Перенос проекта «AnimatedLogo» в Unity

Из всех файлов проекта (java), нам понадобятся только следующие – AniLogoRenderer, Apple, Graft, Grid, Leaf. Остальные файлы отвечают за запуск приложения под Андроид и стандартную заставку-яблоко для приложений, разрабатываемых на курсе Владислава Германовича Шубникова.

Коротко о файлах:

1) AniLogoRenderer – по сути, менеджер запуска проекта. Отвечает за то, в какое время что будет отрисовываться на экране.
2) Apple – класс, отвечающий за создание и отрисовку яблока
3) Graft – класс, отвечающий за создание и отрисовку черенка яблока
4) Grid – класс, отвечающий за создание и отрисовку сетки
5) Leaf – класс, отвечающий за создание и отрисовку листочка

Сценарий текущего приложения:

1) Появляется сетка зеленого цвета. Изначально она похожа на прямоугольную сетку, но постепенно превращается сначала в сферу, а потом в яблоко.
2) Какое-то время крутится сеточное яблоко.
3) Яблоко начинает приобретать заливку и становится уже сплошным. При этом включается освещение на один из его боков (получается блик).
4) Из яблока начинают вырастать черенок и листик.
5) Когда яблоко полностью создастся (черенок и листик окончательно вырастут), нижняя правая четверть его становится сеточной. Яблоко становится логотипом кафедры.

При этом практически все время объекты вращаются. В начале она высока, но постепенно скорость падает и яблоко останавливается, когда оно становится логотипом (четверть его становится сеточной). Весь сценарий зациклен и постоянно повторяется.

О реализации в Unity

Нам понадобится **объект Яблоко.** В него будет входить само яблоко, его черенок и листочек.

В файлах Apple, Grift и Leaf можно найти все формулы задания вершинного буфера, нормалей и индексов (для формирования треугольник) для

соответствующих частей яблока. Таким образом, при создании одного объекта можно просто объединить массивы каждого типа в один и отрисовывать.

В Unity по сути можно создать любые игровые объекты, похожие на нужные нам (например, сферу вместо яблока). В скрипте, повешенном на объект можно переопределять его mesh примерно следующим образом:

```
private float size = 2.5f;

    void Update()
    {
        Mesh mesh = transform.GetComponent<MeshFilter>().mesh;

        mesh.triangles = new int[] { 0, 1, 2, 0, 2, 3 };
        mesh.vertices = new Vector3[] { new Vector3(-size, -size, 0.01f), new
Vector3(size, -size, 0.01f), new Vector3(size, size, 0.01f), new Vector3(-size, size,
0.01f) };
        mesh.uv = new Vector2[] { new Vector2(0, 0), new Vector2(0, 1), new Vector2(1,
1), new Vector2(1, 0) };
        mesh.RecalculateNormals();
    }
```

Еще на эту тему:

http://answers.unity3d.com/questions/36480/how-do-you-make-mesh-in-unity.html

http://answers.unity3d.com/questions/459786/can-you-generate-a-mesh-at-runtime.html

http://answers.unity3d.com/questions/139808/creating-a-plane-mesh-directly-from-code.html

В проекте Надежды Кацман **сетка** – это отдельный объект. В начале появляется именно сетка, а потом по сути в том же месте рисуется яблоко и пропадает. Здесь я вижу для нас два решения – или оставить также и сделать отдельный объект, или совместить сетку с яблоком и в зависимости от времени использовать разные скрипты для отрисовки. Кажется, что все-таки проще сделать сетку тем же самым объектом и использовать просто разные скрипты.

Также нам нужен будет на сцене **точечный источник света** и **скрипт** для него, позволяющий включать и выключать его по мере надобности.

Нужен **скрипт-менеджер**, который будет осуществлять сценарий отрисовки и эффектов. Т.к. объект будет один, то скрипт будет на нем и висеть, а по основным событиям – Start, Update и т.д. просто вызывать функции из разных скриптов. Вызов функций одного скрипта из другого выглядит примерно следующим образом:

```
myObject.GetComponent<MyScript>().MyFunction();
```

Подробнее:

https://forum.unity3d.com/threads/calling-function-from-other-scripts-c.57072/

Вызовы скриптов должны быть связаны со временем. Для подсчета времени можно использоваться встроенные возможности Unity:

http://answers.unity3d.com/questions/64498/time-counter-up.html

Также в скрипте должно быть реализовано постоянное вращение объекта в функции Update, а скорость вращения должна зависеть от времени (формулы зависимости также есть в коде Надежды в файле AniLogoRenderer).

Внешними аргументами скрипта должны быть: время продолжительности одного цикла анимации, а также коэффициенты, позволяющие определить в какой момент должен меняться эффект. Количество коэффициентов и их значения будут зависеть от сценария эффектов. Также сейчас у Надежды отдельно задается время в секундах, когда логотип висит неподвижно после анимации. Можно или сделать его отдельным параметров, либо задать коэффициентом как часть общего цикла анимации.

Также скрипт-менеджер должен иметь возможность воздействовать на точечный источник света (скорее всего, через скрипт источника света).

Альтернативный вариант: создать для яблока, черенка и листика разные объекты. Повесить скрипт-менеджер на какой-то один объект, передавать ему в качестве аргументов другие объекты и воздействовать на них внутри скрипта. По сути, оба варианта могут быть реализованы – лучше обсудить, какой из способов лучше. Часть объектов будут некоторое время вообще не отрисовываться.

Аргументы для остальных скриптов (не менеджера):

1) Яблоко
   nSections – количество долек (должно быть кратно 2 * sectionRatio)
   sectionRatio – количество долек в проволочной модели будет nSections / sectionRatio
   nSegments – количество сегментов в линии (должно быть кратно 3)
   segmentRatio – количество сегментов в проволочной модели будет nSegments / segmentRatio
   rApple – параметр яблока
   color – цвет яблока
   specular – отражающая способность
   shininess – яркость яблока
   lineWidth – толщина линии

2) Черенок
   nSections – количество долек (должно быть кратно 2 * sectionRatio)
   sectionRatio – количество долек в проволочной модели будет nSectionsions/sectionRatio
   nSegments – количество сегментов в линии (должно быть кратно 3)
   segmentRatio – количество сегментов в проволочной модели будет nSegmentsments / segmentRatio
   color - цвет
   specular – отражающая способность
   shininess – яркость черенка
   lineWidth – толщина линии
3) Лист
   nSections – количество долек
   nSegments – количество сегментов в линии
   width – ширина половинки листа
   length – длина листа
   k – коэффициент при изгибе листа
   color – цвет
   specular – отражающая способность
   shininess - яркость
   lineWidth – толщина линии
4) Сетка
   nSections – количество долек
   nSegments – количество сегментов в линии
   len – длина линии в сетке
   rApple – параметр яблока
   color – цвет
   lineWidth – толщина линии

Значения всех этих параметров можно найти в коде.

Собранный проект для запуска в Android Studio выложен в облако: https://cloud.mail.ru/public/CRih/eJjYTYrc5