Growth Game

Introduction

Growth Game is a game jam project. The player is the commander of a village, and can **upgrade buildings**, **research technology,** and **send out military units** to try to destroy the enemy.

The player is trying to get **gold**, which can be obtained at the start of the game, by mining, by **sending out units**, and **killing units.**

The player can also gain **wood, food,** and **ore**, by upgrading infrastructure.

When the outgoing unit leaves the town, the player will get the unit's **labor bonus**.

Resources

Resources in the game are as follow:

- **Gold**: Used to buy expensive units and upgrade buildings.
    - Can be mined with ore at a *very* low rate.
    - Can be gained from killing units.
    - Can be gained from leaving the town with units.
- **Food:** Required to buy bio units
- **Ore:** Used to upgrade buildings and usually to buy armored units.
- **Wood:** Used to build houses and usually to buy ranged units.
- **Supply:** Used to cap unit space, can be upgraded by building more houses. There is a supply cap of X.

Resources are kept in the Controller.

Outgoing Units

Outgoing Units will have the following information:

```
FName UnitID
FText UnitName
FText UnitDescription

int32 SupplyCost
        ///    i guess this could be an unordered map
TArray<FStruct ResourcePair>        //  What resources are needed to train this unit
        {ResourceType ResourceType, float Amount}
float TrainingTime                  //  How long it takes to train this unit

class Unit                          //  Unit to spawn

//  to be passed to the unit

float CurrentHP
float MaximumHP

enum UnitType
        {Unarmored, Armored, Piercing}
enum WeaponType
        {Melee, Ranged}

float[4] AttackRange
float[4] AttackDamage
float[4] AttackPenetration
float[4] Armor
float[4] AttackSpeed

enum Faction                        EXPOSEONSPAWN
        {Friendly, Enemy}

int32 GoldBounty                    //  Give this much gold to the KILLER when killed, show
                                    //  icon + gold gain
enum ResourceType
        {Gold, Ore, Wood}
//  when unit passes by, show icon + tax gain
int32 UnitTax                       //  Give this much to the OWNER when leaves the town
```

Controller

Controller is the parent class of the PlayerController and the AIController

Purchasing Anything

```
bCheckIfCanBuy(){
	for (auto Resource : ResourcePair)
	{
		if (Resource.Amount > Controller.Resource) return false;
	}
	return true;
}

bHandleTransaction
	for (auto Resource : ResourcePair)
	{
		Controller.Resource - Resource.Amount
	}
```

ResourcesComponent

```
//  Resources category

private:

int32 CurrentSupply
int32 MaximumSupply

//  Resource collection ticks at 0.1s

float Wood;
float getWood();
void deltaWood(float DeltaNum);
```

```
float WoodCollectionRate;
float getWoodCollectionRate();
void deltaWoodCollectionRate(float DeltaNum);

float Food;
float FoodCollectionRate;

float Gold;
float GoldCollectionRate;

float Ore;
float OreCollectionRate;
```

Player Pawn

The player pawn is merely a representation of the camera work.

A/D	Pans the camera left and right (moves the pawn)
W	Zooms the camera in (clamped)
S	Zooms the camera out

UpgradesComponent

UpgradeComponent will store all of these values.

Combat upgrade
// all of these are int32, default at 0

Melee Weapon	Damage
Melee Weapon	Armor
Melee Weapon	Penetration
Melee Weapon	Attack Speed

Ranged	Damage
Ranged	Range
Ranged	Penetration
Ranged	Attack Speed

Tech upgrade

Melee Light	4
Melee Armored	4
Melee Piercing	4

Ranged Light	4
Ranged Piercing	4

Hall of Heroes

Healer	Heal over time
Rallyer	More damage
Sentinel	More armor
Captain	More penetration

Mage	AoE attack
Rogue	Poison Attack ranged
Pyromancer	Sunfire

Combat

FinalDamage = IncomingDamage * (100 / 100 + TargetArmor - InstigatorArmorPenetration)