Rapport de deuxième Soutenance

History

Elsa FRANCOIS*(Leader)*
Quentin ROBERT
Lorenzo TAALBA
Aglaé TOURNOIS

Table des matières

1 Intro :

1.1 Rappel :

History est un jeu avec une histoire simple à comprendre, vous contrôlez un robot qui aide un dragon à rejoindre son ère. Pour cela vous aurez le choix entre, solo ou aidé d'un ami. En solo vous contrôlerez les deux, il vous suffira de changer de l'un à l'autre, alors qu'en duo vous contrôlerez chacun un personnage.

L'univers du jeu changera tout au long de l'histoire, elle augmentera petit à petit en commençant en 2D, 16x16pixels jusqu'en 128x128pixels pour passer ensuite sur de la 2.5D et finir sur de la 3D. Vous acquerrez aussi des compétences qui vous seront utiles pour la suite.

Détail :
Dans la suite de notre rapport quand l'on mentionnera des noms de jeux, ils font en réalité référence à nos scènes qui sont en relation à ces jeux.
(Jeux : Pong, Pac-Man, Marion, Street-Fighter).

Pour tout vous expliquer sans rien oublier commençons, faisons la description de notre avancée dans l'ordre de notre tableau d'avancée.

1.2 Tableaux :

Répartition	Elsa	Quentin	Lorenzo	Aglaé
Réseau				
Site				
Caractère				
Maps				
Histoire				
Gameplay				
Combat				
IA				
UI				
Sound				

Attendu $\Longrightarrow$ Fait	Attendu	Fait
Reseau	80%	$\Longrightarrow$ 80%
Site	50%	$\Longrightarrow$ 50%
chara-design	60%	$\Longrightarrow$ 60%
Art carte	60%	$\Longrightarrow$ 60%
Histoire	100%	$\Longrightarrow$ 95%
Gameplay	100%	$\Longrightarrow$ 95%
Combat	60%	$\Longrightarrow$ 60%
AI	50%	$\Longrightarrow$ 60%
UI-Interfaces	80%	$\Longrightarrow$ 90%
Sound	50%	$\Longrightarrow$ 40%

Pour tout ce qui est à 95%, le sujet est normalement fini mais vu que l'on est jamais à l'abri de quelques modifications qui apporterai au jeu une expérience plus agréable, ou un aspect plus pratique que ce soit pour le code ou pour le GamePlay.

2 Reseau :

2.1 Recodage total :

En approfondissant nos recherches sur le réseau et donc plus particulièrement sur Photon, nous nous sommes rendu compte qu'il manquait au code beaucoup de clarté et de logique. Nous avons donc préféré repartir sur de bonne base et tout recoder. Pas mal de lignes ont, aux finales, été gardé, mais cela nous a permis de comprend tous ce qui était le « CallBack » et ce que faisait réellement Photon.

Nous avons pu donc ajouter une fonctionnalité qui prévenait lorsque le projet était lancé sans connexion, la création de « rooms » bien plus efficace et pour finir une liaison au « Master » ce qui nous posait des problèmes avant.

```
public override void OnConnectedToMaster()
{
    Debug.Log("Pun Basic tutorial/Launcher: OnConnectedToMaster() was called by PUN");
    PhotonNetwork.JoinLobby();
}


public override void OnJoinedLobby()
{
    SceneManager.LoadScene("Launcher");
}

public override void OnDisconnected(DisconnectCause cause)
{
    Debug.LogWarningFormat("PUN Basic Tutorial/Launcher: OnDisconnected() was called by PUN with reason {0}", cause);
}
```

Autre gros ajout, une liste des parties créées en multijoueur. La documentation Photon a été bien utile, car bien complète, surtout pour ce qui a été des attributs, des noms et du listage des salles.

Dernier détail que nous mentionnerons serai donc la compréhension de la différence entre le un changement de LoadScene du SceneManager(un simple changement de scène) et un LoalLevel du PhotonNetwork(la création d'un emplacement dans les bases de données de Photon pour accueillir la création de nos salles).

Sinon le principe même reste identique, les salles ne peuvent pas contenir plus deux joueurs, vous pouvez très bien jouer tout seul qu'avec un ami, les joueurs ont un personnage et une caméra indépendantes de l'autre, chacun sa vue et dès lors que quelqu'un change de scène, l'autre est au même moment téléporté.

Nous avions mentionné deux objectifs dans notre précédent rapport :

-Le premier était de faire un meilleur Lobby design, ce qui est atteint et qui sera présenté plus loin dans l'UI.

-Le deuxième était de faire le switch entre les deux personnages lors d'une partie solo, qui est en progression et qui ne devrait pas prendre beaucoup de temps.

Ce qui conclue en quelque sorte notre partie sur le réseau car ces deux éléments ne relèvent en rien du réseau et le reste n'est pas non plus concerné par la partie réseau.

On a néanmoins préféré mettre 95% au lieux de 100% car nous ne sommes jamais à l'abri de potentiels changements.

3 UI :

Nous allons donc directement enchainer sur l'UI, qui concerne principalement le lobby précédemment mentionné, le menu Echap, et certains ajouts d'affichages.

3.1 Menu Echap :

Nous nous étions arrêtés la dernière fois avec un menu échap un peu vide que ce soit dans le visuel que dans le contenu. Vous ne pouviez que quitter le jeu ou enlever le menu. Nous avons donc ajouté la possibilité de changement vos touches à n'importe quel moment pour un gameplay plus agréable.

Aspect Visuel et Partique :

Pour ce qui est du visuel des sprites et police d'écriture ont été ajouté aux boutons.
Pour ce qui est de l'assignement un deuxième canvas est utilisé qui apparait lorsque vous appuyez sur le bouton "Paramètre". Sur ce canvas de nouveaux boutons apparaissent vous montrant les touches qui vous seront utiles au long du jeu.

Pour changer vos touches, vous n'avez plus qu'à appuyer sur le bouton assigné à la commande que vous voulez remplacer puis appuyer sur la touche que vous voulez mettre à la place.

Aspect Codage :

Pour ce qui est du code en lui-même, il a fallu que je créer un Dictionnaire avec comme key des "string" et comme valeur des "KeyCode".

```
public static Dictionary<string, KeyCode> buttonskeys = new Dictionary<string, KeyCode>()
{
    {"UP", KeyCode.UpArrow},
    {"Down", KeyCode.DownArrow},
    {"Left", KeyCode.LeftArrow},
    {"Right", KeyCode.RightArrow},
    {"Jump", KeyCode.Space},
    {"S1", KeyCode.C},
    {"S2", KeyCode.V},
    {"S3", KeyCode.B},
    {"Ex", KeyCode.H},
};
```

Chaque bouton est assigné à une touche donc quand celui-ci est enclenché il envoie sa "string" associé et attend donc qu'une autre touche soit appuyé pour remplacer l'ancienne. Il n'y a pas vraiment de manière de récupérer cette nouvelle touche simplement, donc pour cela on passe en revue toutes les touches disponibles et comparer si elles sont égales. Si elles le sont on arrête la boucle et on met à jour l'affichage.

```
if(Input.anyKeyDown)
{
    KeyCode[] possiblekeys = (KeyCode[])Enum.GetValues( typeof(KeyCode) );

    foreach(KeyCode key in possiblekeys)
    {
        if (Input.GetKey(key))
        {
            buttonskeys[deplacement] = key;
            Updateall();
            break;
        }
    }
```

Petit Changement pour les déplacements :

Même si cela relève plus du "Gameplay", nous avons préféré mettre sa accompagné du reste pour tout enchainer. Les anciens déplacements étaient basés sur les axes avec le "Input.GetAxis" qui ne permettait l'utilisation que des déplacement qwerty ou les flèches. Sachant que le dictionnaire est static on a plus qu'a l'appeler dans notre script "Move" :

```
if(Input.GetKey(Keybinds.buttonskeys["Left"]))
    this.x = -1;
else if(Input.GetKey(Keybinds.buttonskeys["Right"]))
    this.x = 1;
else
    this.x = 0;
```

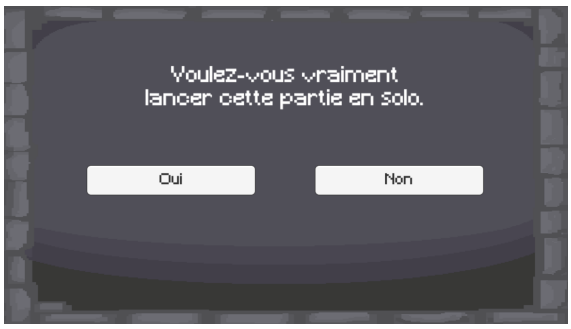
Avec ça si la touche assignée a "Left" est appuyer on renvoie -1 qui sera utilisé plus loin pour se déplacer à gauche. De même pour toutes les directions.

3.2 Lobby :

Ici trois canvas sont présent :

-Le premier permet de choisir entre si vous voulez créer une partie tous seul ou si vous voulez créer une partie accessible par quelqu'un d'autre, avec des images réalisées à la main.

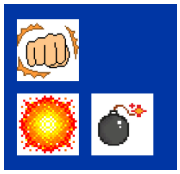
-Le deuxième apparait lorsque l'on appuis sur le côté "Solo", c'est un canvas d'entre-deux juste créé au cas où l'on se serai trompé de côté et pour être sûr que la partie soit créé que pour vous.



-Le dernier apparait donc lorsque l'on appuis sur le côté "Multiplayers". Là est présent la liste des rooms déjà créé et d'un emplacement pour pouvoir créer votre propre room.

3.3 Interface :

Dans la partie du Street Fighter, des pixels arts représentant les différentes attaques possédant le joueur ont été implémenté avec un temps de rechargement ce qui évite au joueur la possibilité de pouvoir envoyer plein de projectiles d'affilés.



Cela a été mis en place en trois étapes :

-La première est déjà vu pour les barres de vie, c'est des "sliders" qui permettent de griser les compétences utilisées et les dégriser petit à petit.

-La deuxième est donc d'associer ces "sliders" à un script qui, selon la compétence, fasse un compte à rebours plus ou moins long grâce au "FixedUpdate" et une décrémentation de nombres.

```
private IEnumerator StartCoolDown(string type)
{
    switch (type)
    {
        case "contact":
            Contactvalide = false;
            yield return new WaitForSeconds(CDR_contact);
            Contactvalide = true;
            break;
                You, il y a 1 seconde • Uncommitted change
        case "project":
            Projvalide = false;
            yield return new WaitForSeconds(CDR_proj);
            Projvalide = true;
            break;
```

-La dernière est donc la manière dont on enlève l'accès au lancement des compétences. Pour ça, des booléens sont créés pour chacune des différentes attaques. Pour faire un *arrêt dans le temps*, une fonction IEnumerator est créé pour appeler des fonctions "WatiForSeconds" qui accompagné du nombre de seconde de rechargement à attendre. Les booléens sont donc changé en "false" avant et remis en "true" après le temps d'attente.

```
void FixedUpdate()
{
    if(!SF_Power.Projvalide && proj_tps == 0)
        proj_tps = SF_Power.CDR_proj*51;
    proj_tps -= proj_tps > 0?1:0;
```

Donc si on revient sur l'ensemble de nos interfaces, quasiment tout est implémenté, on pourrait donc mettre 100%, cependant ne sachant pas ce qui pourrais être plus agréable nous préférons mettre notre UI plus a 95% au cas où des modifications seraient apportées.

4 L'IA :

L'IA est omniprésent dans notre jeu, que ce soit en multijoueur qu'en solo, nos deux personnages doivent être présents sur la scène et avancer ensemble. En solo, si vous contrôlez le robot, le dragon doit pouvoir vous suivre partout où que vous alliez. En multijoueur le principe est le même a l'exception que, si un des deux joueurs se déconnecte, alors le personnage qu'il contrôlait doit être pris en charge pas l'IA pour ne pas nuire à l'avancer du deuxième joueur.

Cela s'applique tout le temps, quelque soit le personnage et le mode de jeu (de la 2D à 3D), sauf au début car le dragon n'a toujours pas éclos". De plus des IA "secondaire" seront aussi présent représentant les ennemis ou des items.

A la première soutenance nous avions fait un semblant d'IA avec les plateformes du Pong et le script du boss du Street-Fighter.
Nous avons pourtant décidé de laisser le script du boss de côté le temps d'avoir plus d'idée sur comment seront gérer ses attaques et ses différentes phases, et partiellement recoder l'IA du Pong car il ne convenait plus à ce que l'on voulait.

4.1 Pong :

Dans la deuxième salle, une IA, associé à un chariot, est là pour pouvoir calculer et contrer les rebonds de l'œuf de notre dragon. Celle-ci devant accéléré à l'approche de l'œuf.

Pour cela nous avons utilisé la méthode "Lerp" qui permet de calculer une position par rapport à une de départ et une d'arrivée ainsi que d'une vitesse.
La position de départ étant dans notre cas la position actuelle, celle d'arrivée la hauteur de l'œuf et la vitesse calculée grâce à un petit programme ci-dessous.

```
transform.position = (Vector3)Vector2.Lerp( a: (Vector2)new Vector3(x,y), b: (Vector2)new Vector3(x, y: playery),speed);

 playerx  = player.transform.position.x;
 speed = (float)1 / (int)Pow((playerx-x),2);
```

Tout ça avec un "GameObject" avec un "collider" et une apparence d'œuf nous permet de faire un semblant de "Pong".

4.2 Pac-Man :

4.2.1 IA secondaire :

Dans le pacman des ennemis font une ronde pour ajouter du dynamisme au labyrinthe. Elles se déplacent grâce à une liste de points posés à chaque tournant du parcours. Elles se dirigent donc vers un point et arrivé à ce point il passe au suivant.
Il existe deux types de parcours soit en faisant une boucle (quand il arrive au dernier point il se dirige vers le premier) soit en faisant le chemin inverse (après avoir incrémenter jusqu'au dernier point il décrémente jusqu' au premier). En code cela peut se traduire comme ça :

```
//si arrive près du pts alors on passe au suivant
if (Vector3.Distance(a:transform.position, b:targets.position) < 0.3f)
{
    target_place += next;//passe au point suivant

    if (loop && target_place == len) //si c'était le dernier point et que c'est une boucle on revient au début
        target_place = 0;
    else if (!loop && target_place == len - 1 || target_place == 0)
            //si c'est pas une boucle et que c'est le dernier ou le premier on inverse le sens d'incrémentation
            next *= -1;

    targets = pts[target_place];
}
```

Avec un point de départ, un point d'arrivé et l'apparition de notre joueur, ça nous créer un labyrinthe dans l'univers du Pac-Man.

4.2.2 IA principale :

Mais pour ce qui est de l'IA principale, devant nous suivre pendant tout le jeu, nous avons d'abord essayé de le faire par nos propres moyens, comparer les distances entre les deux personnages, sa position relative au vrai joueur, et essayer de le rapprocher sans se prendre d'obstacle mais nous avons vite compris que des complexités rendraient le travail infaisable.

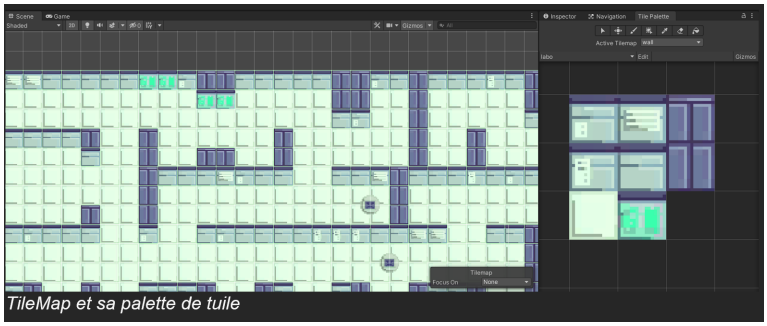
Alors nous nous sommes penché vers le "PathFinding" qui est une IA permettant de trouver un chemin vers une cible automatiquement et ça en prenant en compte les murs et autres obstacles et les déplacements de la cible.

Sur unity il n'existe qu'un "PathFinding" 3D alors nous avons dû chercher un moyen de l'amener pour les scene2D, mes recherches nous ont amené à un package de "H8man", "NavMesh Plus", permettant de le faire en 2D.

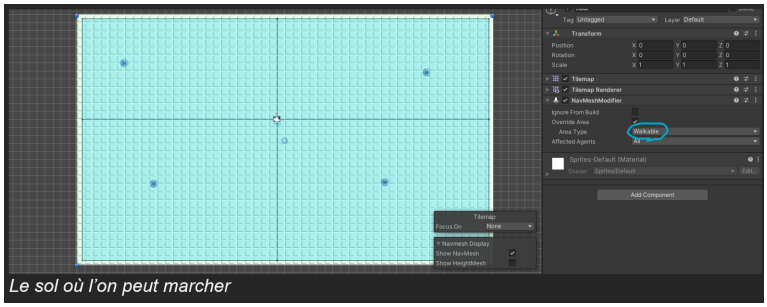
Si on applique ça au Pac-Man, il faut donc faire une IA qui peut se déplacer dans un labyrinthe vu d'au-dessus, se déplacer vers un joueur lui-même en mouvement et éviter les autres IA.

Pour arriver à cet objectif il faut dans un premier temps créer un "navmesh" qui est un outil permettant de mettre en place le "PathFinding".

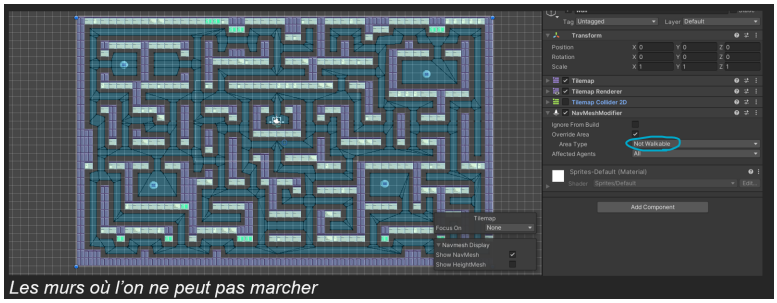
Maintenant il faut définir le terrain pour ce "navmesh", donc il a fallu rajouter des "navmesh modifier" sur le tilemap (une grille où l'on peut rajouter des tuiles pour former un tableau) qui vont permettre de définir la ou l'IA peut marcher ou pas.



TileMap et sa palette de tuile

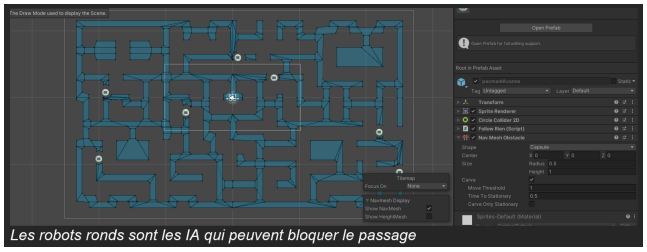


Le sol où l'on peut marcher



Les murs où l'on ne peut pas marcher

Un dernier problème vient sur ce terrain, c'est qu'il y a d'autres IA qui peuvent bloquer le chemin donc nous devons faire en sorte qu'elles ne peuvent pas prendre un chemin bloqué par une autre IA. Pour ça nous avons utilisé un "navmesh obstacle" qui permet de faire une zone not walkable (une zone inaccessible par l'IA) et de la mettre à jour automatiquement.



Les robots ronds sont les IA qui peuvent bloquer le passage

4.3 Mario :

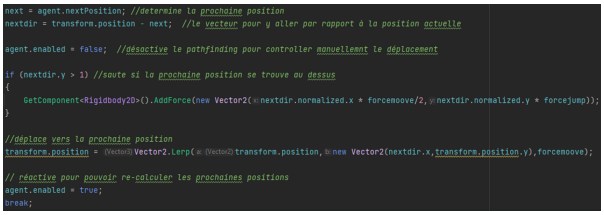
Pour le Mario l'implémentation de l'IA est beaucoup plus compliquée. Comme dit précédemment nous avons premièrement essayé par nos propres moyens mais beaucoup de cas ne pouvant être traiter par l'IA nous avons finalement essayé de trouver une autre voie et cela grâce au "PathFinding".

Un second problème est que le "navmesh" ne permet pas de gérer les déplacements par nous-même, celle-ci gérer donc par le "navmesh" lui-même et donc l'IA ne tiendrai pas compte de la gravité.

Alors nous avons voulu tenter un "PathFinding" 3D, mais un nouveau problème vient, celui des sauts, ceux-ci devant être géré grâce à des "navmesh link" qui sont comme des cordes joignant deux points de la map et prolongeant donc la zone walkable (accessible) par l'IA.

Ces sauts doivent donc être créés en amont avec la map ce qui est impossible quand on doit ajouter des objets dynamiques comme des caisses déplaçables qu'on doit sauter dessus.

Actuellement nous sommes revenu au "PathFinding" 2D que nous utilisons uniquement pour déterminer le prochain point où doit aller l'IA que nous utilisons par la suite pour faire bouger manuellement l'IA tout en prenant en compte la gravité et les sauts bien que cette méthode ne soit pas encore parfaitement au point.



```
next = agent.nextPosition; //determine la prochaine position
nextdir = transform.position - next;  //le vecteur pour y aller par rapport à la position actuelle

agent.enabled = false;  //désactive le pathfinding pour controller manuellemnt le déplacement

if (nextdir.y > 1) //saute si la prochaine position se trouve au dessus
{
    GetComponent<Rigidbody2D>().AddForce(new Vector2(x:nextdir.normalized.x * forcemoove/2,y:nextdir.normalized.y * forcejump));
}

//déplace vers la prochaine position
transform.position = (Vector3)Vector2.Lerp(a:(Vector2)transform.position,b:new Vector2(nextdir.x,transform.position.y),forcemoove);

// réactive pour pouvoir re-calculer les prochaines positions
agent.enabled = true;
break;
```

D'autre ajout ont été fait mais qui ne nécessite pas un point entier :
— un script pour que la caméra nous suive.
— réorganisation des scripts des joueurs.
— commencement d'un moyen de sauvegarde.
— Une nouvelle attaque pour le Street-Fighter.
— Un texte animé qui apparait au fur et à mesure.
— Un décor temporaire pour le labyrinthe du PAc-Man.

5 Map :

(*Rappel : tout est entièrement designé et créé à la main*)
Nous avions, pour la première soutenance, déjà quelques bases de décors pour nos "Rooms" :

— Poubelle.
— Lampadaire.
— Drap de stand de restaurant.
— Croquis de future immeuble et kiosque.

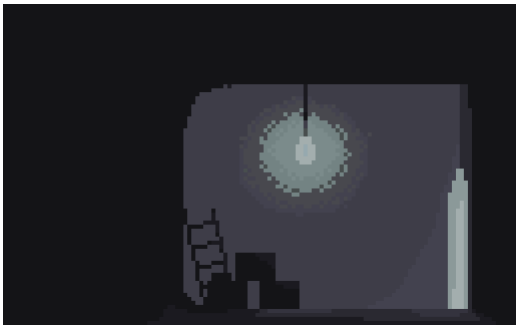
Si on repasse en vue toutes les scènes et avancées qu'il y a, on a donc :

1. Un fond style entrepôt pour l'éveil du robot.
2. Un bureau pour le Pac-Man.
3. Une salle d'experience pour l'éclosion de l'œuf.
4. Une scène vue de haut représentant les couloirs du laboratoire ; pour l'instant temporaire mais reste tout de même dans l'ambiance du début du jeu.
5. Une scène très large que l'on parcourra pour le Mario. N'est pas encore implémenté mais la quasi-totalité des éléments pour la créer son fini.
6. Une scène pour le boss, gardera ce font de scène style ville.

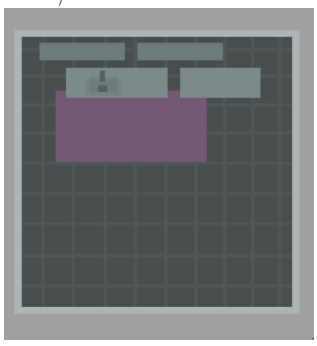
Depuis cette soutenance beaucoup de décors ont été fini, créé ou en cours de création.
Il y a bien la moitié des décors qui ont été implémenté dans nos scènes, et l'autre fini mais pas encore mis en place.

Implémentés :

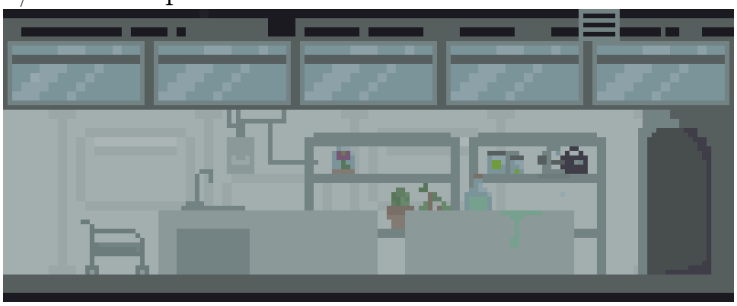
1/Entrepose



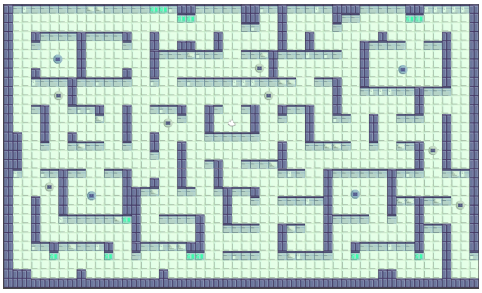
2/Bureau



3/Salle d'experience



4/Couloir du Laboratoire

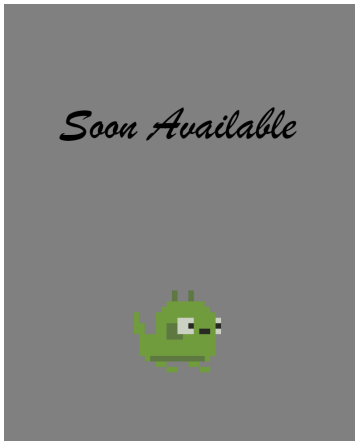


Non-Implémentés :

Les différents éléments à disposition sont donc :
Partie de mur que l'on peut répéter pour faire de grand immeuble, un haut d'immeuble pour le finir, une porte d'immeuble, une fenêtre de restaurant, un kiosque (*EQLA, Your Name et Le voyage de Chihiro pour les connaisseurs*), deux différents arrêt de bus, des pavés de rue et les anciens éléments.

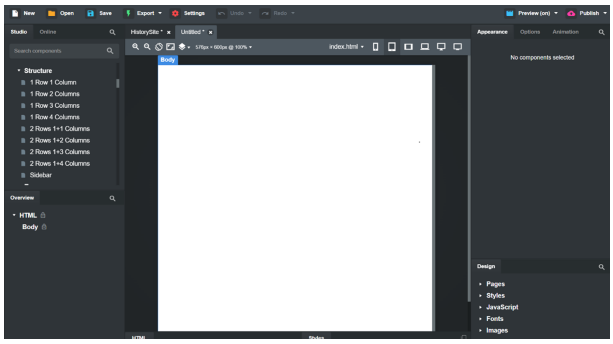
6 Site :

Lors de la première soutenance, nous devions présenter un début de site, nous avions donc présenté une page sur laquelle on voyait un gif de notre dragon courant. Comme tout ce qui est du design et du dessin il a entièrement été fait à "la main" image par image.

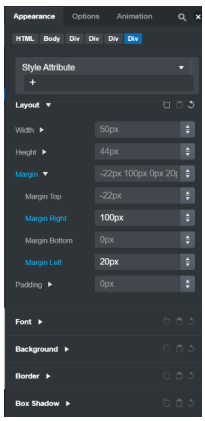


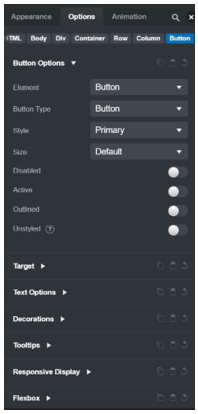
Ce début de site avait été fait sur Repl.it. Depuis, un nouvel logiciel a été téléchargé et utilisé : Bootstrap.

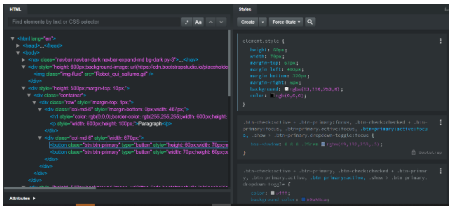
Ce logiciel nous permet de créer une structure/ un squelette du site bien plus simplement, grâce à plusieurs structures pré-implémentées. Il est donc possible d'introduire nos textes, images et liens, dans différents compartiments, bien plus aisément et rapidement que si on avait dû tout coder à la main, ce qui était prévu. On est alors partis d'une page blanche et, au fur et à mesure, on a construit le squelette du site grâce à un enchainement de différents compartiments choisis



Dans ces compartiments, il est ensuite possible d'ajouter des zones de texte, des boutons, des images ou encore des interactions. Pour chaque élément rajouté, il est possible de modifier toutes ces options. Si une option n'est pas modifiable dans les réglages proposés, il est toujours possible d'accéder au code HTML et de le modifier manuellement (Ce qui est souvent nécessaire pour la dimension des images et leur place sur la page). Connaître les bases du HTML et son fonctionnement est alors capital afin de se repérer et avoir un site à la hauteur de nos attentes, car les modèles préfabriqués ne sont pas parfaits.

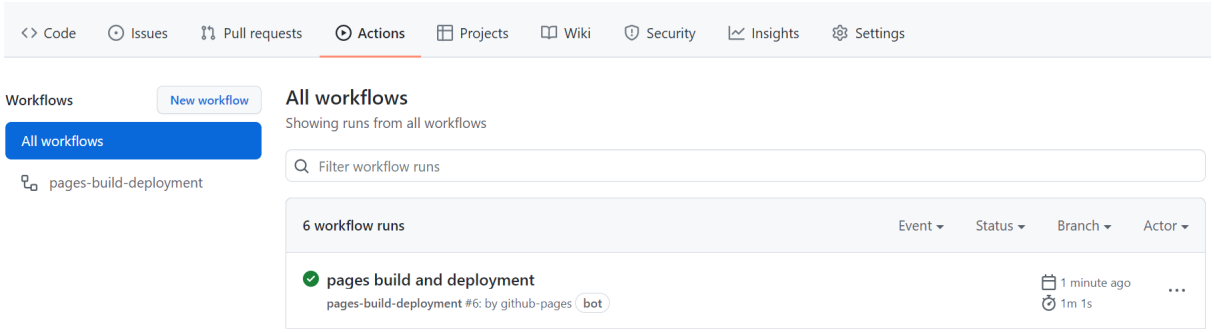






Une fois, le squelette du site créé, il a été nécessaire de créer les images qui remplaceront les espaces blancs qui remplissent pour le moment le site. Nous nous sommes donc chargés de dessiner encore une fois à la main chaque bouton et chaque flèche du site. Une fois les design créés et importés, il fallu redimensionner chaque image afin que celle-ci corresponde à la place attribuée.

Une fois le site en lui même créé, il faut le push sur un repot git. Sur le site de gitHub, on peut donc voir que le projet s'est correctement uploader. Le site est maintenant acessible pour nous tous.



7 Autres petits ajouts :

7.1 Animations :

Nous avons commencé à ajouter quelques animations pour que le jeu paraisse moins monotone. Des animations plutôt simples tel que celle du robot ou de l'œuf qui passe en boucle les mêmes images.

image des images animation

Puis une un peu plus compliqué car il faut l'activer et la désactivé a un moment bien précis. Sur la première scène, notre robot est éteint, un dialogue apparait mettant en scène l'allumage du robot, une fois le texte passé le robot commence à s'animé et à s'allumer.

Après un certain temps le composant « animation » du robot est détruit, donc le robot garde son apparence d'allumé.

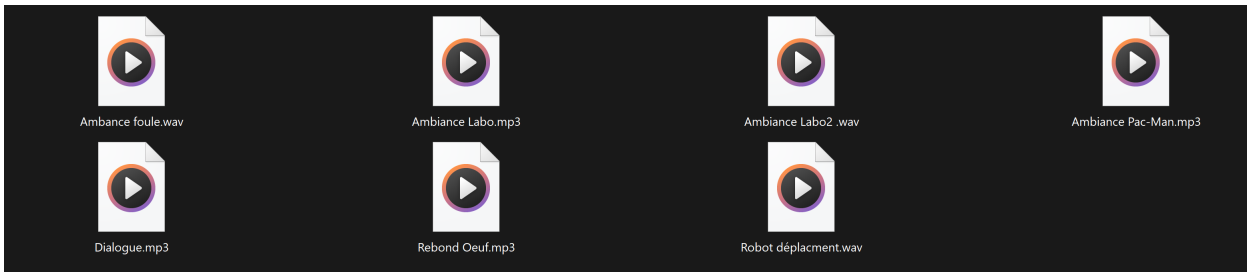
7.2 Sons :

Pour toute la partie son, les implémenter à nos mouvements, à nos scènes, nos ambiances, nos animations alors qu'elles ne sont pas toutes au point et fini n'est pas forcément pratique.

Ce n'est pas pour autant que nous n'avons rien fait. Nous étions tous d'accord sur le fait qui valait mieux implémenter tous les sons en même temps pour éviter d'en oublier. On a donc commencé à faire un dossier les contenant et à les ajouter petit à petit au long du développement du jeu.

Il nous faudrai :
Robot $\Longrightarrow$ Eveil et Flotte
Dragon $\Longrightarrow$ Eclosion et Marche
Œuf $\Longrightarrow$ Rebondit
Ambiance $\Longrightarrow$ Foule – Labo – bruit d'ambiance

On a pour l'instant :



8 Problèmes rencontrés :

Durant ce projet, nous avons rencontré divers problèmes autre que ceux concernant l'IA. En effet, le travail sur un même environnement amène à la création de merge conflict, plus ou moins long à régler. Pour diminuer le risque d'en avoir, la communication au sein du groupe est un point essentiel et les zones d'actions de chacun se précise au fur et à mesure pour les éviter.

Un autre de nos soucis fut les dimensions. Comment faire pour que ce que je vois sur un écran rende la même chose sur un autre avec une résolution différente ? Pour le site ou le jeu lui-même, cette question s'est imposé.

En effet cela devient gênant si le site est totalement déstructuré sur nos téléphones et se déforme d'un ordinateur à un autre. Similairement, le jeu en plein écran ne se comporte pas de la même façon partout. Dans les deux cas, trouver les dimensions adéquates fut un challenge afin d'avoir un confort visuel, un enjeu au moment de la création de design.

9 Conclusion :

9.1 Fait :

Pour conclure récapitulons tout depuis le début :

· La partie réseau est presque fini. Il est possible qu'on n'y revienne plus tard en raison de l'ajout d'un Bonus : la Sauvegarde.

· La partie sur les interfaces est elle aussi bien avancé. Il est aussi possible qu'on revient dessus pour certaines modifications, que se soit sur les graphismes ou sur l'aspect pratique de son emploi.

· L'IA a posé beaucoup de problème et est une partie très fastidieuse, il n'empêche qu'elle avant bien et à un rythme plus que suffisant pour en voir la fin. Les IA « secondaire » sont faites et si on en ajoute le principe sera, entre autres, le même. Pour ce qui est des IA un peu plus important, tel que le suivie du personnage principal ou le script du Boss, il manque quelques recherches et quelques journées et le but sera atteint.

· La Map avance bien, bien que peu de scène soit entièrement fini, et qu'il reste encore quelques éléments de décors pour la finition à faire le gros du travail est fait.

·Pour ce qui est du site, ce qui a presque été le plus dur c'est la prise en main du logiciel. Une fois le logiciel métrisé, la conception du la page est allée beaucoup plus vite bien que pas encore fini.

·Les détails seront ajoutés au fur et à mesure pour que le jeu puisse se construire avec leur implémentation sauf bien évidemment pour le son qui sera ajouté en dernier.

·Pour ce qui est des problèmes, bien qu'ils aient été embêtant on est passé outre et soit on a fait autrement soit on l'a résolu.

9.2 Prévu pour la prochaine fois :

Pour ces dernières semaines on s'attardera sur toutes les finitions des différentes parties et ensuite on s'attaquera donc à notre dernière partie qui concerne notre « Pokemon ».

Un monde beaucoup plus ouvert ou ne fera que le code. Pour cette partie, notre maps sera constitué d'éléments libre de droits pris d'internet. On se concentrera surtout sur les différentes animations, le code et la création d'un univers agréable à parcourir.

Elsa et Aglaé ayant beaucoup touché à la partie graphique du jeu se concentreront plus sur tout ce qui est la partie codage et inversement pour Lorenzo et Quentin.

Pour finir sur le site, bien qu'il ne soit pas si avancé sa finition ne devrait pas prendre tant de temps que ça.