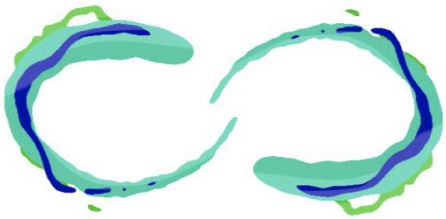
Rapport de projet - Projet S2

EPITA 2022

Sacha Reggiani
Clément Fossati
Aboubakar Charf
Eytan Benhamou



CEAS

Sommaires

Cahier des charges - Projet S2

EPITA 2022

- Introduction :

Dans ce cahier des charges nous allons présenter notre jeu au travers divers aspects permettant d'expliquer clairement pourquoi nous avons pris ces décisions. Le but de ce projet est de créer un jeux-vidéo type Bomberman, pour comprendre ce choix il faut revenir sur nos compétences actuelles. Au début de ce projet on s'est dit que faire un jeu ayant à la fois du multijoueur et une intelligence artificielle était particulièrement ardu pour de parfait novice en la matière. On a donc préféré s'orienter vers une stratégie plus efficace, nous allons donc accorder plus d'importance à un produit fini, jouable, sans bug, avec plusieurs fonctionnalités plutôt qu'à un projet trop ambitieux ne pouvant être réaliser dans les temps. Nous nous sommes fixés comme objectifs de produire un jeu au graphisme agréable, au gameplay dynamique et intuitif, au level design bien travaillé et enfin à la bande sonore efficace et agréable. Nous allons donc nous concentrer sur ces points et laisser de côté les autres, cela nous permettra d'avoir un jeu complet à la fin du projet.

Ce projet commencera tout juste, nous avons donc déjà effectuer de nombreuses recherches, et nous allons nous concentrer principalement sur nos domaines d'expertises, il va falloir donc que l'on se familiarise avec le monde de la programmation, des mathématiques et de l'informatique. Nous allons donc commencer notre jeu en utilisant un langage orienté objet, plus simple à programmer pour un débutant. Ce choix nous permettra de nous familiariser rapidement avec la programmation, grâce à nos connaissances sur le sujet.

Nous allons donc utiliser la plateforme Unity pour créer le jeu, ce logiciel nous permet de créer notre propre jeu avec les fonctionnalités dont nous avons besoin pour monter le projet. Unity est un programme open source permettant de créer des jeux multiplateformes.

1.1 Objet d'étude :

L'objet de cette étude est avant tout d'apprendre comment travailler en équipe sur un projet de cette envergure. En effet c'est la première fois que nous nous attelons à un projet si long, en effet la création d'un jeux-vidéo nécessite beaucoup de temps, en plus de demander un travail de recherche intensive sur une grande variétés de sujets.

Ce sera aussi la première fois que nous utiliserons GitHub ou GitLab, voire GitKraken, il faut d'abord évaluer quel utilitaire

sera le plus intéressant à utiliser pour ensuite ce fixé dessus et démarrer le projet.

1.2 État de l’art :

Le tout premier jeu Bomberman est sorti en 1983 sur ZSX et ZX Spectrum. À l'époque, il s'agissait d'un petit bonhomme avec un chapeau. Il est apparu sur MSX en 1985 puis sur la NES en 1987 sous le nom de Bomberman. Il s’agit d’un jeu ayant pour genre action et labyrinthe, ou on y incarne bomberman un petit personnage évoluant dans un niveau faisant penser à un labyrinthe ou vous devez avancer pour tuer vos ennemis, pour ce faire vous pouvez poser des bombes, explosant après un cours délai détruisant aussi les blocs se trouvant dans le rayon d’explosions.

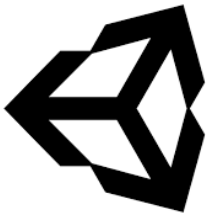
La série Bomberman a connu beaucoup de clones mais assez peu de renouveau au sein de la série en elle-même, il-y-a bien sûr eu des modifications sur les différentes versions comme l’apparition de nouveaux “power-up” ou encore de mode de jeu différent du mode de jeu classique, à savoir tuez les ennemis pour gagner.

Mais le clone le plus marquant est probablement Bomb-it, c’est un jeu flash sur navigateur le principe est simple c’est un Bomberman sauf que l’avantage de celui-ci c’est qu’il est très facile d’accès, il est gratuit, bien finis, et à quand même quelques nouveautés qu’on ne retrouve pas sur le jeu de

base comme l'ajout des véhicules ayant chacun une particularité différente.

- Technologies :

  3.1 Outils, Logiciels et Framework



Unity est un moteur de jeu open source, de plus Unity permet de partager facilement des ressources graphiques et sonores. Il permet aussi un rendu accéléré et un chargement plus rapide.

Visual Studio est un IDE capable de gérer le C# qui va nous être utile pour les scripts sur Unity.

Git est un logiciel de gestion de versions décentralisé, qui permet de gérer les modifications d'un projet. Il est possible de créer des branches, de les fusionner, de les supprimer, etc.

Inno Setup est un logiciel libre permettant de créer des installateurs pour Windows.

Audacity est un logiciel de traitement sonore gratuit et Open Source qui permet d'enregistrer en direct, de copier-coller ou d'importer des fichiers audios depuis n'importe quelle source et de les mixer.

LaTeX est un langage et un système de composition de documents. Ce langage permet de créer des documents de qualité professionnelle, sans avoir à écrire une seule ligne de code.

GIMP, est un outil d'édition et de retouche d'image, diffusé sous la licence GPLv3 comme un logiciel gratuit et libre. Il est un bon équivalent à Photoshop.

photon

Photon est un utilitaire spécialisé dans la gestion du multijoueur, ce sera grâce à lui que nous créerons nos serveurs.

- Organisation du travail :

  3.1 Les différentes parties du projet

Pour des questions de répartitions des tâches, le projet a été découpé en grands domaines. En voici la

Liste :

1. Menu/Options
2. Site web
3. Multijoueur
4. Audio
5. IA

6. Gameplay
7. Items
8. Graphismes

3.2 Répartition des tâches

Pour nous répartir équitablement le travail, nous avons séparé le projet en plusieurs grandes parties que nous avons attribué à chacun selon nos points forts :

Répartition des tâches	**Sacha Reggiani**	**Clément Fossati**	**Aboubakar Charf**	**Eytan Benhamou**
Menu/Options			*	+
Multijoueur	*		+	
Audio	+			*
IA	+	+	*	
Gameplay	*		+	
Items		*		
Graphismes			*	+
Site Web	+	*		

Légende :

* : Responsable

+ : Aide directe

Bien sûr certaines taches seront plus difficiles que d'autres donc nous nous aideront mutuellement pour pouvoir les accomplir efficacement et dans les temps.

3.3 Planning :

Objectif 1er soutenance (semaine du 7 mars 2022) :

Pour cette première soutenance les objectifs sont tout d'abord dans un premier temps de démarrer le projet. Les fonctionnalités attendues sont :

- Avoir une première carte pour pouvoir tester le jeu, pas nécessairement avec d'excellent graphismes ni le style graphique du rendu final.

- Avoir le cœur du jeu, les déplacements, les collisions, le système de bombes que le joueur peut poser.

- Commencer le multijoueur pour avoir au moins une base dès le début du projet pour ne pas avoir à gérer deux versions jeu, une en solo, et une en multijoueur car il pourrait y avoir du retard sur la version en solo.

- Avoir un menu opérationnel qui permet de lancer le jeu et le quitter.

-Avoir un site en ligne.

Objectif 2ème soutenance (semaine du 25 avril 2022) :

Pour cette seconde soutenance l'objectif principale est d'avoir quelque chose de jouable sur lequel on peut lancer une partie, ce ne sera évidemment pas le rendu final, il manquera surement de nombreuses fonctionnalités. Les fonctionnalités attendues sont :

- Avoir une ou deux cartes avec le style graphique souhaité qui sont terminées.

- Le cœur du jeu est fini, avec toutes les différentes idées proposer au début du projet, quelque power-up seront ajoutés dans les parties.

- Le multijoueur doit être stable et fonctionnel.

- Le menu/option doit être avancé notamment sur la partie configuration des touches, gestion du volume et modification de la résolution.

- Une musique devrez aussi être présente dans le menu ou les parties.

- L'intelligence artificielle devrez être commencé bien que très limité dans ses choix et actions.

-Avoir un site présentable avec plusieurs pages et une mise en page correcte

Objectif 3ème soutenance (semaine du 6 juin 2022) :

Si tout c’est bien passer à cette soutenance, le jeu devrait être fini. Il doit comporter un menu/option fonctionnel avec un bouton jouer qui vous connecte directement au serveur de Photon pour jouer en multijoueur comme en solo (pour jouer en solo il faudra quand même être connecté à internet). Le multijoueur sera fini, on pourra se connecter pour jouer à 20 joueurs maximum (limite imposée pas photon) et choisir d’inclure ou non une ou plusieurs intelligences artificielles. Le gameplay sera fini avec tous les power-up et l’IA doit avoir une difficulté faible. La musique et les bruitages doivent aussi être fini.

- Gameplay :

  4.1 Objectif d’une partie

Le gameplay de notre Bomberman like est assez simple. Le joueur dirige un personnage qui doit éliminer tous les ennemis qui se présentent à lui grâce à ses bombes. Mais des blocs le séparent d'eux, il faudra donc les casser pour pouvoir avancer. Ces blocs auront une chance de lâcher des items qui pourront être des bonus temporaires ou des améliorations qui resteront jusqu'à la fin de la partie. Une partie se termine quand votre personnage meurt ou quand tous les ennemis sont morts.

Conclusion :

En conclusion, au travers des divers points abordés ci-dessus, nous espérons pouvoir créer un bon jeu, qui soit amusant, sans bug, bien fini et surtout rendu dans les temps. De plus nous allons créer des deadlines afin de mieux structurer notre travail et d'augmenter la productivité au sein du groupe. Cependant, nous ne prétendons pas avoir inventé un style de jeu nouveau ou révolutionnaire, mais nous comptons bien faire de ce jeu la meilleure expérience possible, qui je l'espère saura vous plaire.

Voilà pour le cahier des charges, comme vous avez pu le constatez il a été restitué tel qu'il était lors de sa création, soit au tout début du projet. Cela permettra dans la suite de ce rapport de pouvoir comparer les éléments actuels avec ce que nous imaginions.

Dans la suite de ce rapport nous allons énumérer et détailler tout ce qui s'est passé au sein du groupe lors de la réalisation de ce projet. Pour cela chaque membre du groupe va expliquer et montrer ses avancés personnelles lors des diverses étapes qui ont constitué la création du jeu que nous devions faire. Cela peut paraitre surprenant que chaque membre explique sa partie surtout sur un projet de groupe, mais c'est tout à fait normal car nous avons préféré se répartir les tâches et laisser chaque membre avancer à son rythme permettant ainsi d'éviter des problèmes de disponibilité pour chaque membre.

Sacha Reggiani :

1er Soutenance :

2.Gameplay et multijoueur :

2.1. Le Commencement du projet :

Pour le début du projet après avoir créé le repo sur GitHub puis l'avoir cloné dans un dossier sur le bureau via l'installation de git pour manipuler l'environnement Windows comme s'il s'agissait d'un terminal linux. Le travail pouvait commencer, tout d'abord il y avait une pléthore de guide sur internet sur comment réaliser un bomber man, le seul qui ait réellement servi est celui de Brackeys, après avoir vu le tutoriel en entier, j'ai vite compris que le projet allait prendre beaucoup plus de temps que prévu. Il fallait tout d'abord comprendre un nouveau système de création de niveau, à savoir les grilles (ou tilemaps).

2.2 Les Tilemaps :

C'est sur cette partie que le projet a véritablement démarré, après être allait voir la vidéo de Brackeys sur les Tilemaps,

tout était plus claire mais pour ce faire il fallait des sprites basique de murs, de sol, d'obstacle etc.…

Étant au début du projet et voulant avancer rapidement sans être encombré par la partie graphique, il a fallu importer des sprites venant du Unity asset store, qui était d'ailleurs assez mal adapté à la vue du dessus. L'utilisation d'une tilemap était assez facile à prendre en main malgré le fait que les sprites était initialement conçu pour une vue du dessus incliné vers l'avant.

Une fois la toute première carte finie avec les différentes couches pour le fond et le gameplay, il fallait une vraie carte pour premièrement jouer dessus et faire différents tests et aussi pour la soutenance avoir une carte suffisamment belle pour ne paraitre être une prémices de ce qu'on aurait pu faire pour la première soutenance.

2.3 Les Bombes :

Pour ce qui est des bombes au départ je voulais pouvoir les poser à la souris pour voir déjà si elles marcher et pouvoir faire d'autres test que nous ne pourrions réaliser avec un personnage qui les pose, comme regarder ce qui se passe si on pose une bombe en dehors des murs. Pour créer les bombes à la base j'ai encore une fois importé un modèle déjà

existant pour pouvoir tester le code pour voir si tout fonctionne bien. Il y a plusieurs scripts associés aux bombes, tout d'abord il y en a un qui regarde le préfabriquée bombes et qui le détruit au bout de 2 secondes d'existence.

```
public float countdown = 2f;

void Update () {
    countdown -= Time.deltaTime;

    if (countdown <= 0f)
    {
        FindObjectOfType<MapDestroyer>().Explode(transform.position);
        Destroy(gameObject);
    }
}
```

Vient ensuite celui qui permet d'instancier les bombes via l'emplacement de la souris sur la grille

```
void Update () {
    if (Input.GetMouseButtonDown(0))
    {
        Vector3 worldPos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        Vector3Int cell = tilemap.WorldToCell(worldPos);
        Vector3 cellCenterPos = tilemap.GetCellCenterWorld(cell);

        Instantiate(bombPrefab, cellCenterPos, Quaternion.identity);
    }
```

Les différentes méthodes que nous voyons la sont assez complexe à appréhender mais seront expliqué plus tard dans la partie script.

On peut d'ailleurs voir dans le premier script lié aux bombes on peut lire map destroyer qui est le dernier script relié aux bombes du jeu, il indique simplement à l'objet bombes de lancer l'animation associé sur plusieurs cases à la suite tout en détruisant les différents obstacles qui dressent sur son chemin (évidemment les murs ne sont pas pris en compte dans l'explosion, seuls les objets avec le tag destructible peuvent être détruit).

```
bool ExplodeCell (Vector3Int cell)
{
    Tile tile = tilemap.GetTile<Tile>(cell);

    if (tile == wallTile)
    {
        return false;
    }

    if (tile == destructibleTile)
    {
        tilemap.SetTile(cell, null);
    }

    Vector3 pos = tilemap.GetCellCenterWorld(cell);
    Instantiate(explosionPrefab, pos, Quaternion.identity);

    return true;
}
```

Cette méthode comme dit juste au-dessus détruit les blocs, il y a donc une autre fonction qui permet d'appeler cette méthode sur toutes les cases souhaitées, actuellement la façon de le faire est assez peu ergonomique, cela ne fait que répéter le script sur toutes les cases en forme de croix toute

autour. Le grand problème de cette méthode est donc que ça ne laisse pas la possibilité d'ajuster la taille de l'explosion, ce qui sera problématique pour plus tard lorsque ne nous ajouterons les power-up.

2.4 Le Personnage :

C'est la première partie ou il y a des repères, c'est quelque chose d'assez facile à prendre en main et intuitif, on a eu les cours NTS ou justement le personnage prenez une grande partie du TP à faire, donc pour ce qui en est actuellement, le visuel du personnage est loin d'être achevé (c'est une capsule bleue) maintenant mais au vu du temps restant on a préféré se focaliser sur tout le reste à savoir le gameplay, le multijoueur et le menu. Donc pour ce qui est du personnage en lui-même on a quelque chose de très simple mais de fonctionnel, en termes de déplacement le personnage se déplace via un rigidBody sur lequel on applique une force. Pour tout ce qui est collisions avec les murs c'est directement gérer pas la tilemap gameplay où il y a un rigidBody homogène à tous les murs.

```
void FixedUpdate()
{
    rb.MovePosition(rb.position + movement * moveSpeed * Time.fixedDeltaTime);
}
```

Dans ce screen on voit aussi la méthode est défini dans FixedUpdate, cela permet ne pas dépendre du nombre de FPS de l'utilisateur (ce qui est le cas de la fonction Update), donc FixedUpdate sera constant dans son appelle ce qui le mieux pour gérer le déplacement d'un personnage. Juste au-dessus rb est le rigidBody du joueur et movement est défini en fonction de si le joueur appui sur ses touches de déplacements

```
movement.x = Input.GetAxisRaw("Horizontal");
movement.y = Input.GetAxisRaw("Vertical");
```

3. Le Multijoueur :

3.1 Loading :

Avec l'ajout du multijoueur dans le projet, certaines fonctionnalités ont disparu comme le fait de pouvoir poser des bombes directement sur le personnage mais pas de panique toutes ces erreurs ont vite été réparées, mais penchons-nous plutôt sur comment le multijoueur a été implémenté.

La majeure aide pour la confection du multijoueur fut le guide de BlackThornProd qui montre comment ajouté cette fonctionnalité en 9 étapes assez simple. Pour le multijoueur on s'est tourné vers les serveurs de Photon, qui sont une très bonne méthode car gratuit, fiable et marche bien avec Unity. Donc après avoir créé le serveur Photon et importé l'asset PUN2 du Unity asset store, tout été prêt.

Maintenant on a créé une nouvelle scène qui permet de se connecter au serveur et de lancer une nouvelle scène qui sera là où on pourra créer un lobby ou en rejoindre un.

```
public class ConnectToServer : MonoBehaviourPunCallbacks
{
    void Start()
    {
        PhotonNetwork.ConnectUsingSettings();
    }
```

Dans le code juste au-dessus on voit le script associé à la scène de chargement, à la place de l'habituelle MonoBehaviour cette fois on a MonoBehaviourPunCallbacks ce qui va rappeler le serveur jusqu'à être connecté sur celui-ci. La fonction start launch justement cette connexion.

```
public override void OnConnectedToMaster()
{
    PhotonNetwork.JoinLobby();
}

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

Le code du dessus est la suite directe au code d'avant, une fois connecté il va rejoindre le lobby (le serveur dédié que j'ai créé), puis quand on est connecté on va changer de scène pour enfin pouvoir créer ou rejoindre des parties.

3.2 Lobby :

Voilà la scène du Lobby, pour l'instant c'est très épuré mais ce n'est pas encore le rendu final.

Sur cette scène on peut observer deux boutons create lobby et join lobby qui respectivement prennent les contenus des champs d'écritures à droite et appliquent l'effet associé.

```
public class CreateAndJoinRooms : MonoBehaviourPunCallbacks
{
    public InputField createInput;
    public InputField joinInput;

    public void CreateRoom()
    {
        PhotonNetwork.CreateRoom(createInput.text);
    }

    public void JoinRoom()
    {
        PhotonNetwork.JoinRoom(joinInput.text);
    }

    public override void OnJoinedRoom()
    {
        PhotonNetwork.LoadLevel("PlayTest");
    }
}
```

Voilà le script de la scène on peut voir les deux méthode CreateRoom et JoinRoom qui respectivement crée une nouvelle partie et rejoignent une partie déjà existante. Vous l'aurez probablement compris, mais le système de création de partie repose juste sur un nom pour rejoindre une partie. D'ailleurs pour jouer en solo il sera donc impératif de se connecter d'abord au serveur

4.Menu/Options :

4.1 Menu :

En ce qui concerne le menu, on s'est aidé des vidéos de Brackeys, qui nous explique comment implémenter un simple menu, que l'on étoffera pour la seconde soutenance.

Comme on peut le voir sur le screen ci-dessous, il y a pour le moment un bouton Play fonctionnel qui nous permet d'arriver sur la loading scene.

Pour ce faire on a créé un nouveau panel qui servira de background, sur lequel on a importé une

Image de fond, et contrasté les couleurs de façon à ce que l'on puisse avoir un bon fond pour le menu.


```
public void play()
{
    SceneManager.LoadScene("Loading");
}
```

Pour ce qui est du bouton play, on a créé la fonction play() qui permet donc d'accéder au lobby.

Pour le bouton exit, on a créé la fonction exit() qui permet donc de fermer le jeu.

2nd Soutenance :

Problème du multijoueur :

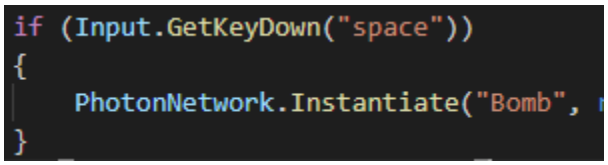
En reprenant le projet depuis là où il en était je me suis heurté à un gros problème. Le multijoueur, le problème avec le multijoueur depuis la dernière fois c'est que les bombes, les explosions et les destructions de blocs n'étaient pas reliées, c'est à dire qu'un joueur pouvait faire ce qu'il voulait, ça n'avait pas de conséquence pour les autres joueurs.

Le multijoueur marchait car on pouvait très bien voir tout le monde se déplacer sur la carte mais aucune possibilité d'interagir. Donc ce fut le problème majeur, le tout premier qu'il fallait résoudre avant de pouvoir faire quelques autres implémentations, Mais le défi était de taille, c'était probablement l'étape la plus dure car le groupe n'ayant aucune expertise en multijoueur il fallait commencer à zéro.

Heureusement il y avait quelques tutoriels bien ficelés et détaillés qui pouvaient aider à la résolution de ce problème. Malheureusement il y avait très peu d'information

concernant des maps modulaires qui peuvent être modifié au cours du temps.

C’est donc ainsi que commença la construction d’un multijoueur fonctionnel. Tout d’abord je me suis penché sur la documentation officielle de Photon (qui est notre logiciel qui permet d’héberger des serveurs, sur lesquelles on peut connecter tous les joueurs), et j’y ai rapidement la bonne méthode, il fallait juste ajouter PhotonNetwork.(ma fonction) afin d’envoyer sur le serveur les bombes.



Là on pourrait se dire que s’était quand même plutôt simple compte tenu des paragraphes ci- dessus, sauf que les problèmes arrivent maintenant. Une fois cet ajout fait j’ai lancé le jeu et remarqué qu’il y avait pas mal d’erreur, je me suis renseigné, j’ai trouvé et j’ai corrigé.

Le problème était que chaque joueur devez faire disparaitre sa version, pour les bombes ou les explosions, il fallait donc ajouter un composant sur les bombes et les explosions afin de savoir qui a quoi.

```
PhotonView view;

void Start ()
{
    view = GetComponent<PhotonView>();
}
```

```
if (view.IsMine)
{
    PhotonNetwork.Destroy(gameObject);
}
```

Une fois ceci fait, la majorité des erreurs avaient disparues, mais le problème de taille qui s'est imposé juste après a été et est encore le plus difficile de tout le projet. Le multijoueur.

Ce problème si difficile à résoudre vient du fait qu'au cours du partie, tout semble se dérouler comme prévu lorsque qu'une animation de bombe s'effectue deux fois, en réappuyant sur la touche pour poser une bombe je me rends compte que là il n'y a plus qu'une seule animation je réessaye plusieurs fois et à chaque fois j'obtiens un résultat différent sans aucune explication logique. J'essaye en marchant, en ne bougeant plus en posant deux bombes côte-à-côte, mais rien n'y fait il n'y a rien qui puisse expliquer ce phénomène bizarre. Alors c'est naturellement que le lance la session sur différent ordinateurs afin de comprendre ce qui se passe, et là je constate que lorsqu'il n'y a qu'une explosion les destructions occasionnées par la bombe ne sont pas retransmis pour tous les joueurs. Et lorsque qu'il y en a

plusieurs les destructions sont bien réparties pour tous les joueurs, et un autre fait bizarre c'est que plus il y a de joueurs connectés en même temps plus il y a d'explosions lorsqu'elles sont multiples (elles sont d'ailleurs proportionnelles au nombre de joueurs).

A ce moment ça ne fait aucun doute que le problème vient de mon script, mais pour toute la partie multijoueur il n'y probablement aucune erreur car le code utilisé revient très souvent dans de nombreux tutoriels ainsi que sur la documentation officielle de Photon qui est notre serveur

```
void Start()
{
    PhotonNetwork.ConnectUsingSettings();
}

public override void OnConnectedToMaster()
{

    PhotonNetwork.JoinLobby();
}

public override void OnJoinedLobby()
{

    SceneManager.LoadScene("Lobby");
}
```

```
public InputField createInput;
public InputField joinInput;


public void CreateRoom()
{

    PhotonNetwork.CreateRoom(createInput.text);
}

public void JoinRoom()
{

    PhotonNetwork.JoinRoom(joinInput.text);
}

public override void OnJoinedRoom()
{

    PhotonNetwork.LoadLevel("PlayTest");
}
```

Après avoir relu tout mon code pour pouvoir identifier d'où venait le problème, je pense enfin l'avoir trouvé, c'est uniquement dans l'appel des explosions qui sont en charges de les répartir autour de la bombe et de gérer le cas où il y aurait des blocs à détruire. Une fois ceci trouvé, il y a eu deux autres problèmes, je ne sais pas si le problème vient directement de mes fonctions qui gèrent l'explosion des

bombes ou si ça vient de l'appel aux fonctions qui gèrent les explosions des bombes.

```
void Update ()
{
    countdown -= Time.deltaTime;

    if (countdown <= 0f || explode)
    {
        FindObjectOfType<MapDestroyer>().Explode(transform.position, 2);
        if (view.IsMine)
        {
            PhotonNetwork.Destroy(gameObject);
        }
    }
}
```

Voilà après de nombreuses tentatives pour corriger cette fonction je n'ai pas réussi à trouver ce qui n'allait pas avec

Ajout de fonctionnalités utiles :

Mort :

Dans cette section nous aborderons plein d'ajout fait en vrac qui ne sont pas très long, nous les traiterons ici car ils ne méritent pas de sections à part mais constituent quand même des éléments importants dans la construction et amélioration du jeu.

Premièrement la mort. Pour l'instant le jeu marche à peu près comme convenu, sauf qu'il manque un élément clé dans les mécaniques basique du jeu, la mort.

Pour l'implémenter il aura simplement suffi de regarder si un joueur et sur le sprite d'une explosion et si oui on fait disparaitre le joueur tout simplement

```
PhotonView view;

void Start ()
{
    view = GetComponent<PhotonView>();
}

private void OnCollisionEnter2D(Collision2D other)
{
    if(other.gameObject.tag == "Explosion")
    {
        if(view.IsMine)
            {
                PhotonNetwork.Destroy(gameObject);
            }
    }
}
```

Pour effectuer cela on a créé un tout nouveau script qui ne sera dédié qu'à ça. Comme on peut le voir il y a un composant Photon view qui permet de garder en mémoire qui est qui et aussi surtout de pouvoir détruire l'objet sur chaque version du jeu pour chaque joueur. Sinon le reste parle de lui-même, on a juste rajouté une collision à la bombe ce qui permet de lancer la fonction OnCollisionEnter2D qui vérifie s'il y a une collision entre l'objet et autre chose, en l'occurrence si c'est une explosion on détruit le joueur représenté la par gameObject.

Chaines d'explosions :

Une mécanique emblématique des jeux bomber-man c'est évidemment les chaines d'explosions, qui permettent des réactions inattendues et souvent spectaculaires. Encore c'est assez simple à implémenter

```
if (countdown <= 0f || explode)
```

```
if(other.gameObject.tag == "Explosion")
{
    explode = true;
}
else
{
    explode = false;
}
```

On le voit ici on a juste à rajouter un booléen qui va dire si ça doit exploser ou non en plus du compteur classique. Pour modifier ce booléen on ajouter un composant déclencheur aux explosions ce qui va permettre de savoir si la bombe et sur une explosion et ainsi déclencher prématurément l'explosion de la bombe

Bombes solides :

Encore une mécanique phare des jeux de cette licence, les bombes qui agissent comme des murs, et oui sinon comment pourrions-nous bloquer nos adversaires pour les faire exploser. Pour cette mécanique c'était un petit peu plus dur car juste ajouter une composante solide aux bombes et une fausse bonne idée, pourquoi ? Pour la simple et bonne raison que si la bombe été instantanément solide et bien le joueur se ferait expulser en dehors de la bombe. Donc pour il faut

que la bombe n’ait pas de corps solide au moment de la pose mais dès que le joueur ne touche plus la bombe elle devient solide et agit comme un mur.

```
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.gameObject.tag == "Player" && IsIn)
    {
        collide.enabled = false;
        IsIn = false;
    }
    else
    {
        collide.enabled = true;
    }
```

Pour faire ça c’est à peu près pareil que pour les chaines d’explosions à savoir on check les collisions, même si là c’est un déclencheur qui n’est pas tangible au moment de la pose mais qui le devient dès que plus aucun joueur n’est dans la bombe.

<u>Réduction de la puissance des bombes :</u>

Pour l’instant les bombes étaient un petit peu trop puissantes, c’est à dire que s’il y avait un obstacle à détruire elle le détruisait mais continuer quand même après ce qui est une mauvaise chose, car on pourrait se croire en sécurité derrière des blocs mais la bombe explose au travers et nous tue. Pour remédier à cela :

```
bool ExplodeCell (Vector3Int cell)
{
    Tile tile = tilemap.GetTile<Tile>(cell);

    if (tile == wallTile)
    {
        return false;
    }

    if (tile == destructibleTile)
    {
        tilemap.SetTile(cell, null);
        Vector3 pos1 = tilemap.GetCellCenterWorld(cell);
        PhotonNetwork.Instantiate("Explosion", pos1, Quaternion.identity);
        return false;
    }

    Vector3 pos = tilemap.GetCellCenterWorld(cell);
    PhotonNetwork.Instantiate("Explosion", pos, Quaternion.identity);

    return true;
}
```

Pas besoin de tout comprendre ici juste à savoir que si c’est la tile est une destructibletile alors on return false ce qui bloqué la propagation des explosions.

Dernière Soutenance :

Tout d'abord en reprenant de là ou je m'étais arrêté j'ai voulu rapidement avancer et terminer des choses basiques et simple. C'est pour ça que je me suis dirigé le menu/options car c'était quelque chose de délaissé depuis la première soutenance. Donc pour améliorer les fonctionnalités qu'avais le menu pour les options j'ai commencé par regarder sur internet s'il y avait des tutoriels ou d'éventuels guides. Et effectivement il y'en avait une pléthore, mais le plus intéressant et celui de Brackeys un youtubeur spécialisé dans les guides pour débutant de Unity. En m'inspirant de son tutoriel j'ai finalement abouti à un résultat tout à fait convenable :

Ensuite il fallait bien avoir un vrai lobby pour accueillir les joueurs se connectant au serveur multijoueur de notre jeu. Pour se faire, encore une fois et comme à chaque fois j'ai cherché sur internet, et comme presque à chaque fois il y avait des solutions.

Ce nouveau lobby et assez différent mais en même temps assez proche de l'ancien, ce qui a changé par rapport à l'ancienne version c'est que premièrement quand vous cliquez sur le bouton Online vous êtes amené sur une nouvelle scène qui vous demande de rentrer votre pseudo, cela servira un petit peu plus tard. Ensuite une fois que vous avez rentré votre pseudo vous arrivez comme avant sur la même scène ou vous pouvez au choix créer une salle pour accueillir d'autre joueurs ou de rejoindre une salle déjà existante. Par rapport à avant la grosse modification c'est une qu'on a au choix créer ou rejoint une salle, à ce moment vous avez une interface ou il y a tous les joueurs connectés avec leurs pseudos et leurs personnages qu'ils ont sélectionnés, enfin il y a en haut le nom de la salle. A savoir que pour lancer une partie en multijoueur il est impératif d'être au minimum deux personnes connectés pour pouvoir lancer la partie et aussi que pour lancer une partie il n'y a que le créateur de la salle qui peux le faire (il peut aussi choisir la taille de la partie).

Le vrai gros travail pour cette soutenance finale a été le multijoueur, et surtout la correction de tous les problèmes liés au multijoueur. Après avoir revérifier tout mon code lié

au multijoueur, j'ai émis plusieurs hypothèses quant à la provenance des erreurs de synchronisations, et la plus convaincante était celle liée au tile, notamment dans un script ou il faut faire disparaitre des tiles pour tous les joueurs sauf que photon ne dispose d'aucune méthode permettant de supprimer des tiles pour tous les joueurs. C'est donc à partir de ce moment que je me suis penché sur la documentation de photon et que j'ai regardé tous leurs tutoriels sur le multijoueur, et après ça j'ai commencé à comprendre certaines choses. Donc la chose essentielle pour synchroniser tout sur le multijoueur de chaque personne connectée. Cette simple ligne avec une syntaxe particulière se mettant au-dessus d'une fonction peut être appelée par une méthode de Photon pouvant synchroniser une action pour toutes les personnes connectées au serveur

```
[PunRPC]
void PlaceBomb()
{
```

```
view.RPC("PlaceBomb", RpcTarget.All);
```

Après s'être heurté au problème le plus dur que j'ai rencontré depuis le début du projet, j'étais très content de moi car Photon n'est vraiment pas très compréhensible pour des débutants.

Ensuite j'ai ajouté deux nouvelles cartes sur lesquelles on peut jouer, malheureusement il n'y a pas de nouvelle texture les accompagnant mais elle reste quand même différente.

Pour finir un autre gros du travail qui m'a occupé pour la réalisation du projet c'est un mode seul, et oui car pour l'instant il n'y avait pas encore d'intelligence artificielle dans notre jeu. Donc à la place d'incorporer les IA directement au mode multijoueur on a préféré créer un mode seul contre des IA. Cette tâche qui aurait pu sembler être anodine et même triviale c'est en fait révéler beaucoup plus complexe et surtout beaucoup plus longue que prévue, pourquoi ? Pour la simple et bonne raison que tout notre jeu est basé sur le multijoueur et donc TOUT nos composants (scripts, personnages, cartes …) ont été pensé et créer autour du multijoueur, ce qui fait que transposer tout ça en version un joueur déconnecté des serveurs de Photon été impossible.

Mais pourquoi faire un mode un joueur alors que tout est penser pour le multijoueur ? Car l'intelligence artificielle est beaucoup plus simple à programmer en solo et est aussi beaucoup plus simple à déployer en solo. C'est donc pour ça qu'il a fallu refaire tous les scripts pour en avoir une version déconnectée ainsi que refaire toutes les cartes et tous les personnages, c'est vraiment quelque chose qui a nécessité un temps considérable.

Après avoir effectué toutes les choses ci-dessus il fallait encore s'occupait de la partie intelligence artificielle des bots qui allait apparaitre dans le mode solo, et encore une fois quelle tâche fastidieuse et complexe. Pour commencer comment coder une IA pour un bomberman ? En se documentant sur internet il y avait plusieurs pistes

intéressantes, mais la meilleure à quand même était une option très défensive, c'est à dire que le bot ce balade sur la carte en posant des bombes de façon aléatoires, mais dès qu'il y a une bombe le bot essaie à tout prix de sortir de la zone d'explosion.

C'est une assez bonne méthode vue que la complexité n'est pas trop élevée, sauf que le problème c'est que le bot n'évolue pas dans une grille ou une matrice ce qui rend le fait de détecter les bombes et les murs très dures, c'est donc par manque de temps que l'intelligence artificielle de notre jeu sera incomplète.

items :

Nous avons aussi implémenté un système s'items.

Dans notre jeu il y a deux grandes catégories d'Items :

_Les malus (qui sont temporaires et qui ont tous la même apparence)

_Les bonus (qui sont soit permanent et concerne donc des améliorations de stats (comme la vitesse) ou sont temporaires)

Pour l'instant il existe 6 items différents fonctionnels, il y a 4 malus et 2 bonus permanent :

_Malus inversion : qui inverse les commandes du joueur pendant 5 secondes (gauche devient droite, haut devient bas etc.…)

_Malus ralentissement : qui réduit de beaucoup la vitesse du joueur qui le ramasse pendant 5 secondes.

_Malus vitesse max : qui augmente énormément la vitesse du joueur qui le ramasse pendant 3 secondes.

_Malus No bomb : qui désactive le fait de pouvoir des bombes pendant 5 secondes, donc pendant 5 secondes le joueur ne pourra plus poser aucune bombe.

_Bonus permanent de vitesse : qui augmente un peu la vitesse du joueur qui le ramasse de manière définitive.

_Bonus permanent de power : qui augmente d'un cran la taille d'explosion des bombes.

Tous ces malus ont la même apparence (une pastille rouge) et le bonus est pour l'instant sous la forme d'une pastille jaune.

Il existe aussi un autre item spécial sous la forme d'une pastille en or qui est plus rare que les autres et qui permet de

changer les bombes que l'on place par des bombes dorées qui ont un énorme rayon d'explosion.

Tous ces items ont une chance de tomber à chaque fois qu'un mur est cassé.

5.Le Site :

Nous avons mis en ligne un site via GitHub, le but étant de créer un site partant de zéro, avec html et css, nous avons donc un site disponible à l'adresse : https://yeckes.github.io/Ceas.website/Index.html

Ce site contient un dossier image, contenant toutes les images du site, un dossier pdf contenant tous nos rapports et notre cahier des charges, ainsi que l'installeur.

Il contient aussi un fichier en .css nommé "style.css" qui permet de créer des classes pour la mise en forme

.

Il contient surtout les 4 pages html qui composent notre site :

Toutes les pages contiennent la même en-tête avec le logo de Ceas ainsi que les noms des autres pages auxquelles sont associés les liens vers celle-ci.

_Index.html Il y a sur la page d'accueil une description des membres du groupe où il est notamment spécifié leurs rôles selon notre répartition des tâches.

_Présentation.html contient une description complète de notre avancement selon le travail qui a été fait aux dates des anciennes soutenances, ainsi qu'une description de ce qui est prévu d'avoir accompli pour la prochaine et dernière soutenance. Ainsi que la description du déroulement d'une partie et en bas de cette page se trouve aussi des liens qui permettent d'accéder à nos rapports de soutenance et cahier des charges sous format pdf.

_Logiciels.html Contient une liste avec les logos de tous les logiciels que nous avons utilisé jusqu'ici ainsi qu'une description de ces logiciels et de leurs utilités. De plus un lien vers le site principal de ses logiciels est attaché à chacun des logos, donc si vous cliquez sur le logo d'un de ces logiciels vous allez arriver sur le site de celui-ci.

_Installeur.html Contient un bouton qui permet de télécharger notre installeur.

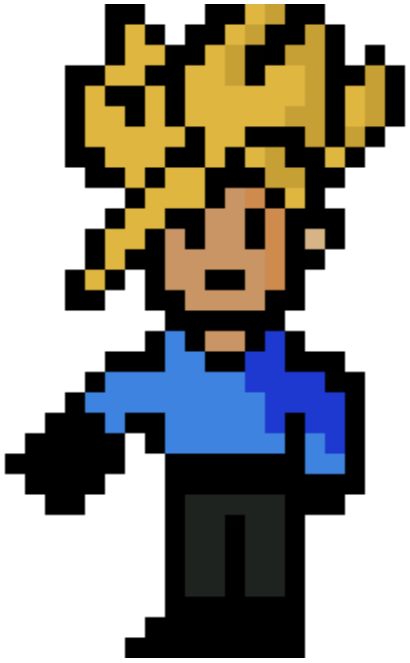
Audio/Graphisme:

Pour reprendre après la 2 -ème soutenance j'ai voulu terminer la musique du jeu. Je me suis renseigné sur Internet et notamment sur YouTube grâce aux vidéos de "Labo des Réseaux" qui m'a permis de confectionner la musique de notre jeu a l'aide d'Audacity.

Pour continuer, je me suis occupé de la partie graphismes et notamment des Sprite de personnage. Le but pour la soutenance finale était d'avoir plusieurs personnages différents à disposition pour diversifier les parties.

Pour ce faire j'ai dû créer les personnages en pixel art a l'aide de Gimp, mais je n'en connaissais pas grand-chose. J'ai donc dû me renseigner sur YouTube, et me suis appuyé de vidéos de la chaine "Creersonjeu.fr".

J'ai donc créé un premier personnage, de face, qui sera utilisé dans la sélection du personnage. Avec mon premier modèle accompli , j'ai pu continuer en créant 2 autres personnages.



Premier personnage

Deuxième personnage

Troisième personnage

Après ces personnages de séléction avant que la partie de commence, il a fallu crée les différents sprites des différents personnes , dépendament du mouvement effectué.

J’ai donc du crée trois autres sprites pour un meme personnage.