Favetto Ambre
Rougé Maxence

Groupe 8

Maze Builder

Table des matières

Introduction **3**
1 - Présentation du jeu **4**
2 - Présentation des outils et de l'organisation **4**
2.1 - Présentation des outils utilisés 4
2.2 - Présentation de l'organisation 5
3 - Présentation des différentes fonctionnalités **7**
3.1 – Le menu principal 7
3.2 – Le menu d'options 8
3.3 - L'éditeur de niveaux 9
3.2 – L'interface de tests de niveaux 15
4 - Détails de code de certaines fonctionnalités **16**
4.1 – Sauvegarde d'un labyrinthe 16
4.2 – Chargement d'un labyrinthe 18
4.3 – Pathfinding et Algorithme A* 19
Conclusion **23**

Introduction

Nous avions le choix entre différents projets à réaliser, par groupes de deux, dans le cadre du module de suivi de projet.
Notre choix s'est porté vers le développement d'un logiciel avec conception d'une IA car c'est quelque chose que nous n'avions jamais fait et nous étions curieux d'en apprendre plus sur cette facette de la programmation.

En plus de nous apporter de nombreuses connaissances au niveau du développement et de la base de données, ce projet nous a également demandé une rigueur au niveau de la gestion de projet. Effectivement nous avions, à notre sens, vraiment peu de temps vis-à-vis de la quantité de nouvelles informations et de toutes les idées que nous voulions mettre en pratique.
Le fait de ne pas connaître la plupart des technologies présentes dans le projet nous a rendu les tâches plus compliquées à prévoir et à estimer.

Ce document reprendra une présentation de toutes ces technologies que nous avons eu l'occasion d'utiliser, suivi d'une présentation des différentes fonctionnalités du jeu avec quelques mises en lumière de différents codes qui nous semblent pertinents.

1 - Présentation du jeu

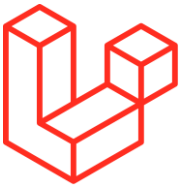
Notre jeu, **Maze Builder,** est un jeu vidéo de type “Labyrinthe”.
L’utilisateur va avoir la possibilité de créer des niveaux à l’aide des différentes pièces mises à sa disposition.
L’IA déterminera ensuite si ce labyrinthe est faisable tout en trouvant le chemin le plus cours pour atteindre l’arrivée.
L’utilisateur pourra, lorsque le niveau est faisable, le sauvegarder dans une base de données, pour le récupérer plus tard afin de le tester et/ou l’éditer.

De nombreuses fonctionnalités sont mises en place pour rendre la création et la gestion des labyrinthes le plus claire et fluide possible (cf. 3 - Présentation des différentes fonctionnalités du jeu)

2 - Présentation des outils et de l’organisation

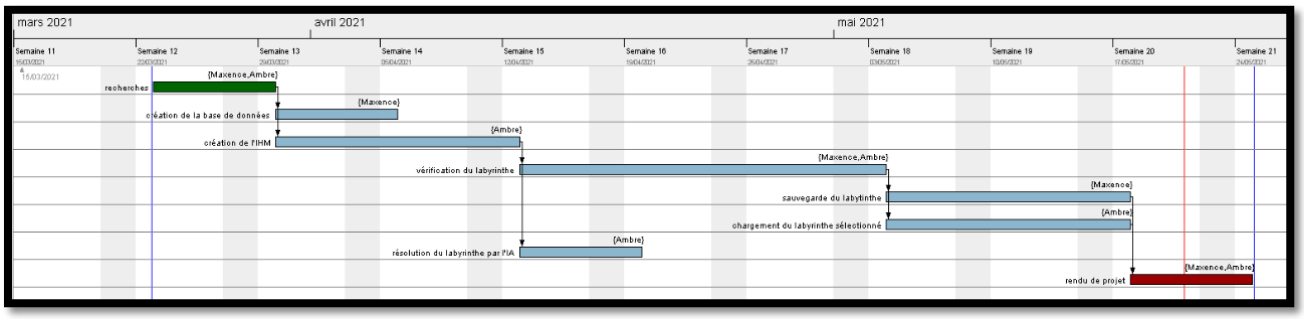
2.1 - Présentation des outils utilisés

• **Unity** est un moteur de jeu multiplateforme. Ce logiciel a la particularité d'utiliser soit du code en C# soit en javascript. Nous avons décidé de l’utiliser afin de nous faciliter la création de toute la partie graphique de notre logiciel. Nous avons opté pour le C# pour ce qui est du code car bien qu’il soit plus exigeant il nous donnait la possibilité de réaliser des actions plus complexes.



• **Laravel** est un framework web open-source écrit en PHP, son installation est basée sur le gestionnaire de paquets Composer. Laravel fournit des fonctionnalités en termes de routage de requêtes, de mapping objet-relationnel de migration de base de données, de gestion des exceptions et de tests unitaires. Nous avons décidé de l’utiliser pour gérer notre base de données locale car elle nous semblait facile d’accès, complète tout en étant gratuite.

2.2 - Présentation de l'organisation



Organisation du GANTT

Nous avons commencé notre projet par la conception d'un Gantt. Nous l'avons découpé en différentes tâches qui nous semblaient pertinentes :

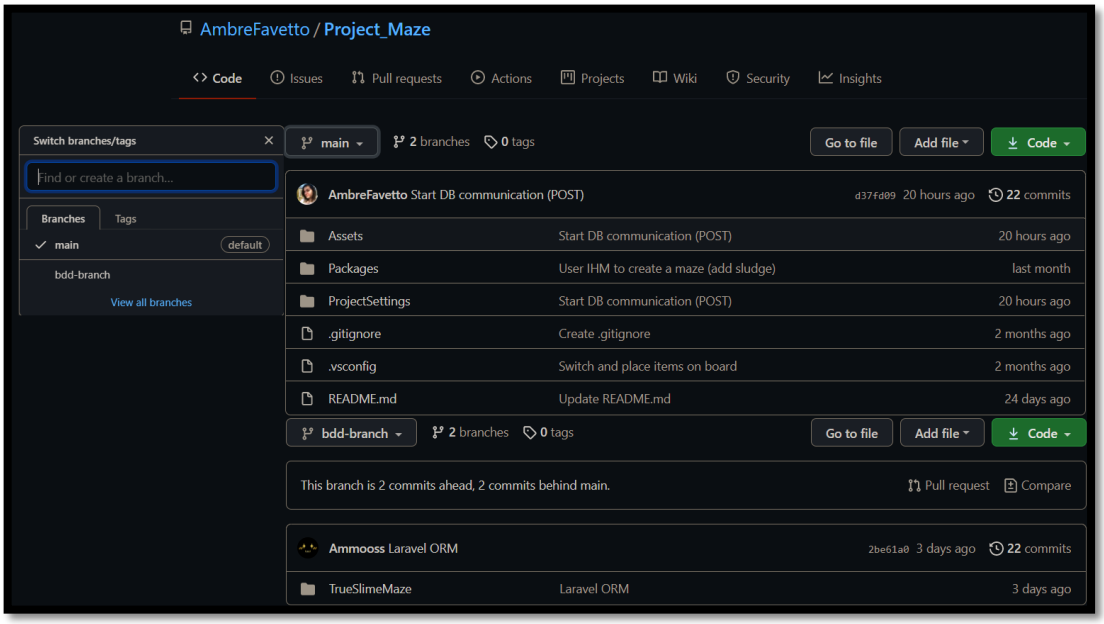
• **création de la base de données** : pour avoir une vision claire de tous les objets que nous allions devoir manipuler.

• **création des IHM** : indispensable pour avoir un aperçu visuel de ce que nous allions coder ensuite (déplacement de l'IA, pathfinding, chargement en base de données, …). C'était également une façon d'avoir une première approche un peu plus légère d'Unity (*moteur que nous n'avions jamais utilisé*).

• **vérification du labyrinthe et résolution du labyrinthe par l'IA** : c'était ici la suite logique. Effectivement, nous avons décidé de proposer une légère modification dans le sujet. Nous ne sauvegardons que les labyrinthes corrects. Il était donc nécessaire d'avoir cette vérification avant la sauvegarde en base de données.

• **sauvegarde et chargement du labyrinthe** : dernière étape et non des moindres de ce projet. Permettant de récupérer un labyrinthe déjà créé et de le tester à n'importe quel moment.

Nous avons essayé au maximum de suivre ces prévisions tout au long du projet même si, au final, nous n'avons pas atteint tous les objectifs que nous nous étions fixés.



GitHub du projet

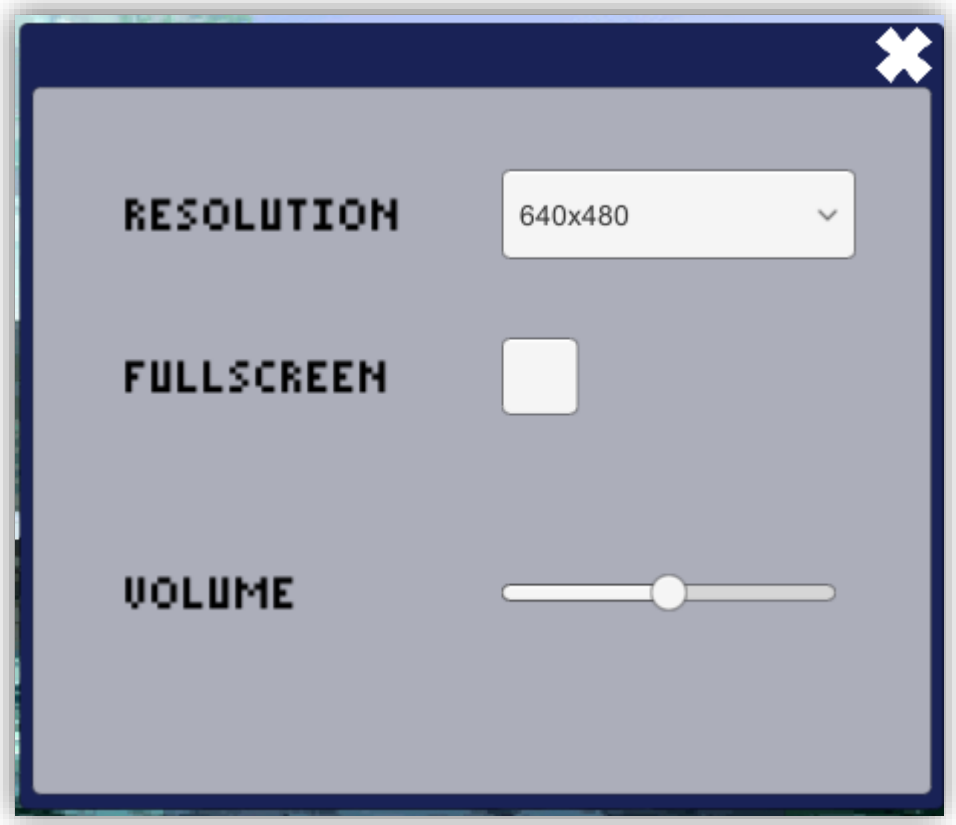
Nous avons utilisé Github pour tout ce qui est partage et sauvegarde de notre code. Cela nous permettait d'avoir un suivi tout au long des semaines de l'avancée de chacun.

3 - Présentation des différentes fonctionnalités

3.1 – Le menu principal

Il nous permet de choisir d'accéder soit à l'éditeur de labyrinthe soit au testeur. Nous avons également ajouté un petit onglet d'options à modifier pour ajouter un peu plus de profondeur à l'expérience utilisateur.

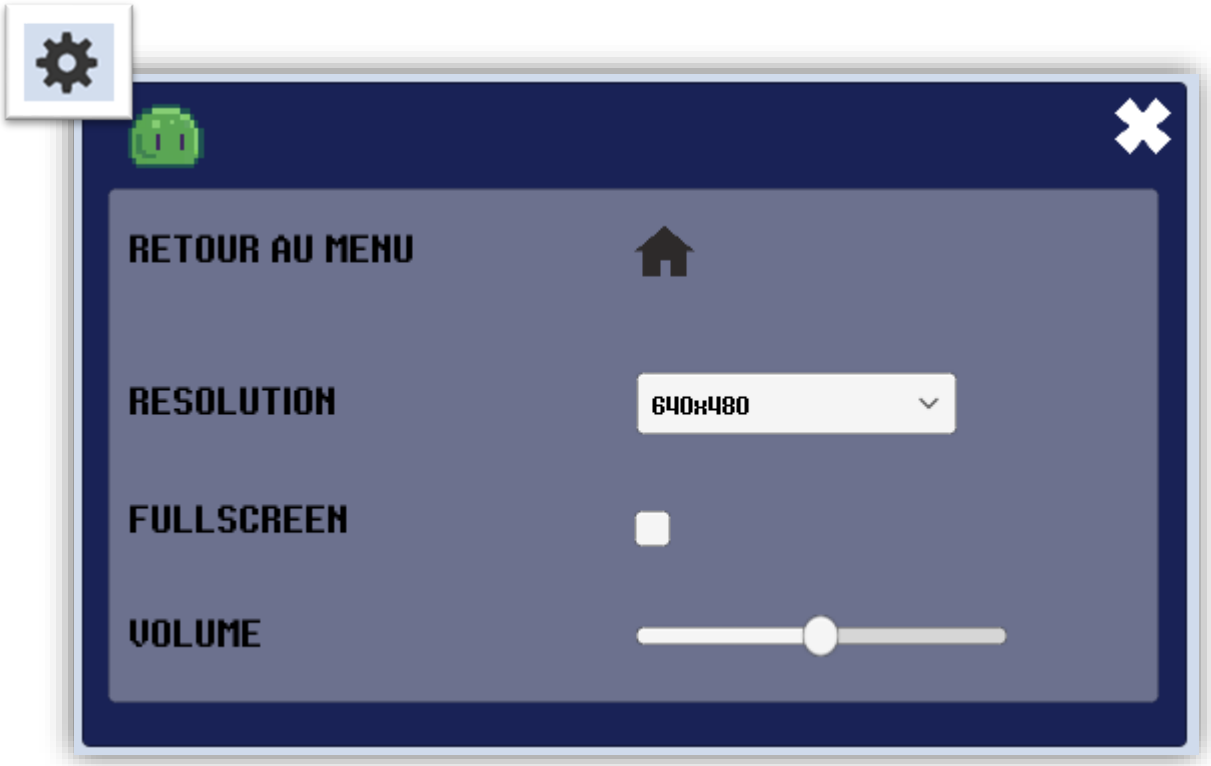
3.2 – Le menu d’options



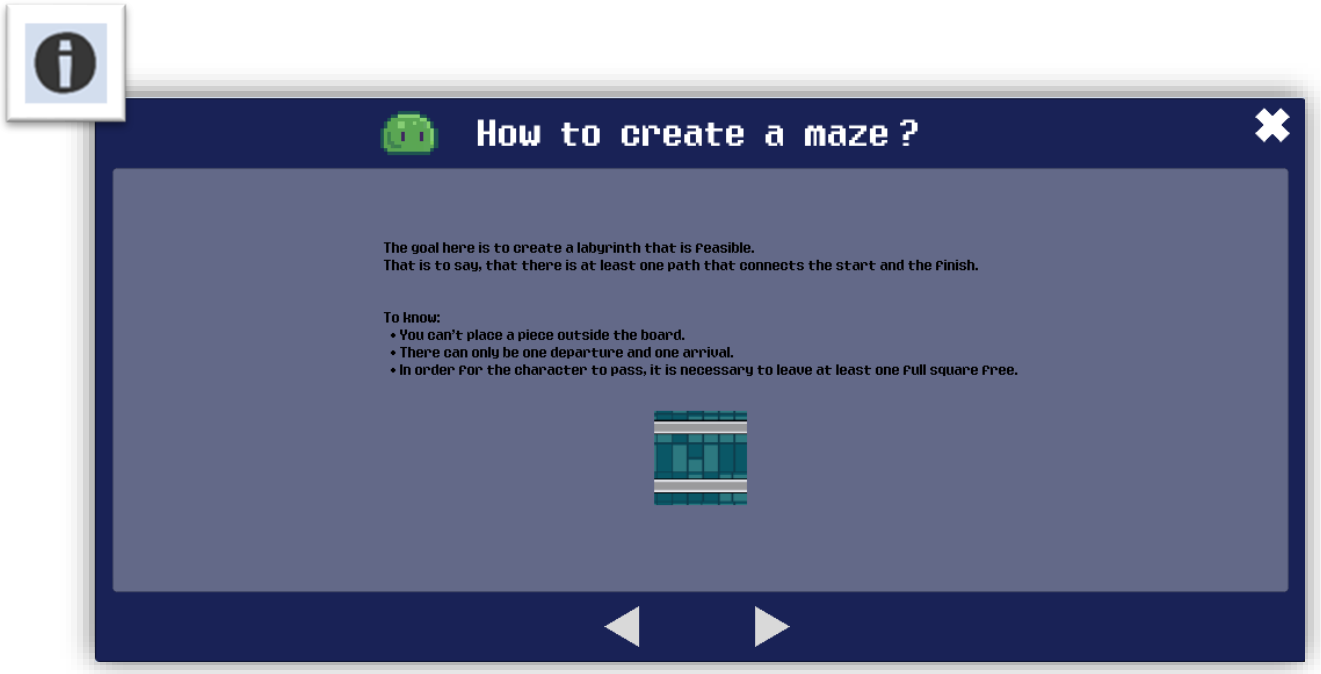
Le menu d’options permet de modifier les options de base du logiciel. Chacune de ces options est fonctionnelle. C’est un menu très simple qui pourrait être amené à évoluer par la suite.

3.3 - L'éditeur de niveau

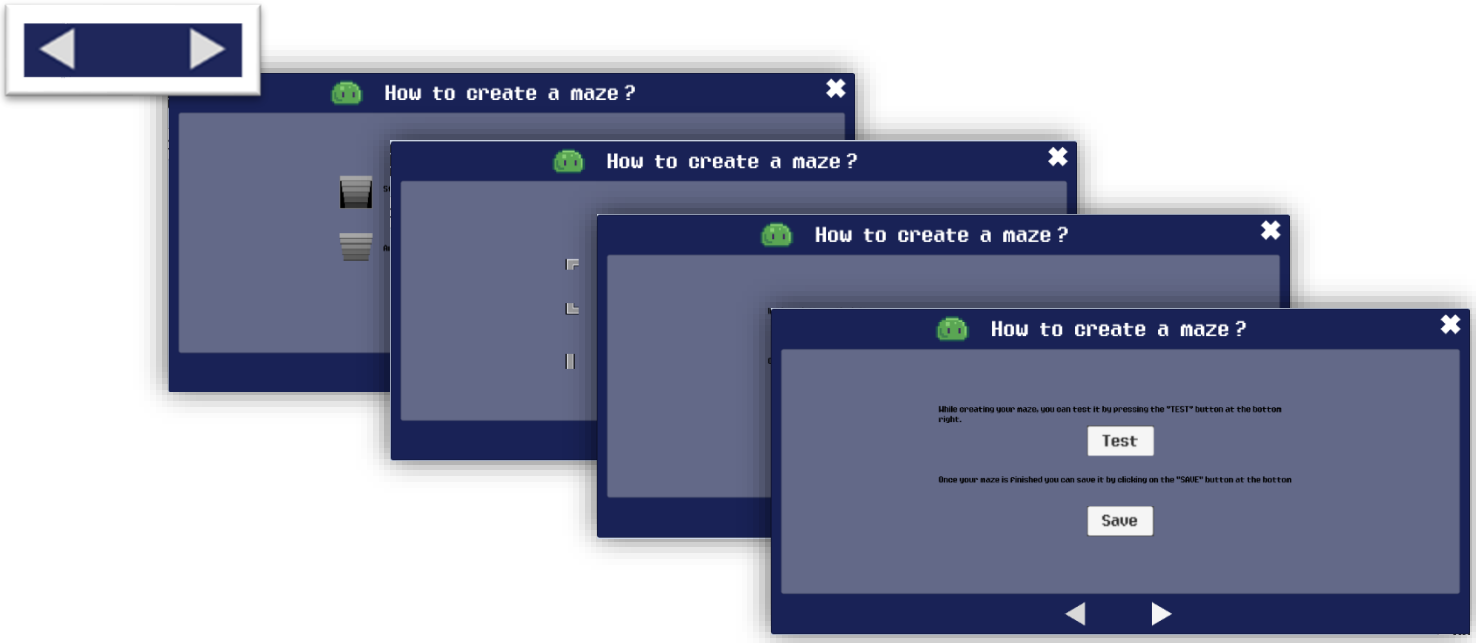
C’est ici que les différents labyrinthes vont être créés.



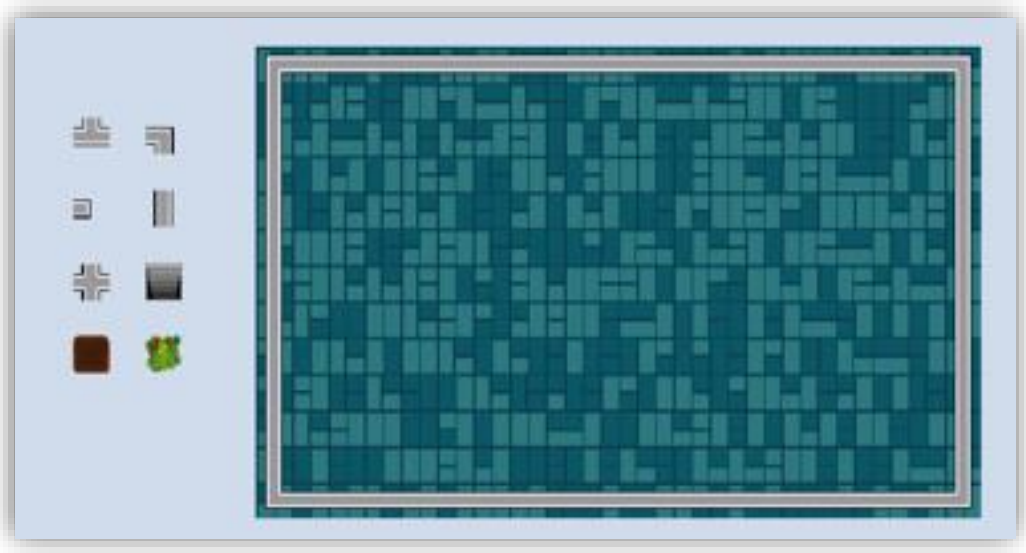
Le **bouton de paramètres** nous ouvre un menu nous permettant soit de retourner au menu principal soit de modifier les différentes options vues précédemment.



Le **bouton d'informations** nous permet d'avoir accès aux différentes aides et explications quant à la création du labyrinthe.



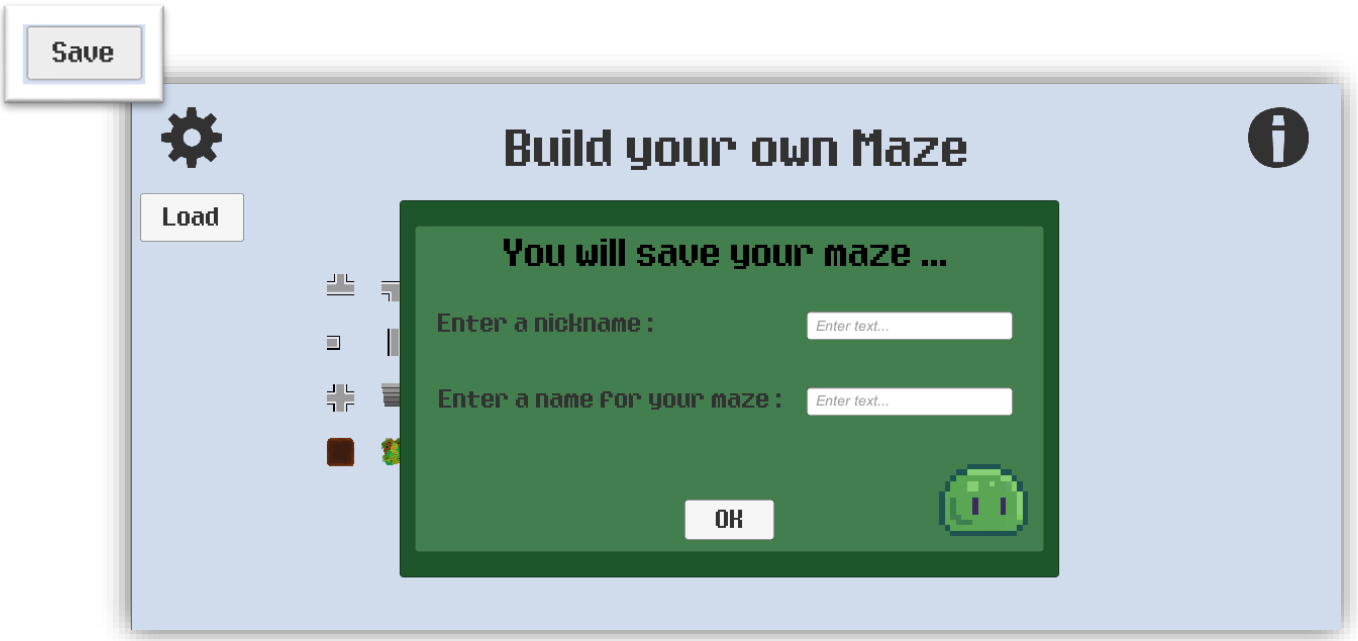
Les **flèches** situées en bas du menu d'informations nous permettent de naviguer facilement d'une page à l'autre.



Voici le plateau et la palette permettant de créer nos labyrinthes. Leur utilisation est assez simple :
Un *clic droit* sur une pièce de la palette permet, pour les murs de les pivoter et pour les escaliers de passer d'un escalier de départ à un escalier d'arrivé. Cela n'aura aucun effet sur la boue et les feuilles (le piège).
Un *clic gauche* sur une pièce permettra de la placer au centre du plateau.
Une fois qu'une pièce est placée sur le plateau il est possible de la déplacer en maintenant un *clic gauche* enfoncé sur celle-ci.
Un *clic droit* aura pour effet de supprimer une pièce placée sur le plateau.

Le **bouton « TEST »** n'est accessible qu'une fois qu'un escalier de départ et un escalier d'arrivée sont placés. Cela fait apparaître notre personnage sur

l’escalier de départ et nous permet de le déplacer à notre guise afin de tester les différents éléments que nous aurions pu disposer dans notre labyrinthe.
On remarque également que les boutons « TEST » et « SAVE » ont été remplacés par un bouton « MODIFY ». Ce bouton nous permet de supprimer notre personnage et de reprendre l’édition de notre labyrinthe.



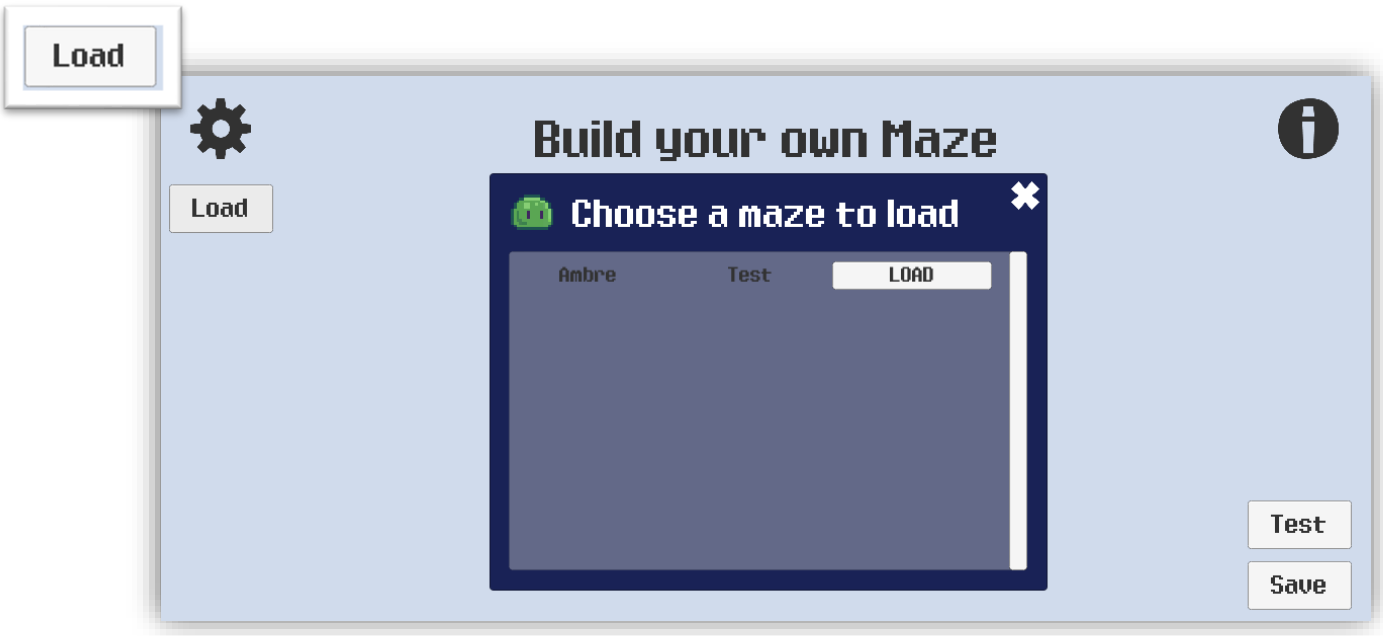
Le **bouton « SAVE »** comme le bouton « TEST » n’est accessible que s’il y a au moins un escalier de départ et un escalier d’arrivée sur le plateau.
Une fois cela fait cette fenêtre apparait si le labyrinthe est correct. Elle nous demande un pseudo pour identifier le créateur et un nom pour identifier le labyrinthe.

Il reste néanmoins quelques problèmes de gestion des obstacles que nous n’avons pas eu le temps de corriger. Cela crée des problèmes dans la vérification qui a pour résultat de nous signaler une erreur alors même que le labyrinthe est en réalité totalement faisable.

Tant que les deux champs ne sont pas remplis cette fenêtre apparait pour nous le signaler.

Une fois que les deux champs ont été correctement remplis cette fenêtre apparaît pour informer que le labyrinthe a été correctement enregistré.
Une fois qu'on appuie sur le bouton « OK » nous pouvons continuer de modifier notre labyrinthe de la même façon, le re-tester, le réenregistrer, …

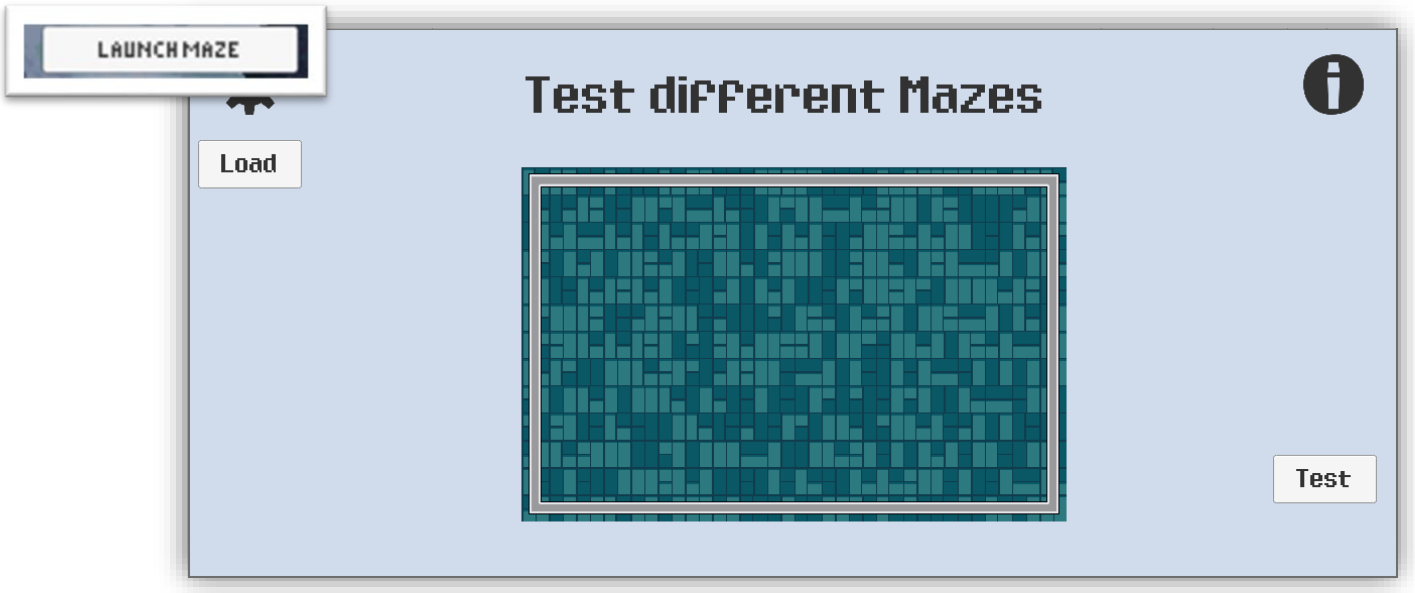
S'il n'existe aucun chemin permettant de relier le départ à l'arrivée, cette fenêtre apparaît pour nous le signaler. Après avoir appuyé sur le bouton « OK » nous pouvons procéder aux modifications permettant de corriger cela.



Le **bouton « LOAD »** nous permet d'afficher la liste des labyrinthes enregistrés afin de charger celui qu'on veut éditer, tester, …

Par manque de temps cette fonctionnalité n'a pas pu être achevée, elle n'est donc pas encore fonctionnelle.

3.2 – L’interface de tests de niveaux



Cette interface permet de charger des labyrinthes déjà créés afin de les tester.

Comme dit plus haut la fonctionnalité de chargement de labyrinthes n’a pas encore été finalisée donc on ne pourra malheureusement rien observer visuellement.

4 - Détails de code de certaines fonctionnalités

4.1 – Sauvegarde d'un labyrinthe

```
1 référence
public IEnumerator Save(string levelTitle, string creatorName, string jsonString)
{
    WWWForm formData = new WWWForm();
    formData.AddField("name", levelTitle);
    formData.AddField("creator", creatorName);
    formData.AddField("maze_data", jsonString);

    UnityWebRequest www = UnityWebRequest.Post(baseURL, formData);
    yield return www.SendWebRequest();

    if(www.result != UnityWebRequest.Result.Success)
    {
        Debug.Log(www.error);
    } else
    {
        AddUrlId(levelTitle);
    }
}
```

Voici la fonction principale permettant de sauvegarder un labyrinthe en base de données. Il nous suffit de récupérer un titre de labyrinthe, un nom de créateur et le json correspondant aux pièces placées ainsi que leur emplacement.

```
foreach (string myObject in nameOfObject) {
    foreach (GameObject newObject in GameObject.FindGameObjectsWithTag(myObject)) {
        if (newObject.transform.position.x > -10)
        {
            i = 0;
            ItemToSave tmp = new ItemToSave();
            foreach(string type in nameOfObject) {
                if(type == myObject)
                {
                    tmp.type = (typeOfObject)i;
                }
                i++;
            }
            tmp.pos = newObject.transform.position;
            items.Add(tmp);
        }
    }
}
```


Pour créer le Json, on remplit tout d'abord une liste avec le type et la position de chaque pièce.

```
[Serializable]
5 références
public class ItemToSave {
    public typeOfObject type;
    public Vector3 pos;
}
```

Chaque pièce appartient à cette classe ItemToSave qui permet de la définir par rapport à son type et à sa position.
Le type est un enum comportant tous les types de pièces possibles.

```
i = 0;
foreach (ItemToSave item in items)
{
    if (i == 0)
    {
        Json = "[" + JsonUtility.ToJson(item);
    }
    else if (i == items.Count - 1)
    {
        Json = Json + "," + JsonUtility.ToJson(item) + "]";
    } else
    {
        Json = Json + "," + JsonUtility.ToJson(item);
    }
    i++;
}

StartCoroutine(saveLoadMaze.Save(inputInfosMaze.text, inputNickname.text, Json));
items.Clear();
```

C'est ensuite ici que nous créons le Json en concaténant correctement chaque élément de la liste items.

Nous finissons par appeler notre coroutine qui va se charger de sauvegarder notre labyrinthe avec les informations que nous lui avons récupéré (le nom du créateur, le nom du labyrinthe et le json).

Une fois un labyrinthe enregistré nous pouvons observer le résultat dans PhpMyAdmin pour voir que tout a bien été enregistré :

id	name	creator	created_at	updated_at	maze_data
1	b	a	2021-05-21 15:56:21	2021-05-21 15:56:21	[{"type":0,"pos":{"x":-2.5,"y":-2.5,"z":0.0}},{"ty...
2	b2	a2	2021-05-21 15:56:32	2021-05-21 15:56:32	[{"type":0,"pos":{"x":-2.5,"y":-2.5,"z":0.0}},{"ty...
3	b3	a3	2021-05-21 15:56:42	2021-05-21 15:56:42	[{"type":0,"pos":{"x":-2.5,"y":-2.5,"z":0.0}},{"ty...
4	b4	a4	2021-05-21 15:57:09	2021-05-21 15:57:09	[{"type":0,"pos":{"x":-2.5,"y":-2.5,"z":0.0}},{"ty...
5	b4	a4	2021-05-21 15:57:19	2021-05-21 15:57:19	[{"type":0,"pos":{"x":-2.5,"y":-2.5,"z":0.0}},{"ty...

4.2 – Chargement d'un labyrinthe

```
0 références
public IEnumerator Load(string levelTitle) {
    using(UnityWebRequest mazeInfosRequest = UnityWebRequest.Get(GetUrl(levelTitle)))
    {
        string[] mazesInfos;

        yield return mazeInfosRequest.SendWebRequest();
        if (mazeInfosRequest.result == UnityWebRequest.Result.ConnectionError)
        {
            Debug.LogError(mazeInfosRequest.error);
        }
        else
        {
            string rawInfos = mazeInfosRequest.downloadHandler.text;
            mazesInfos = rawInfos.Split(new string[] { "}," }, StringSplitOptions.None);

            for (int i = 0, counter = 0; i < mazesInfos.Length; i++, counter++)
            {
                /* Getting the position of the object */
                float x = getFloat(mazesInfos[i], "x", ":");
                mazesInfos[i] = mazesInfos[i].Substring(mazesInfos[i].IndexOf(','));
                float y = getFloat(mazesInfos[i], "y", ":");
                mazesInfos[i] = mazesInfos[i].Substring(mazesInfos[i].IndexOf(','));
                float z = 0;

                i = i + 1;
                int position = mazesInfos[i].IndexOf(':') + 1;

                // TODO : creation de l'objet en fonction de son type et de sa position à l'aide d'un switch
                // switch (mazesInfos[i][position]) {
                //         case 0 :
                //             Instantiate(Angle1, transform.position, Quaternion.identity);
                //         [...]
                // }
            }
        }
    }
}
```

Voici la fonction principale permettant de charger un labyrinthe depuis la base de données. Il nous suffit d'envoyer le titre du labyrinthe pour récupérer le json que nous allons ensuite décomposer pour récupérer les pièces et leurs emplacements.

C'est malheureusement la fonction qui n'est pas finalisée, il manque la partie de recréation des pièces que nous n'avons pas eu le temps de terminer.

4.3 – Pathfinding et Algorithme A*

Ça a sans doute été la partie la plus compliqué et celle qui nous a fait perdre le plus de temps. Effectivement, de base nous avions choisi la solution dites de facilité qui était d'utiliser la librairie A* fournit par Unity et qui calculait d'elle-même le pathfinding.
Cependant, nous avons finalement décidé de le recoder entièrement afin d'avoir la main mise sur tous les détails de ce code et pouvoir ensuite aller plus loin.

Dans un premier temps nous avons créé une classe MazeGrid nous permettant de créer la zone dans laquelle l'IA allait rechercher son pathfinding.

Ensuite nous avons besoin d'une classe AIPathNode permettant de définir chaque case comme un nœud à part entière.

```
1 référence
public AIPathNode(MazeGrid<AIPathNode> _grid, int _x, int _y)
{
    this.grid = _grid;
    this.x = _x;
    this.y = _y;
    isWalkable = true;
}
```

Chaque nœud est défini sur la grille par rapport à ses positions en x et en y.
Nous avons ajouté un boolean isWalkable afin de différencier les chemins sur lequel l'IA pouvait passer des murs.

```
1 référence
private List<AIPathNode> GetNeighbourList(AIPathNode currentNode)
{
    List<AIPathNode> neighbourList = new List<AIPathNode>();

    if (currentNode.x - 1 >= 1)
    {
        // Left
        neighbourList.Add(GetNode(currentNode.x - 1, currentNode.y));
        // Left Down
        if (currentNode.y - 1 >= 1) neighbourList.Add(GetNode(currentNode.x - 1, currentNode.y - 1));
        // Left UP
        if (currentNode.y + 1 < grid.GetHeight()) neighbourList.Add(GetNode(currentNode.x - 1, currentNode.y + 1));
    }
    if (currentNode.x + 1 < grid.GetWidth())
    {
        // Right
        neighbourList.Add(GetNode(currentNode.x + 1, currentNode.y));
        // Right Down
        if (currentNode.y - 1 >= 1) neighbourList.Add(GetNode(currentNode.x + 1, currentNode.y - 1));
        // Right Up
        if (currentNode.y + 1 < grid.GetHeight()) neighbourList.Add(GetNode(currentNode.x + 1, currentNode.y + 1));
    }
    // Down
    if (currentNode.y - 1 >= 1) neighbourList.Add(GetNode(currentNode.x, currentNode.y - 1));
    // Up
    if (currentNode.y + 1 < grid.GetHeight()) neighbourList.Add(GetNode(currentNode.x, currentNode.y + 1));

    return neighbourList;
}
```

Nous avons une fonction GetNeighbourList qui nous permet de récupérer tous les nœuds voisins, sur lequel l'IA peut marcher, à notre nœud courant. Cela va nous permettre de parcourir les différentes possibilités en calculant le coût de trajet jusqu'à atteindre notre arrivée et ainsi sélectionner le chemin le moins coûteux et donc le plus court.

Nous allons calculer cela grâce, entre autres, à notre fonction FindPath.

```
while(openList.Count > 0)
{
    AIPathNode currentNode = GetLowestFCostNode(openList);
    if(currentNode == endNode)
    {
        return CalculatePath(endNode);
    }

    openList.Remove(currentNode);
    closedList.Add(currentNode);

    foreach(AIPathNode neighbourNode in GetNeighbourList(currentNode))
    {
        if (closedList.Contains(neighbourNode)) continue;
        //Debug.Log("x : " + neighbourNode.x + " y : " + neighbourNode.y + " " + neighbourNode.isWalkable);
        if(!neighbourNode.isWalkable)
        {
            closedList.Add(neighbourNode);
            continue;
        }

        int tentativeGCost = currentNode.gCost + CalculateDistanceCost(currentNode, neighbourNode);
        if(tentativeGCost < neighbourNode.gCost)
        {
            neighbourNode.cameFromNode = currentNode;
            neighbourNode.gCost = tentativeGCost;
            neighbourNode.hCost = CalculateDistanceCost(neighbourNode, endNode);
            neighbourNode.CalculateFCost();

            if(!openList.Contains(neighbourNode))
            {
                openList.Add(neighbourNode);
            }
        }
    }
}
```

C’est dans cette boucle que se crée notre pathfinding. Nous parcourons les différents nœuds jusqu’à atteindre l’arrivée et ensuite grâce aux nœuds récupérés nous calculons le pathfinding.

```
1 référence
private List<AIPathNode> CalculatePath(AIPathNode endNode)
{
    List<AIPathNode> path = new List<AIPathNode>();
    path.Add(endNode);
    AIPathNode currentNode = endNode;
    while(currentNode.cameFromNode != null)
    {
        path.Add(currentNode.cameFromNode);
        currentNode = currentNode.cameFromNode;
    }
    path.Reverse();
    return path;
}
```

Nous partons de l’arrivée pour remonter jusqu’au point de départ et récupérer ainsi tous les nœuds qui composeront notre pathfinding final.

```
List<AIPathNode> path = pathfinding.FindPath((int)startInstance.transform.position.x + 9,
                                             (int)startInstance.transform.position.y + 7,
                                             (int)endInstance.transform.position.x + 9,
                                             (int)endInstance.transform.position.y + 7);
```

Nous n'avons plus qu'à utiliser notre méthode de la façon suivante afin de trouver le chemin permettant de rejoindre notre arrivée.

4.4 – Design Pattern : Le singleton

```
2 références
public static Singleton _instance { get; private set; }

// Data that will always be unique
public MazeGrid<AIPathNode> mazeGrid;

Message Unity | 0 références
private void Awake()
{
    if (_instance == null)
    {
        _instance = this;
        DontDestroyOnLoad(gameObject);
    }
    else
        Destroy(gameObject);
}
}
```

Nous avons utilisé un singleton afin de nous assurer de ne travailler toujours que sur une grille dans notre recherche de pathfinding.

Conclusion

Nous sommes contents d'avoir réalisé ce projet. Il nous a demandé beaucoup d'investissement tout au long de ces 2 mois et nous a permit de réutiliser une grande partie des notions que nous avons pu étudier tout au long de nos études.
Il nous a demandé un gros travail de formation en autodidacte ce qui est quelque chose de très courant dans notre milieu au niveau professionnel. Cela a donc été très enrichissant.

Nous sommes néanmoins déçus des quelques fonctionnalités que nous n'avons pas eu le temps de finaliser. Nous avons perdu du temps par manque de connaissances des différentes technologies. Cela nous a apprit à être réactif et à savoir prendre des décisions comme mettre de côtés certaines fonctionnalités pour en développer d'autres.

Nous sommes fiers de notre logiciel que nous avons réussi à coder en si peu de temps et avec si peu de connaissances dans les technologies utilisées. Ça aura été finalement une très bonne expérience tant au niveau professionnel que personnel.