a acheter:

Dungeon Generator.

Pathfinding Dynamic.

UI.

***Attributes System***

**Il serait bien de précéder un maximum de nos classes de sealed -> dropbox opti. et portfolio final.**

**Alarm -> Timer et timer -> alarm puis renvoyer sur dropbox, pastebin, gist.**

**Problématiques à résoudre :**

* Code commun entre les attributs du joueur et ceux des ennemies. : getAttribute, stringAttribute.
* Code s’adaptant rapidement à nos futurs systèmes : succès, shrine.
* Code pouvant être sérializer.
* Code éditable : attribute editor.
* Les classe d’attributs permettent d’éditer chaque calcul de valeur par un système d’évênements.
* Gestion de charactéristiques, de résistances, de dommages sans devoir modifier l’architecture et elle doit marcher pour mon ennemi.

**Solutions :**

* classe template.
* abstraction.
* comportement virtuelle commun.
* virtual void GetCalculValue();
* hériter de la classe EntityAttributes();

playerAttribute devient entityAttributes.

If (GetAttrib(attribName) == null)

Debug.LogError()

**Refonte des attributs :**

Class PlayerAttributes

{

Public Attribute atStart ;

Public Attribute bonus ;

Public Attribute current ;

Public PlayerAttributes percent = NULL ;

Public Initialize(float bonus , float atStart, bool adddefaultcallback, PlayerAttributes percent = NULL, string attributeName)

{

Percent = percent ;

Current = new Attribute() ;

AtStart = new Attribute() ;

Bonus = new Attribute () ;

If (adddefaultcallback)

AddupdateCallback(Defaultupdatecallback) ;

Current.Initialize(0.0f, AttributeName) ;

Bonus.Initialize(bonus,, AttributeName) ;

AtStart .Initialize(atStart,, AttributeName) ;

}

Public AddupdateCallback (action updatecallback)

{

atStart .subscribe(updatecallback ) ;

bonus .subscribe(updatecallback ) ;

}

Public AddupdateCallbackthenupdate(action updatecallback)

{

AddupdateCallback(updatecallback) ;

atStart.CallAction() ;

bonus.CallAction() ;

}

Public Void Defaultupdatecallback()

{

Current.Value = getdefaultupdatevaluewithpercent() ;

}

Public float getdefaultupdatevaluewithoutpercent()

{

Return atStart.value + bonus.value ;

}

Public float getdefaultupdatevaluewithpercent()

{

Return getdefaultupdatevaluewithoutpercent() \* Getpercentvalue() ;

}

Public float Getpercentvalue()

{

Return NULL == percent ?

1.0f :

percent. Value \* PERCENT\_TO\_UNIT ;

}

}

**Refontes des catégories d’attributs :**

* Ne plus utiliser de double tableau car pour la serialization et pour faire le liens avec les attributs des shrines ou des objets mon système n’est pas pratique du tout. Car il faut 2 énumérations dont une qui change tous le temps :

Class PlayerCategoriedAtrributes

{

Start ()

{

Int[] ids = objectcontainerhelper.createid(enumsubcategory.enumtoarrayofstring) ;

}

Getattributes(ecategory category, string atreibuteId)

{

Return [geindex(category)][objectcontainerhelper.getid(atreibuteId)] ;

}

Getresistances(string attributeid) // accès aux résistances et dommages ?

{

Return résistances[objectcontainerhelper.getid(atreibuteId )] ;

}

}

2) revoir toutes mes énumérations, en supprimer une bonne partie

3) Recoder tous nos attributs.

4) rajoute dans le service container des données de configuration et les utiliser.

- Il ne faut plus avoir plein de constante de caractéristiques .

5) Rajouter dans mon le code des attributs : successValue; characteristicValue, shrineValue;

***Succès***

enum ESuccessCategory

{

Kill,

Campaign,

General,

…

}

class SuccessManagerService

{

SuccessData[][] sucesses = new SuccessData[][];

sucesses[EnumHelper.GetIndex<ESucessCategory>(ESuccessCategory.Kill)] = new Success[]

{

new blablaSucessKill();

…

};

// pareil pour les autres catégories.

}

class SucessLevel

{

float minValue;

float maxValue;

float bonusAttributes;

}

class ASucessData

{

List<SucessLevel> levelsData;

Success currentLevel;

int levelIndex;

string description;

string backgroundImage; // couleur en fonction du niveau ?

string imageID;

void UpdateDescription();

void UpdateEffect();

}

**GUI :**

| Général | Campagne | Kill | : Ce sont des onglets |
| --- | --- | --- | --- |
| color background + image | Description |  | Cette colonne scroll bar |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

***Shrine System***

public enum EShrine

{

MovementSpeed,

AttackSpeed,

RegenerateSpeed,

Defence,

Damage,

CastSpeed,

GodModeRecharge,

SpawnItem,

SpawnBoss,

SpawnGobelin,

UpgradeNearestMonsterCategory,

…

}

class Shrine

{

const float distanceToEnableShrineEffect;

ShrineEffect effect;

Transform player;

Transform trans;

Timer waitTimer = new Timer(30.0f);

// roba pour chaque effet ?

void Start()

{

GenerateEffect();

}

void Update()

{

waitTimer.Update();

if (waitTimer.IsRinging() &&

Vector3.Distance(player.position, trans) < distanceToEnableShrineEffect)

Effect();

}

void GenerateEffect()

{

EShrine randomEffect = (EShrine)(Random.Range(0, EnumHelper.Count<EShrine>() +1);

switch (randomEffect)

{

case EShrine.MovementSpeed :

effect = new ShrineEffectMovementSpeed();

break;

…

}

material.color = effect.color;

waitTimer.Reset();

}

void Effect()

{

// If effect.category == bonusAttribute

player.GetComponent<ShrineEffectHolder>().AddShrineEffect(effect);

// else

effect.Cast();

DisplayName(effect.name);

GenerateEffect();

}

}

enum EShrineEffectCategory

{

Buff,

Other,

}

class AShrineEffect

{

//protected Color color;

protected EShrineEffectCategory category;

protected shrineName;

virtual void Effect() = 0;

//virtual void Uneffect() - 0;

}

class AShrineEffectBuff : AShrineEffect

{

protected List<BonusAttribute> attributes = new List<BonusAttribute>();

protected float duration = 30.0f;

override void Effect()

{

PlayerAttributes playerAttribute = ServiceContainer.Instance.GameState.playerAttributes;

playerAttributes.AddListOfBonusAttribute(this.attributes);

player.changeColor(color);

}

override void Uneffect()

{

PlayerAttributes playerAttribute = ServiceContainer.Instance.GameState.playerAttributes;

playerAttributes.RemoveListOfBonusAttribute(this.attributes);

player.changeColor(defaultPlayerColor);

}

}

class BonusAttribute

{

ECategoryAttribute category;

string attributeID;

float bonusValue;

}

***Skeleton Behaviour Tree***

**Résumé :**

If (distancefrominitialposition > maximumdistancefrominitialposition)

gotoonitialposition;

ResetLife;

Else if (distancefromplayer < aggroRange &&

distancefromplayer > attackrange)

GotoPlayer;

Else if (distancefromplayer < attackrange )

If (CanAttackPlayer)

Attackplayer

Else

Findwanderpoint

gotowanderpoint

**Sequences :**

* Wandersequence:
* GotoRandomPositionInWanderRangeAroundInitialposition
* Waitntime.
* Attaquesequence:
* GotoEnoughCloseFromPlayertoattackhim
* Attackplayer.

**Decorators :**

* IsEnoughNearfromplayertoattackhimsecorator
* CanWaitDecorator
* CanAttackPlayer

**Tasks :**

* AGototask
* GotoEnoughCloseFromPlayertoattackhim
* GotoRandomPositionInWanderRangeAroundInitialposition
* Waitntime
* Attackplayer

**Selectors :**

* TooFarFromPlayerOrContinueselector

To do :

- Ui :

- Life, mana, xp

- Caractéristiques

- PlayerAttributes :

- Player caractéristiques

- Player Attributes editor

- IA :

- Behaviour tree

- Object pool :

- Rajouter ma prefab de texte

- Attack System :

- Prefab de texte :

- Elle utilise lobject pool

- Elle a une animation et un animator controller

- Regarde toujours la caméra

- Un script possédant une méthode permettant de modifi

er le nombre

afficher et la couleur, rajouter un préfixe (critical,

frozen), choisir

l'animation, et le temps avant qu'elle disparaisse (ob

ject pool), la

vitesse pour monter.

- Outline l

’

ennemi lorsque l'on l'attaque,

***Monster Spawner***

**A savoir :**

* Le start de chaque monstre appelé une fonction d’initialization permettant en fonction de son niveau et de son type de monstres (Gobelin dérivant de AMonster par exemple, d’override les attributs tels que les dommages, les sorts donc le behaviour tree, la vie, la vitesse de course, etc…) de redéfinir ses attributs et son IA.
* Le spawner va juste permettent d’initialiser le type du monstres (Gobelin, Troll, Wolf) ainsi que sa category de puissance (Boss, Champion, etc..) et potentiellement des paramètres de patrouille d’IA.

class MonsterSpawner // gère un tableau de différent spawner de monstres.

{

[SerializeField]

MonsterToSpawn[] spawnerParameters;

ESpawnerAreaPattern pattern; // circle, rectangle.

bool spawnAreHomogeneous; // true : on va essayer de spawner les monstres de sortes qu’ils soit exactement à la même distance des autres monstres, on les spawn dans une position random.

Vector2 offset;

Vector2 areaSize;

}

[System.Serializable] // un type de spawner particulier.

class MonsterToSpawn

{

EMonster type; // Troll, Wolf, Gobelin.

float timeBeforeEachRespawn; // soit c’est le spawner qui gère ça soit le monstres lui même.

ERespawnOwner respawnOwner; : Spawner ou Monster, si c’est le monster alors après que le monstre soit mort et aura attendu TimeBeforeEachRespawn il respawnera. Si c’est Spawner si il y a par exemple 4 monstres qui sont mort, on devra attendre timeBeforeEachRespawn \* 4 pour qu’il ressuscite tous.

Alarm timerToRespawnMob;

}

***Minion***

**A savoir :**

* Les minions hérite du même Behaviour Tree que ceux des APet.
* Ils overrident ce Behaviour Tree en rajoutant la possibilité d’attaquer et de prendre des coups et donc la mort aussi.

***Waypoint***

[System.Serializable]

class Waypoint

{

[SerializeField]

bool isUnlock;

[SerializeField]

string name;

[SerializeField]

string description;

[SerializeField]

Transform transform;

}

class WaypointManager // penser au langage manager avant.

{

[SerializeField]

Waypoint[][] waypoints; // waypoints[0] -> acte 1, waypoints[3] -> acte IV.

int waypointIndexSelected = -1;

void Teleport(Waypoint waypoint)

{

transform.position = waypoint.position;

}

}

| **Acte I** | **Acte II** | **Acte III** | **[X]** |
| --- | --- | --- | --- |
| **Image** | Name et description |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

***Minimap***

**A savoir :**

* Voir la minimap de Blood Of Evil Unity et Unreal Engine.
* Utiliser le package pour avoir une minimap circulaire.
* Utiliser le package fog of war.
* Zoomer et dézoomer avec des facteurs minimum et maximum.
* Tourner à 90° à gauche puis à droite.
* Ne voit que ce qui est sur son layer minimap.

**Ce qu’elle voit :**

* **Puce de couleur :**
  + **rouge :** enemi de faible catégorie, si c’est un boss l’image sera une tête de mort et si gobelin : image de gobelin.
  + **Vert :** joueur.
  + **bleu clair :** minion.
  + **Jaune :** PNJ avec une quête principale.
  + **Orange :** objet légendaire.
  + **Violet :** quête principale.
* **Images spécifique :**
  + Coffre, forgeron, portail pour se téléporte ou changer de zone :
    - **portail orange :** dungeon.
    - **portail rouge :** portail de bosse ou changer d’acte.
  + **Tous ce qui est relatif au métiers de récoltes :** céréales, bois, minerais, bois.

**Minimap Fade :**

* voir le pade in dans notre caméra vue de haut.

***Territory***

**A savoir :**

* C’est un arbre de connection de scène.

class TerritoryContainer : MonoBehaviour

{

Dictionary<string sceneName, List<TerritoryNode>> territories;

List<TerritoryNode> GetConnections(TerritoryNode node)

{

return territories[GetSceneName(node)];

}

string GetSceneName(TerritoryNode node)

{

return territories.GetKey(node);

}

Teleport(TerritoryNode node)

{

if (Application.loadScene != GetSceneName(node)

Application.LoadScene(node.sceneDirection);

player.transform.position = node.transform.position;

}

}

classe TerritoryNode

{

string sceneDirection;

Transform transform;

}

***Die System***

**A savoir :**

* Regarder mon livre sur Blood Of Evil.
* Animation de mort ou ragdoll ?
* GameState.player = null;
* GameState.player.canMove = GameState.player.canBeHit = false;
* Afficher un timer au dessus du joueur.
* Quand le timer est terminé, affiché les options pour revivre, voir le livre.

***GUI Manager***

**Manager de fenêtres :**

* If one window is open, can’t move or attack.
* Escape : close all windows.

**Chaque fenêtres :**

* Bouton pour quitter.
* Draggable Window.
* Cross resolution.
* CallBack Open.
* Callback Close.
* Cross Resolution : voir le package UI.
* Key To Open -> communication avec mon système de configuration de touches.

Bullet System :

ABulletSystem

Destroy after n time.

string[] targetTag; // player ou enemy ou minion etc..

vector3 velocity;

if collision with something different than targetTag : velocity = 0.

collision withTarget callback + destroy.

BulletDamage

DamageElement[] damages;

***Comment attaquer ?***

1. Instancier une boîte de collision et voir si elle collision avec d’autres entités du bon tag.
2. Pathfinding dynamic ?
3. Distance + orientation ?

**Clique gauche sur un ennemi :**

* Clear Tasks.
* Set Target.
* Go enough near to enemy to attack him.
* attack while not die.
* animEvent à la fin de l’animation d’attaque :
  + if (target != null ou onHoverEnemy)
  + calculDamageElement with elemResistance and criticalDamage;
  + penser à activer les effets de chaque éléments.

***Pet System***

**En plus :**

* tag : pet.
* Animal move.
* minimap : point bleu clair.

class APet

{

Transform trans;

Transform target;

Vector3 destination;

float distanceFromTarget;

List<BonusAttribute> bonusAttributes;

float moveAreaRadius; // déplacement dans une zone circulaire autour du joueur.

float followDistance; // si la distance entre la target et le pet est > followDistance alors suivre le joueur.

float areaOfReaseach;

float distanceFromDestination;

float distanceFromTarget;

float distanceToTPToTarget;

void Update() // modifier tous ça en BehaviourTree

{

distanceFromTarget = Vector.Distance(trans.position, target.position);

if (distanceFromTarget > distanceToTPTarget)

trans.position = target.position;

else if (distanceFromTarget > followDistance)

navMeshAgent.SetDestination(target.position);

else

{

distanceFromDestination = Vector3.Distance(trans.position, navMeshAgent.destination);

if (distanceFromDestination < 0.4f)

ResetDestination();

navMeshAgent.SetDestination(destination);

}

}

void ResetDestination()

{

destination = GetRandomPointAroundTarget(areaOfReaseach, target);

}

void Effect()

{

player.GetCompnent<EntityAttribute>().AddAttributes(bonusAttributes);

}

void Uneffect()

{

player.GetCompnent<EntityAttribute>().RemoveAttributes(bonusAttributes);

}

}

class PetCat : APet

{

bonusAttributes = new BonusAttributes[]

{

new BonusAttribute(GoldFind, 100),

new BonusAttribute(RunSpeed, 20),

};

}