCollector script, which handles adding stars to an array of selected stars and this handles method called AddStar, which takes one argument – selected (tapped) star.

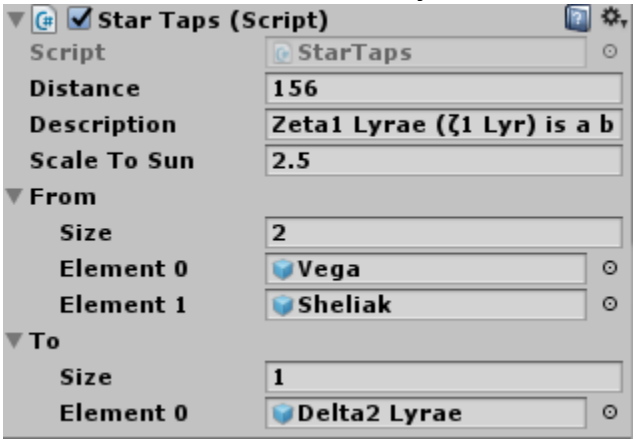
```
public void addStar(GameObject star) {
    if (isSelected(star)) {
        return;
    }
    if (selectedStars[0] == null){
        selectedStars[0] = star;
        index++;
    }
    else if (selectedStars[0].transform.parent == star.transform.parent){
        selectedStars[index] = star;
        index++;
    }
    else if (selectedStars[0].transform.parent != star.transform.parent){
        ClearSelectedStars(star);
    }
    if (size() == selectedStars[0].transform.parent.childCount) {
        GameObject.Find("ConstellationName").GetComponent<Text>().text = selectedStars[0].transform.parent.name;
        anim.CrossFade("NameShow", 1f, 0);
    }
}
```

So, as we can see in picture above, this method first checks, if the star is already selected. If it is, then it returns and does nothing. Then, it checks, if the star that has been selected is in the same constellation as previously selected stars. This is done easily thanks to Unity's hierarchy system, where constellations are empty objects, parenting star objects. This way we can determine in very efficient way where does the star belong. If the star is in the same constellation, then add it to the array. However, if it isn't from the same constellation, clear the array and add selected star to the cleared array. For player, it may seem like nothing happened, which we wanted to fix and let them know, that they are selecting different constellation, but we had more important functionality to cope with and sadly, we didn't have enough time to finish this part as well. The last check in this piece of code is looking for all the stars selected and if number of selected stars is the same as the number of stars in constellation, then display name of the constellation.

Now, when we have done adding stars to a list, we can manage connecting stars with lines. To connect one star with another we are doing a few things:

- On tap, add tapped star to StarCollector script in ClickedCollector object in scene
- Check if in StarCollector has any star to connect tapped one with (stars have predefined routes, so they can't just connect with any other star)
    - If there is any star to connect with, create Line Renderer component for the star beam will go from and set it to point to the other star
    - It there isn't star to connect it with, just start coroutine waiting for notify from other stars it can be connected to.

In the picture below, you can see Star Taps script added as component to every star. So every star has its own array of stars it can receive beams from and array of stars it can fire beam to.



However, problem arises when star has more than one star it can render line to, because Line Renderer needs to have 3 positions and the middle one has to be the actual star, that has size of To array greater than 1. This is solved easily by a few conditions in the code. The biggest problem, and I am aware it's still buggy in some constellations is to notify some of the stars to connect correctly. Following pictures show how this is realized:

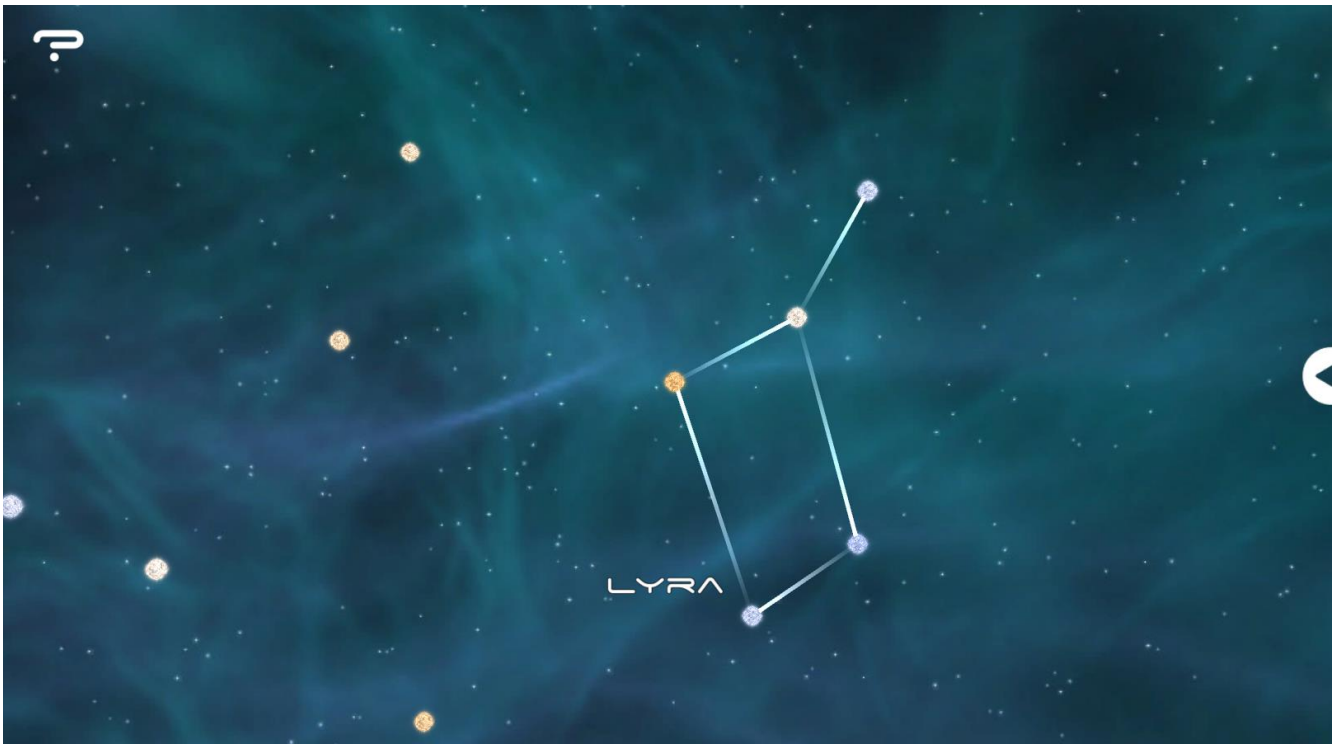
```
private void ConnectTo(){
    for (int i = 0; i < starCollector.GetComponent<StarCollector>().size(); i++){
        for (int j = 0; j < to.Length; j++){
            if (to[j] != null){
                if (to[j] == starCollector.GetComponent<StarCollector>().getStar(i)){
                    if (to.Length == 1){
                        SetUpLine(2);
                        line.SetPosition(0, gameObject.transform.position);
                        line.SetPosition(1, to[j].transform.position);
                    }
                    else if (to.Length == 2 && actualTo == 1){
                        SetUpLine(3);
                        line.SetPosition(0, to[0].transform.position);
                        line.SetPosition(1, gameObject.transform.position);
                        line.SetPosition(2, to[1].transform.position);
                    }
                    else if (to.Length == 2 && actualTo == 0) {
                        SetUpLine(2);
                        line.SetPosition(0, gameObject.transform.position);
                        line.SetPosition(1, to[j].transform.position);
                        actualTo++;
                    }
                }
            }
        }
    }
}

private void ConnectFrom() {
    for (int i = 0; i < starCollector.GetComponent<StarCollector>().size(); i++){
        for (int j = 0; j < from.Length; j++){
            if (from[j] != null){
                if (from[j] == starCollector.GetComponent<StarCollector>().getStar(i)){
                    from[j].GetComponent<StarTaps>().Notify();
                }
            }
        }
    }
}
```

```
private void SetUpLine(int positions) {
    if (line == null) {
        line = gameObject.AddComponent<LineRenderer>();
    }
    line.positionCount = positions;
    line.widthMultiplier = 25f;
    line.material = new Material(Shader.Find("Particles/Additive"));
    line.SetColors(c1, c2);
}

IEnumerator WaitForNotify() {
    while (!connect) {
        yield return new WaitForSeconds(.1f);
    }
    ConnectTo();
    if (to.Length == 1) goto end;
    while (!connect1){
        yield return new WaitForSeconds(.1f);
    }
    ConnectTo();
    end:
    yield return null;
}
```

The result looks like this:



3. Conclusion

I believe we have successfully implemented all the necessary functions and some additional ones for the game to fulfill the condition to be serious, by aim to educate players about the stars and displaying information about them as well as showing constellations. Next, I also believe our game is suitable for players in age from 8 to 12 years, thanks to easy and intuitive control system in combination with the table the game is developed for. Lastly, I don't think the game is ready to be deployed, because there are some buggy parts in functionality as well as some stars don't have information to be displayed, but this has arisen due to the fact we spent more time implementing and resolving problems, than we thought we will.