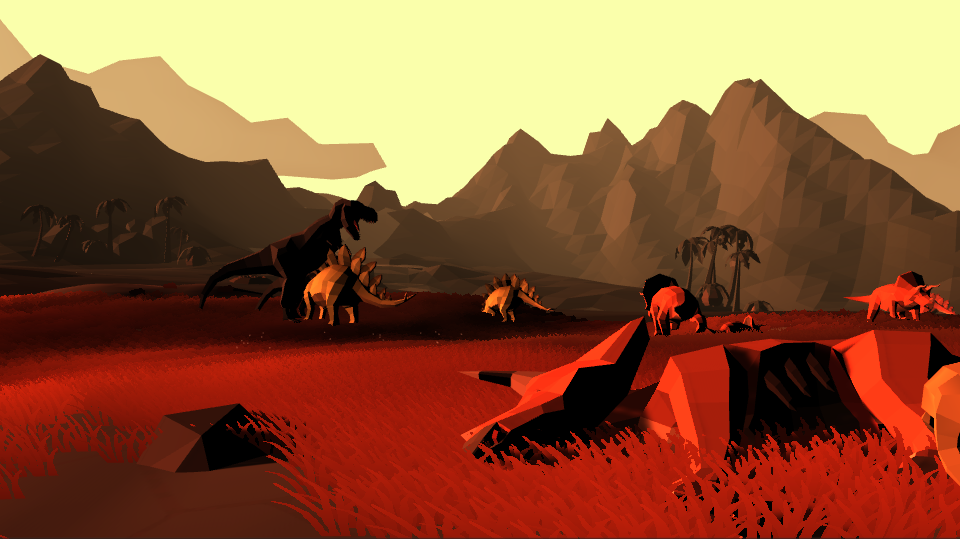
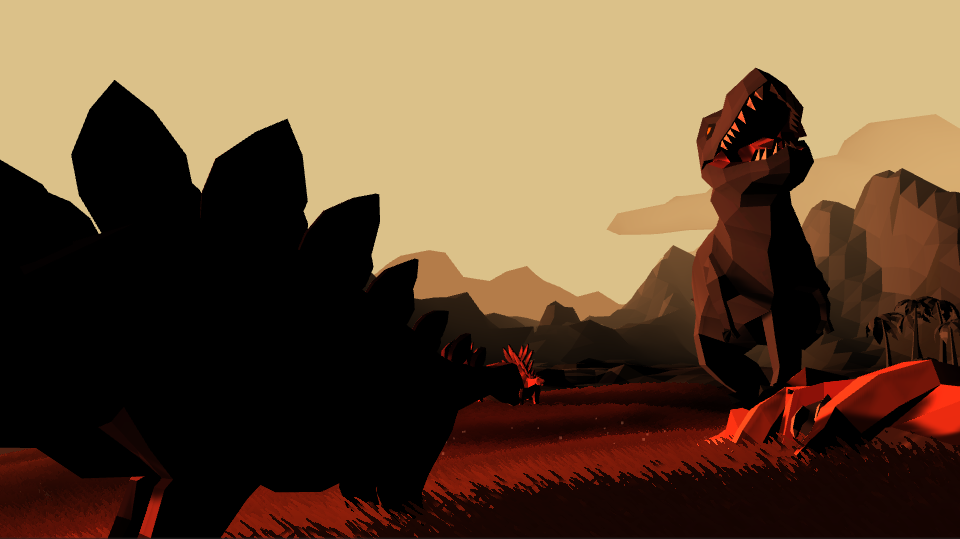
# 案例制作《Dino Park》

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 修改日期 | 修改人员 | 修改内容 |
| 0.50 | 8/11/2019 | 刘刚 |  |
|  |  |  |  |
|  |  |  |  |

## 设计思路



### 开发概况



游戏主题：恐龙公园（Dino Park）

开发工具：Improbable公司的SpatialOS

开发时间：3周（第一期，版本号：0.5）

#### 学习SpatialOS体会

1. 初期，因为A)要学习的内容较多，且B)系统较为开放（不同的例程实现相同功能的方法均不一样），而且C)设计思路与以前C/S（客户端/服务器）架构不一样，不太习惯，所以进展速度较慢。前两周，都是在对最基础的功能进行了解和实现，进度较慢；游戏中所有主要的AI和逻辑实现，都是第三周完成的。
2. 熟悉和了解ECS架构。这种架构是一种【面向数据】编程的思路，而不是之前的【面向对象】编程的思路。改变编程习惯，了解ECS的编程习惯，需要花费一定的时间。ECS就是实体(Entity)、组建（Component）和系统（System）。组件就是【数据】。SpetialOS的“批量自动生成代码”功能，批量生成的就是为实现这种制作模式所需要的代码。也就是说，从【schema】的定义（定义component），到通过Reader和Writer对数据进行写入、发布和读取。
3. 用ECS的方式来实现【有限自动机（Finite State Machine）】。这里我参考了《巫师（Wizard）》例程里的实现方法。首先定义一个“状态机”的组件，然后针对这个组件，编写不同状态下的逻辑。感觉两者可以很好地结合。
4. 结论：正如高洪泉所说，这个SpatialOS工具，目前还处于早期阶段，是否能够承受大型游戏的负载，尚无法证实。但是，适合做一些游戏的原型的设计和开发，可以用较快的时间，实现游戏设计者的意图。
5. 未来规划，我会利用业余时间，通过增加更多的特性和规则，使其变成一个更像策略，模拟类的游戏。大家如果有什么想法，欢迎随时告诉我。我也会把我的一些想法跟大家分享。over

#### 规则概述

1. 分为树木（Tree），腕龙（Brachiosaurus）和霸王龙（T-Rex）三类实体；
2. 他们的所有AI与变化均由【有限状态机】实现；
3. 恐龙实现：躲避天敌；追杀猎物；寻找植物；战斗；啃食（肉食和植物）；无目的漫游；静止；死亡；销毁等状态；
4. 植物实现：树木被啃食后变为木桩，一段时间后可以再生恢复原状；
5. 食物链：霸王龙-》腕龙-》树木；
6. 繁殖：恐龙可以下蛋，蛋变成幼龙，幼龙经过一段时间张大为成年恐龙。

## 树木

分为【健康（Healthy）】、【木桩（Stump）】、【燃烧（Flammable）】

燃烧功能目前尚未制作。

#### 树的初始化Snapshot

###### SnapshotDefault.cs

SnapshotUtil.AddTrees(snapshot, texture, 0.35f, SimulationSettings.AttemptedTreeCount, SimulationSettings.SpawningWorldEdgeLength, SimulationSettings.TreeJitter);

###### EntityTemplateFactory.cs

public static EntityTemplate CreateTreeTemplate(Vector3f position, uint initialRotation)  
{  
 var serverAttribute = UnityGameLogicConnector.**WorkerType**;  
   
 var template = new EntityTemplate();  
 template.AddComponent(new Position.Snapshot(new Coordinates(position.X, position.Y, position.Z)), serverAttribute);  
 template.AddComponent(new Metadata.Snapshot(SimulationSettings.TreePrefabName), serverAttribute);  
 template.AddComponent(new Persistence.Snapshot(), serverAttribute);  
 template.AddComponent(new TransformComponent.Snapshot(initialRotation), serverAttribute);  
   
 template.AddComponent(new Harvestable.Snapshot(), serverAttribute);  
 template.AddComponent(new Health.Snapshot(SimulationSettings.TreeMaxHealth, SimulationSettings.TreeMaxHealth, true), serverAttribute);  
 template.AddComponent(new Flammable.Snapshot(false, true, FireEffectType.**BIG**), serverAttribute);  
 var treeStateComponent = new Dinopark.Plants.TreeState.Snapshot((TreeType) Random.Range(0, 2), TreeFSMState.**HEALTHY**);  
 template.AddComponent(treeStateComponent, serverAttribute);  
   
 template.SetReadAccess(UnityClientConnector.**WorkerType**, UnityGameLogicConnector.**WorkerType**, MobileClientWorkerConnector.**WorkerType**);  
 template.SetComponentWriteAccess(EntityAcl.**ComponentId**, serverAttribute);  
   
 return template;  
}

以下是树的属性，都用【**组件**】来实现：

##### 标准组件

Position,Metadata,Persistence

##### 新增组件

###### TransformComponent

schema\dino\_park\core\Transform.schema

上面的 Position 组件只有坐标的传递，这里是旋转角度的传递。

###### Harvestable

schema\dino\_park\plants\Harvestable.schema

是否可以被砍伐（可以被吃）。

###### Health

schema\dino\_park\life\Health.schema

生命值组件。这个组件被广泛应用于所有的实体。

###### TreeState

schema\dino\_park\plants\trees.schema

树的种类管理。

##### 客户端创建GameObject

###### EntityGameObjectCreator.cs

var isTree = metadata.EntityType == SimulationSettings.TreePrefabName;// 树

if (isTree)  
{  
 var pathPrefab = $"**Prefabs/{\_WorkerType}/**" + SimulationSettings.TreePrefabName;  
 var prefab = Resources.Load(pathPrefab);  
 var entityGameObject = UnityEngine.Object.Instantiate(prefab);  
 entityGameObject.name = SimulationSettings.TreePrefabName + "(ID:" + entity.SpatialOSEntityId + ", Worker: " + \_WorkerType + ")";  
 linker.LinkGameObjectToSpatialOSEntity(entity.SpatialOSEntityId, (GameObject)entityGameObject);  
 //Debug.Log("EntityGameObjectCreator OnEntityCreated - A tree GameObject created");  
}

路径里必须事先保存好对应的预制件。

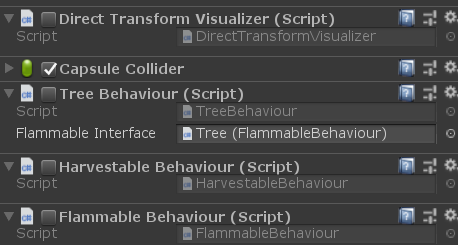
#### 脚本挂接

***注：一般服务器主逻辑我们用【{组件名称}Behaviour.cs】，客户端主显示我们用【{组件名称}Visualizer】来实现。***

***例如：TreeBehavour.cs, TreeVisualizer.cs***

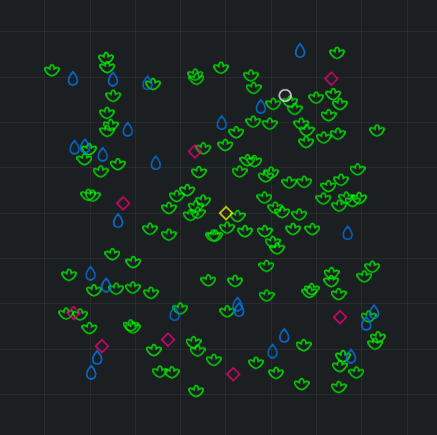
##### 服务器

###### 预制件：Resourcs/Prefabs/UnityGameLogic/Tree.prefab



可以把树的渲染组件（MeshRenderer）去掉，因为服务器端不会显示。

###### DirectTransformBehaviour.cs



接受坐标信息，有了这个脚本，在【Unity/SpatialOS/Open Inspector】菜单打开的页面里才能看到实体坐标的移动。

[Require] private PositionReader positionComponent;  
[Require] private TransformComponentReader transformComponent;

###### TreeBehaviour.cs

private static Dictionary<long, TreeBehaviour> allTrees = new Dictionary<long, TreeBehaviour>();

public static Dictionary<long, TreeBehaviour> AllTrees { get { return allTrees; } }

[Require] private **TreeStateWriter** tree;

[Require] private FlammableWriter flammable;

[Require] private **HealthWriter** health;

[Require] public EntityId \_entityId;

[Require] private **HarvestableCommandSender** cmdSender;

private long \_id;

private **HarvestableBehaviour** \_harvestable;

[SerializeField] private FlammableBehaviour flammableInterface;

private **TreeStateMachine** stateMachine;

* TreeState,树的种类
* Health,树的生命
* HavestableBehaviour,是否可以被砍伐的属性
* HarvestableCommandSenter，发送砍伐（Harvest）命令
* TreeStateMachine，树的状态管理

###### HarvestableBehaviour.cs

[Require] private HealthWriter health;

[Require] private HarvestableCommandReceiver cmdReceiver;

这里有一个CommandReceiver，用于接收Command操作。

private void OnHarvest(Harvestable.Harvest.ReceivedRequest request)  
{  
 var payload = request.Payload;  
 var **nowHealth** = health.Data.CurrentHealth;  
 float **resTaken** = 0;  
 if (**nowHealth** > payload.ResourcesNeed)  
 {  
 **resTaken** = payload.ResourcesNeed;  
 **nowHealth** -= payload.ResourcesNeed;  
 }  
 else  
 {  
 **resTaken** = **nowHealth**;  
 **nowHealth** = 0;  
 }  
  
 var update = new HarvestResponse()  
 {  
 ResourcesTaken = **resTaken** };  
 cmdReceiver.SendHarvestResponse(request.RequestId, update);  
 //Debug.Log("OnHarvest ResourceNeed<"+payload.ResourcesNeed+"> Harvester<"+payload.Harvester.Id+">");  
  
 var update2 = new Health.Update  
 {  
 CurrentHealth = **nowHealth** };  
 health.SendUpdate(update2);  
}

##### 客户端

###### Resourcs/Prefabs/UnityClient/Tree.prefab

###### HealthVisualizer.cs

###### TreeStateVisualizer.cs

##### 属性配置

配置文件在：SimulationSettings.cs，搜索“Tree”

可以配置初始值。

##### 属性发布

##### 属性接收和显示

## 恐龙

分为

IDLE = 0;

EAT = 1;

WANDER = 2;

RUN\_AWAY = 3;

CHASE = 4;

ATTACK = 5;

DEAD = 7;

ROAR = 8;

HATCH = 9;

LOOK\_FOR\_FOOD = 10;

EAT\_FOOD = 11;

VANISH = 18;

ON\_FIRE = 19;

一共13个状态，其中ROAR和ON\_FIRE尚未制作。

###### 腕龙（食草）

###### 霸王龙（食肉）

## 食物链

## 繁殖

