Projet R

16/06/2022

—

SITAL Mathéo / BEULQUE Théo
Douglas CORPS
Num étudiant (SITAL Mathéo): X XXX XXX XX
Num étudiant (BEULQUE Théo): X XXX XXX XX
Jury: Monsieur Bénameur
Soutenance: 27/06/2022
Diplome: LP DAWM

Remerciements

Nous tenons à exprimer toute ma reconnaissance à mon directeur de mémoire, Monsieur Bénameur. Je le remercie de m’avoir encadré.

J’adresse mes sincères remerciements à tous les professeurs, intervenants et toutes les personnes qui par leurs paroles, leurs écrits, leurs conseils et leurs critiques ont guidé mes réflexions et ont accepté de me rencontrer et de répondre à mes questions durant mes recherches.

Je remercie mes très chers parents, qui ont toujours été là pour moi.

Je remercie aussi Luna qui m'a grandement aidé pour traduire les textes de mon site internet et aussi de son soutien tout le long du stage et je la remercie grandement.

Je remercie Théo S, Thomas Lys pour avoir été les testeurs de mon jeu.

Je remercie énormément Isaïa pour le design pour la plupart des textures de notre projet !

À tous ces intervenants, je présente mes remerciements, mon respect et ma gratitude.

Merci -

Vue d'ensemble

Introduction

1. présentation du projet
2. Présentation de l'équipe
3. La méthode de travail
4. Inspiration du projet

Développement

1. Les idées conceptuelles
2. Les mécaniques principales
    a. Le menu
    b. Le personnage
        i. Compétence
        ii. Statistique
        iii. Arme
    c. Shop
    d. Les objectifs
    e. Les objets
3. Les maps

Conclusion

1. la suite de projet R

Introduction

1. présentation du projet

Théo Beulque et moi allons vous présenter notre projet qui se nomme Projet R (On n'a pas encore trouvé de nom pour notre jeu vidéo alors on l'appellera R pour Rogue like).

Le but de notre projet était de réaliser un roguelike difficile mais accessible à tous les joueurs.

Avant de rentrer dans les détails, nous allons vous expliquer le terme "Rogue-like".

Le Rogue-like est un sous genre de jeu vidéo de rôle, dans lequel le joueur explore un donjon infesté de monstres qu'il doit combattre pour gagner.

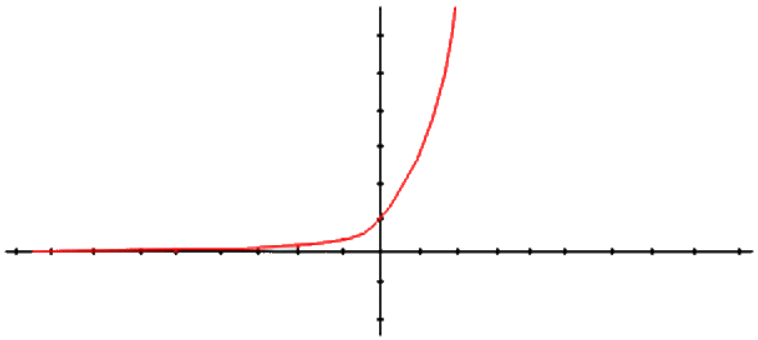
Comme je le disais plus haut, notre objectif est de faire un jeu difficile mais accessible à tous, ce que l'on recherche est qu'il y ait peu de joueurs (environ 0.1 ~ 1%) qui finissent le jeu en une partie et le reste des joueurs en une vingtaine de parties.

Pour les prochaines fois dans le texte j'utiliserai le mot *run* à la place de *partie.*

A force de recommencer la partie, le jeu sera de plus en plus simple car le joueur connaîtra les attaques du boss ou encore comment fonctionnent les salles et pour finir le joueur va cumuler les statistiques de son personnage à force d'en acheter (elle ne disparaît pas après sa mort définitive). Ici on recherche une croissance exponentielle .

(image qui montre la difficulté du jeu et le nombre run image non contractuelle)

Ainsi, moi et T.Beulque cherchons à faire un jeu dynamique car le problème récurrent dans ce genre de jeux c'est la répétitivité. Le point négatif serait donc de reproduire un jeu dynamique qui change à chaque run lancer.



Introduction

2. Présentation de l'équipe

La composition de l'équipe pour réaliser Projet R est assez simple, nous sommes deux.

Théo Beulque a travaillé sur l'interface du jeu, les menus et aussi sur quelques implémentations dans le jeu. Il a aussi supervisé et donné ses idées pour rendre le jeu équilibré: il était à la fois game Développeur et en même temps game Design de Projet R. Il a également réalisé quelques textures pour le menu et le jeu.

Moi, j'ai travaillé pour le jeu sur les parties mécaniques et j'ai participé à l'implémentation de nouvelles idées en demandant à Théo si cela était réalisable dans le contexte d'un jeu équilibré. J'ai enfin réalisé quelques textures dans le jeu.

Introduction

3. Inspiration du projet

Nous allons vous expliquer pourquoi nous nous sommes inspirés de certains projets existant ou encore de certaines mécaniques de jeu de grands et de petits titres qui ont réalisé des rogues likes.

Le jeu numéro un est Enter the gungeon, ce jeu est vraiment le pilier de notre inspiration car il est dynamique, diversifié, original au sens du gameplay.

image dans le jeu Enter the Gungeon

Pour la partie design, on s'est inspiré du jeu Zelda/Omori pour le système de TileMap pour une vision 2D de haut. On trouve que le Rogue-like est plus destiné à un jeu de haut 2D plutôt qu'un jeu sur les côtés en 2D ou 3D.

image de The legend of zelda

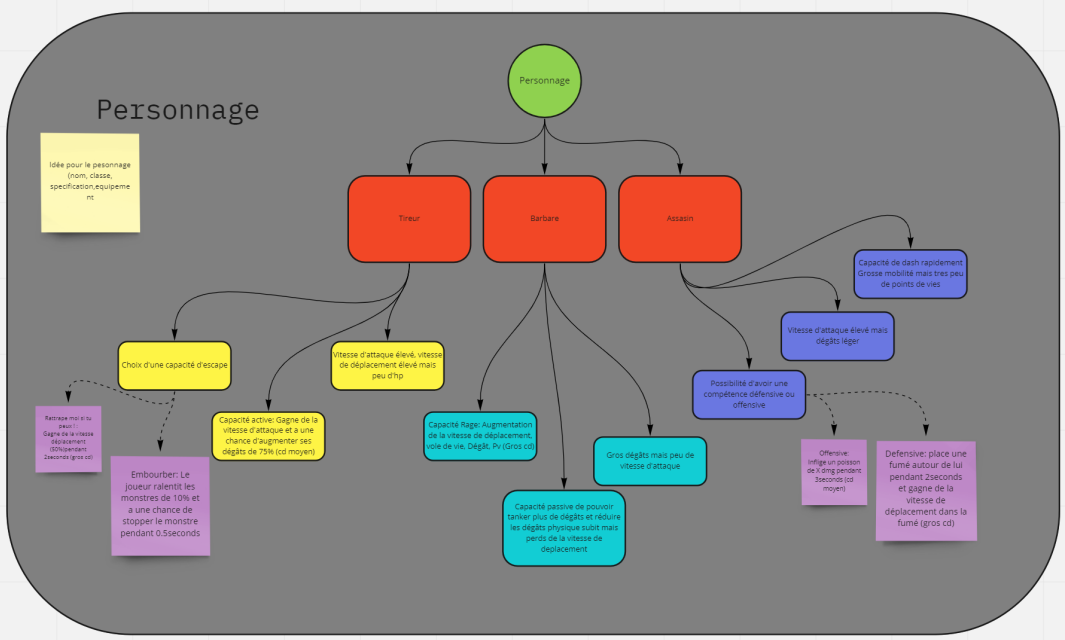
image d'omori

Développement

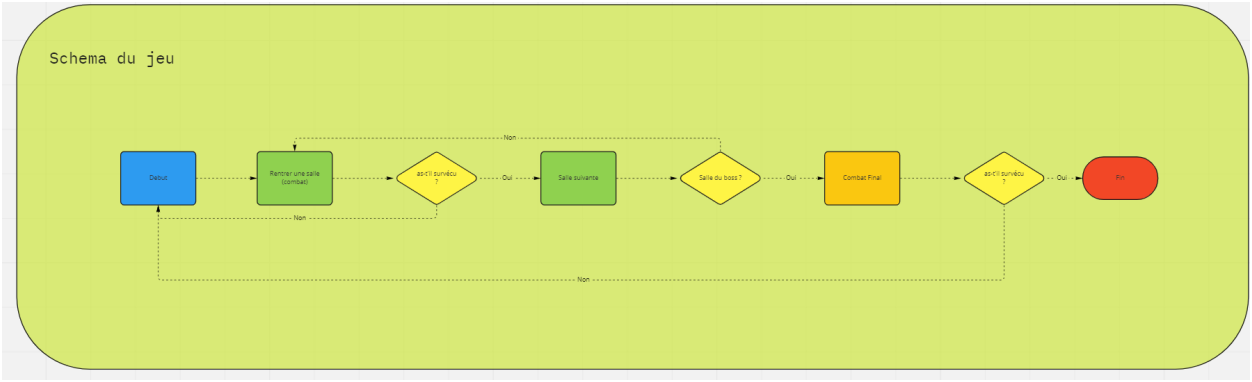
1. Les idées conceptuelles.

   Au début du projet, on était parti sur un personnage qui pouvait posséder plusieurs classes "Guerrier,Mage,Assassin..." et qu'il y aurait des salles aléatoires comme une salle avec *des trash mobs(monstres inutiles)* qui permet de gagner de l'argent pour pouvoir acheter des statistiques au personnage ou encore des compétences.

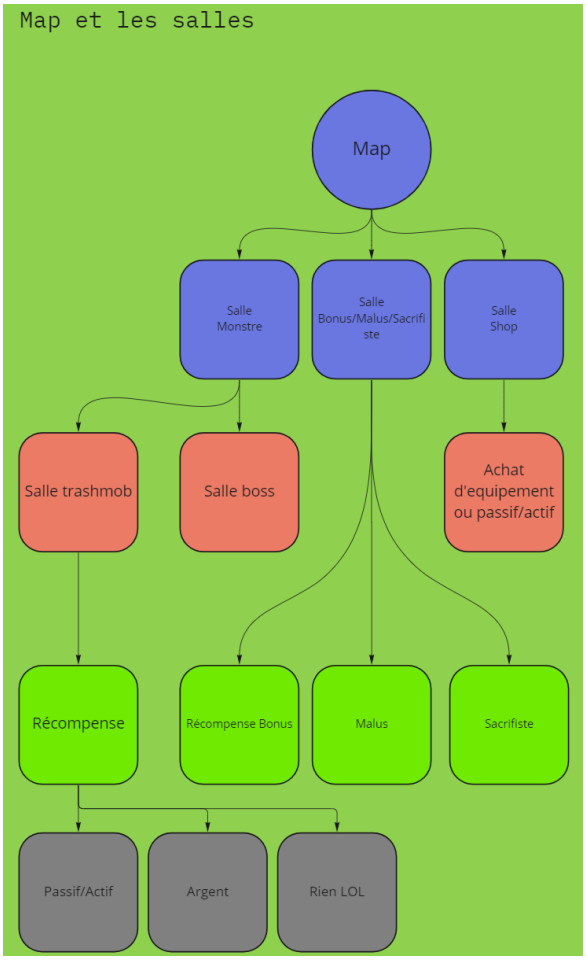
   Voici comment on a fonctionné pour travailler: on travaillait sur un tableau, on ajoutait des idées, ensuite on les a validées puis pour finir on a mis le statut de l'avancement du projet.



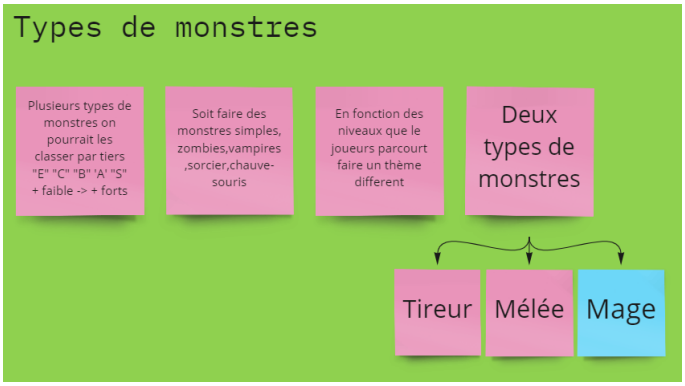
Ici création du personnage avec ses classes et ses compétences

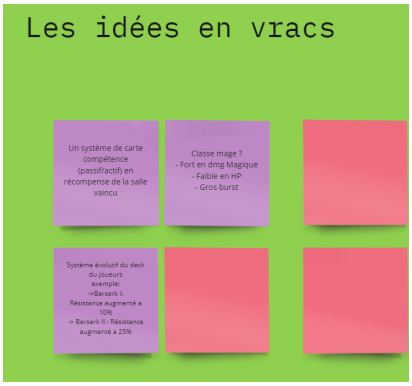


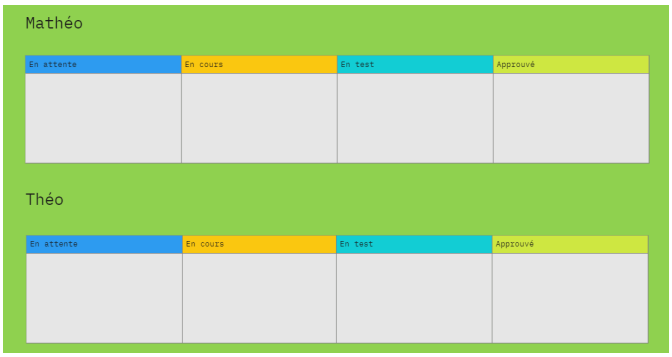
l'organigramme du fonctionnement du jeu - Projet R



le schéma de la map et la constitution des salles et des récompenses







c'est notre tableau virtuel, on met des posts-it avec nos idées sur le tableau virtuel et on les décale pour savoir si oui ou non on le fait ou encore si on a avancé sur la tâche ou encore qui doit le faire.

Notre deuxième outil principal utilisé pour communiquer, était le discord pour parler, faire des réunions ou mettre nos idées.

Notre troisième outil principal, est Github, il a été essentiel pour push et pull notre projet afin de pouvoir travailler sur le même projet sur des branches différentes sans abîmer le projet. Enfin, quand on avait fini notre tâche on amenait notre branche -Dev sur la branche -main et gardait alors un historique des versions que l'on a push.

Ici on voit les patch notes et les commits depuis le dépôt distant

ça permet de mettre un titre du push, on met Patch Version X.X puis en dessous on ajoute un commentaire, donc la plupart du temps on fait un résumé de ce que l'on a rajouté ou patché ou les modifications de certains scripts.

Développement

2. Les mécaniques principales

Dans cette partie, nous allons parler des mécaniques principales de projet R je vais essayer d'être synthétique car il y a beaucoup de choses à voir.

On va suivre cet ordre là :

- Menu
- Personnage
    - Compétence
    - Statistique
    - Arme
- Shop
- Les objectifs
- Les items
- Les pièges

Voilà, on a déterminé les axes de notre partie sur les mécaniques principales. Allons voir ce que cela donne pour notre projet de 100h.

Développement

2. Les Mécaniques principales | Menu

Dans notre projet comme dans tous les jeux, il y a un menu principal qui va permettre de modifier les aspects du jeu tels que la résolution ou encore le volume. Ici nous avons voulu faire quelque chose de simple avec 3 boutons...

Un deuxième menu est disponible directement en jeu pour permettre au joueur de mettre le jeu en pause mais aussi de choisir s' il veut quitter ou non le jeu.

Développement

2. Les Mécaniques principales | Personnage

Dans cette partie, on parlera du personnage qui est le centre majeur du *gameplay* (jeu). Le joueur va devoir utiliser le personnage pour se déplacer et attaquer les monstres et les balises pour avancer dans l'histoire.

Le personnage peut se déplacer sur les axes X / Y *(non Z car le jeu est en Deux dimensions)*

Pour pouvoir utiliser un personnage, j'utilise plusieurs choses un rigid body *(permet d'appliquer une force et une physique au personnage)* une Box Collider(permet d'avoir une boite de collision pour ne pas traverser les murs par exemple) et pour finir, j'ai ajouté un script que j'ai réalisé qui se nomme playerController.cs

Quand je vous présenterai le script je vais couper certaines informations ou ne pas parler des animations car lorsque le personnage bougera vers X ou Y,il jouera une animation même s'il reste en *Idle* (sur place).

PlayerController.cs:

Le Header("Component") : Déclaration de variable de type RigidBody2D.

Le Header("Stats"): Déclaration de variable de type float (Pour la vie/ vie maximum et la vitesse du personnage)

FixedUpdate() : Permet de s'exécuter ou non pendant une *frame* (une image)

Move(): Fonction qui permet de faire avancer le joueur vers X ou Y en choisissant la touche mappée dans le code *(Z S Q D)*

PlayerController.cs : (sans les animations/Raccourcie des variables Stats)

```
public class playerController : MonoBehaviour
{
    [Header("Component")]
    Rigidbody2D rb;
    Animator anim;

    [Header("Stats")]
    [SerializeField]
    public float moveSpeed;
    public float currentHealth;
    public float maxHealth;

    private void FixedUpdate()
    {
        Move();
    }

    void Start()
    {
        currentHealth = maxHealth;
        rb = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
    }
    void Move()
    {

        float x = Input.GetAxis("Horizontal") * moveSpeed * Time.deltaTime;
        float y = Input.GetAxis("Vertical") * moveSpeed * Time.deltaTime;

        //retourne toujours 1 ou -1
        rb.velocity = new Vector2(x, y);
        rb.velocity.Normalize();
    }
}
```

Le code complet est disponible sur github/zarkhein/rogue

Développement

2.Les Mécaniques principales | Compétence

Dans projet R le joueur possède un arbre de compétence ou encore *Skill Tree* l'arbre de compétence permet au joueur de trouver sa voix et s'orienter sur le *gameplay* (Style de jeu).

Si je prends par exemple un joueur plutôt orienté "Assassin" le joueur va s'orienter vers des compétences d'agilité/Rapidité avec un *burst* de dégâts (Beaucoup de dégâts sur un temps court).

Si le choix de l'assassin n'intéresse pas le joueur, il peut devenir Tireur ce qui permet de tirer plusieurs coups grâce à une compétence *Triple Shot* mais n'aura pas la mobilité d'un assassin par exemple.

Voici l'arbre de compétences en beta et skillTree.cs

```
public class SkillTree : MonoBehaviour
{
    public GameObject skillTree;
    public Button btn_mainSkill, btn_doubleshoot, btn_

    private void Start()
    {
        skillTree.SetActive(false);
        btn_mainSkill.onClick.AddListener(doubleShoot)
        btn_doubleshoot.onClick.AddListener(tripleShoo
        btn_dash.onClick.AddListener(dashSkill);
        btn_doubleshoot.interactable = false;
        btn_dash.interactable = false;
    }

    void tripleShoot()
    {
        Weapon.instance.skillLearned2 = true;
    }

    void doubleShoot()
    {
        Weapon.instance.skillLearned1 = true;
        btn_doubleshoot.interactable = true;
        btn_dash.interactable = true;
    }
```

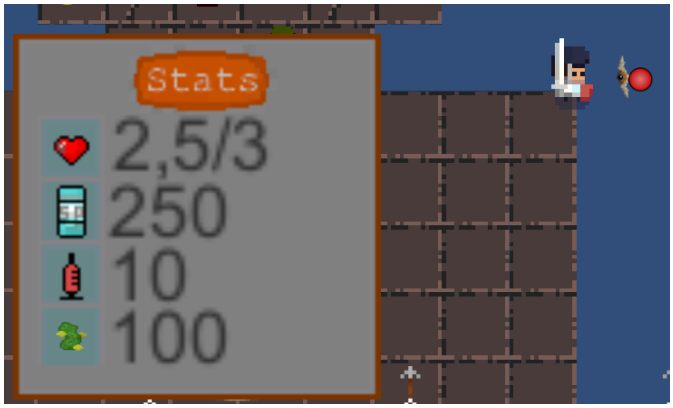
```
private void OnTriggerEnter2D(Collider2D collision)
{
    if(collision.gameObject.name == "Player")
    {
        skillTree.SetActive(true);
    }
}

private void OnTriggerExit2D(Collider2D collision)
{
    if (collision.gameObject.name == "Player")
    {
        skillTree.SetActive(false);
    }
}
```

Développement

2.Les Mécaniques principales | Statistique

Dans ce chapitre , on va développer la partie statistique du personnage. Le joueur va avoir 4 statistiques importantes: *La vitesse, La force, La distance, La vie.* Il possède une autre statistique moins importante *Attack Speed* (Vitesse d'attaque) ou encore Argent du joueur qui va lui permettre d'acheter des compétences ou des stats brutes via le *Shop.* Ce sont des variables totalement classiques qui peuvent être modifiable par le développeur (Ce sont des variables publiques et accessibles dans toutes les autres classes grâce à la fonction Awake() *[Permet de chercher le script avant le lancement même de la partie]*)

Un *GUI (Graphical user interface)* qui affiche les stats pour donner des informations au joueur.




```
[Header("Stat")]
[SerializeField]
public float moveSpeed;
public float currentHealth;
public float maxHealth;

[Header("Weapon")]
[SerializeField]
public float damage;
```

```
public GameObject playerInformation;
private bool information;
private float timeBtwOpen;
private float startTimeBtwOpen = 1f;

public Text lifeStats, speedStats, forceStats, rangeStats;
// Start is called before the first frame update
void Start()
{
    information = false;
    playerInformation.SetActive(false);
}

// Update is called once per frame
void Update()
{
    openPlayerGUI();
    lifeStats.text = playerController.instance.currentHealth.ToString() + "/" + playerController.instance.maxHealth.ToString();
    speedStats.text = playerController.instance.moveSpeed.ToString();
    forceStats.text = playerController.instance.damage.ToString();
    rangeStats.text = Weapon.instance.lifeTime.ToString() + "m";
}
private void FixedUpdate()
{

}

private void openPlayerGUI()
{
    if (Input.GetKey("i"))
    {
        playerInformation.SetActive(true);

    }
    if (Input.GetKey("p"))
    {
        playerInformation.SetActive(false);
    }

}
```

Voici un extrait de variable du script playerController.cs qui montre les stats du joueur.

Le deuxième extrait de code est le script informationPlayer.cs

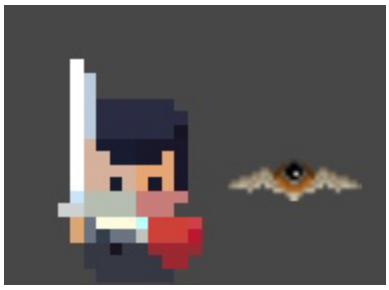
Le code est toujours disponible sur github : Zarkhein/Rogue/Assets/Script

Développement

2.Les Mécaniques principales | Arme

Dans cette partie, nous allons parler de son outil pour tuer les monstres et réussir à monter dans les niveaux. Le joueur possède une arme à côté de lui qui suit le mouvement de la souris, pour tuer le monstre en face de lui il devra juste mettre sa souris devant lui ou sur lui et appuyer sur la touche qui lui permet de lancer un missile pour abattre sa cible.

Pour éviter que le jeu soit trop simple le joueur possède une *Attack Speed* qui rajoute un délai entre chaque tire (mais il pourra reduire le delai de chaque tire en achetant des objets dans le *Shop*) .



Le joueur tire un missile vers la direction où la souris est pointée

```
void Shot()
{
    //traque la souris
    Vector3 difference = Camera.main.ScreenToWorldPoint(Input.mousePosition) - transform.position;
    float rotZ = Mathf.Atan2(difference.y, difference.x) * Mathf.Rad2Deg;
    transform.rotation = Quaternion.Euler(0f, 0f, rotZ + offSet);

    //clock
    if (timeBtwShots <= 0)
    {
        if (Input.GetKey("f"))
        {
            //clone un projectile
            Instantiate(projectile, shotPoint.position, transform.rotation);
            timeBtwShots = startTimeBtwShots;
        }
        if(skillLearned1 == true)
        {
            if (Input.GetKey("m"))
            {
                StartCoroutine(doubleShot());
                timeBtwShots = startTimeBtwShots;
            }
        }
        if (skillLearned2 == true && skillLearned1 == true)
        {
            if (Input.GetKey("m"))
            {
                StartCoroutine(tripleShoot());
                timeBtwShots = startTimeBtwShots;
            }
        }
        else
        {
            print("Vous n'avais pas la connaissance requise...");
        }
    }
    else
    {
        Debug.Log("Reloading");
        timeBtwShots -= Time.deltaTime;
    }
}
```

```
IEnumerator doubleShot()
{
    Instantiate(projectile, shotPoint.position, transform.rotation);
    yield return new WaitForSeconds(.1f);
    Instantiate(projectile, shotPoint.position, transform.rotation);
}
```

IEnumerator est un Thread pour faire simple

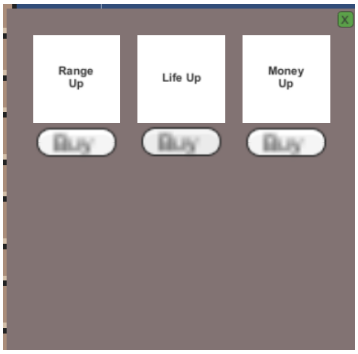
Développement

2.Les Mécaniques principales | Shop

Maintenant nous allons passer au *Shop*. Le shop permet au joueur d'acheter des statistiques brutes pour qu'il soit plus fort contre les ennemis et rendre le jeu plus simple au fur à mesure.

Pour cela, nous avons créé une zone qui va être *Trigger* avec le personnage qui va lui ouvrir un magasin pour qu'il puisse acheter des améliorations de jeux. Si le joueur n'a pas assez d'argent alors il ne pourra pas recevoir sa statistique.

Un *GUI* a été réalisé pour rendre la visibilité plus simple au joueur.



A gauche, voici la première version du shop et la deuxième à droite, le shop s'ouvre seulement quand le joueur s'approche du PNJ (Personnage non jouable) et se ferme quand le joueur s'éloigne.

```
private void Start()
{
    shopMenu.SetActive(false);
    btnShopMoney.onClick.AddListener(buyMoney);
    btnShopHealth.onClick.AddListener(buyHealth);
    btnShopRange.onClick.AddListener(buyRange);
    btnShopExit.onClick.AddListener(shopExit);
    btnShopDamage.onClick.AddListener(buyDamage);


}

private void shopExit()
{
    shopMenu.SetActive(false);
}
```

```
private void OnTriggerEnter2D(Collider2D collision)
{
    print("Test");
    if(collision.gameObject.name == "Player")
    {
        Debug.Log("Bienvenue dans le shop");
        openShop();
    }
}

private void OnTriggerExit2D(Collider2D collision)
{
    Debug.Log("Aurevoir!");
    shopExit();
}
```

Si le joueur clique sur le bouton d'un objet à acheter il appelle une fonction buyXXX

Le deuxième screen montre quand le joueur rentre dans la zone du pnj et sort de la zone de Trigger

Développement

2.Les Mécaniques principales | Les objectifs

Pour que le joueur puisse avancer dans le jeu, il devra casser les totems sur la map. Ce sont des piliers qui sont placés dans la map à des endroits stratégiques pour qu'il soit difficile à atteindre.

Le scriptTotem.Cs est assez simple, on va juste rajouter une *BoxCollider* sur le *GameObject* (Objet dans la scène) puis quand un missile entre dans la collision du totem il perd des points de vie et si sa vie est inférieure à zéro l'objet est détruit par *Destroy(GameObject).*

Quand le joueur tue le totem, il est détruit et il gagne de l'argent aléatoirement une somme prédéfinie par le développeur.

```
void Start()
{
    moneyRand = Random.Range(0, dropMoney);
}
```

```
private void OnTriggerEnter2D(Collider2D collision)
{

    if (collision.gameObject.name == "fireball(Clone)")
    {
        Destroy(collision.gameObject);
        currentLife -= playerController.instance.damage;
        if (currentLife <= 0)
        {
            Destroy(gameObject);
            playerController.instance.moneyPlayer += moneyRand;
            textMoneyPlayer.text = playerController.instance.moneyPlayer.ToString();
        }
    }
```

Extrait du code ScriptTotem.cs

Développement

2.Les Mécaniques principales | Les items

Dans le Projet R, il y a des objets divers comme des différentes potions ou encore des pièges (Le piège n'est pas vraiment un item à proprement parler mais c'est un objet intéractif ou le joueur peut avoir une interaction dessus).

Les *items* ont la capacité d'aider le joueur ou de ralentir le joueur dans la progression il y a des malus et des bonus.

Je prends exemple une potion de soin, le joueur sera soigné mais si le joueur tombe malheureusement un piège, alors il perdra de la vie.

Tous les objets ont une *boxCollider* qui permet d'activer une fonction dans le code .

Les potions dans le jeu et la tourelle qui tire des projectiles sur le joueur pour empêcher qu'il casse les totems

```
[Header("Effet")]
public int effectPotion;

public float vitesse, heal;
public int money;
public Text textMoneyPlayer;


private void OnTriggerEnter2D(Collider2D other)
{
    switch (effectPotion)
    {
        //effet de heal
        case 1:
            PlayerHeal();
            break;
        //effet de speed
        case 2:
            PlayerSpeed();
            break;
        case 3:
            PlayerMoney();
            break;
        default:

            break;
    }
}
```

Le script de gauche est pour toutes les potions (J'utilise un switch case pour permettre d'éviter de faire un script pour chaque effet de potion) chaque case appelle une fonction différente pour la potion. Le développeur a juste besoin de renseigner un nombre dans la variable *effectPotion.*

En dessous, un bout du code de la tourelle, on fait une *coroutine* (Thread) pour lancer un projectile toutes les 1 secondes.

```
IEnumerator spawnBullet()
{
    for (int i = 0; i < 999; i++)
    {
        Instantiate(bullet, new Vector3(gameObject.transform.position.x, gameObject.transform.position.y+0.20f, 0), Quaternion.identity);
        yield return new WaitForSeconds(1);
    }
}
```

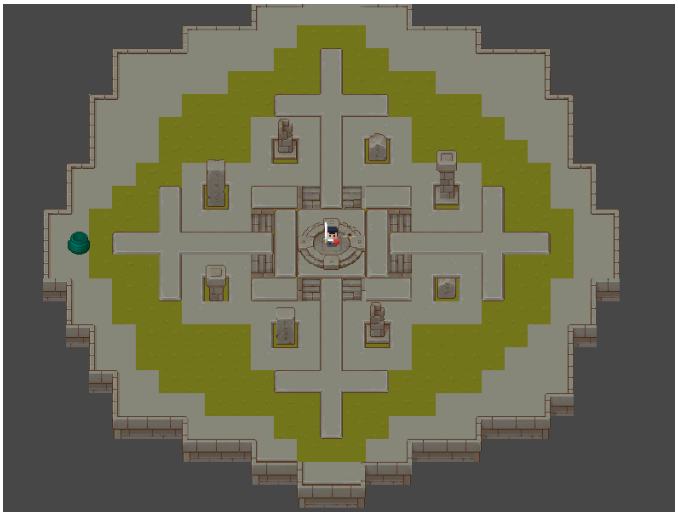
Développement

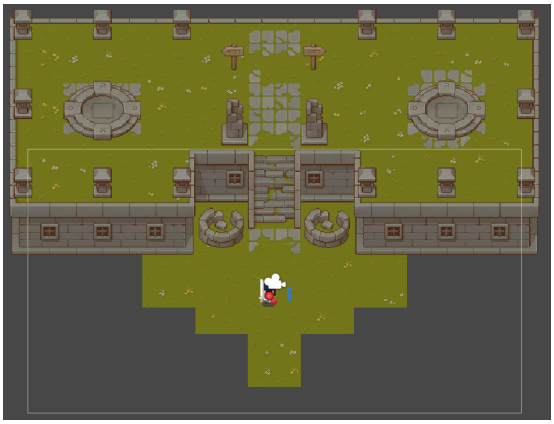
3.Les Mécaniques principales | La map

Au début, le joueur va atterrir dans un *Hub/Lobby* ou il va pouvoir acheter des stats ou encore des compétences. Il aura un téléporteur qui va lui permettre de rentrer dans le labyrinthe ou il va devoir survivre et tuer des monstres pour pouvoir s'en échapper .

Les maps sont composées de *TileMap*(Ce sont des tuiles pour faire simple) on les place dans différents calques *Ground,Water,Wall,Props*(Chacun a un niveau d'élévation ça permet de superpositionner des éléments de la scène.

On a juste rajouté des TilesColliders sur les murs comme ça le joueur ne passe pas à travers.





Exemple de niveau où le joueur devra affronter les monstres.

Conclusion

3.La suite de projet R

Après avoir avancé et travaillé assez durement, nous avons bien apprécié réaliser ce projet et faire une version Alpha du jeu pour voir l impact qu il pourrait avoir sur un grand nombre de personnes.

Faire un jeu vidéo n'est vraiment pas une chose facile, ça prend énormément de temps à cause des bugs, des tests, des idées...

Pour notre part, nous nous sommes amusés à faire ce projet et d'être libre sur le sujet.

Nous pourrions déplorer que nous n'ayons pas eu le temps d'implémenter toutes les *features*(Les idées) mais nous avons ajouté les features que nous souhaitions.

J'ai essayé de rendre le mémoire lisible pour tout le monde sans trop rentrer dans les détails ou dans le code, mais si vous avez des interrogations ou si certains passages ne vous semblent pas très clairs, nous sommes à votre disposition pour répondre à vos questions.

Merci beaucoup de votre lecture et de votre temps.

Cordialement Mathéo Sital & Théo Beulque.