Gameplay répliqué (TP3)

Aurélien ARGOUD, Julien BRILLET

1. **Gérer le Scoring**

Le Game Designer en charge du scoring a eu l’idée géniale d’ajouter du gameplay sur la bombe. Il vous demande de prototyper son gameplay. Ce qu’il veut c’est qu’un kill de NPC rapporte 10 points. Son expérience lui permet d’anticiper les problèmes, et pour éviter de frustrer les joueurs, il vous demande de partager (en parts égales) les points de kill entre tous les joueurs qui feront des dégâts dans un certain laps de temps. Au vu de la vitesse de déflagration, vous vous mettez d’accord sur 450 ms.

1. Spawnez un ennemi dans la map, en utilisant le code existant. Complétez-le pour que l’ennemi soit répliqué. Ce NPC peut rester immobile pour votre prototype.

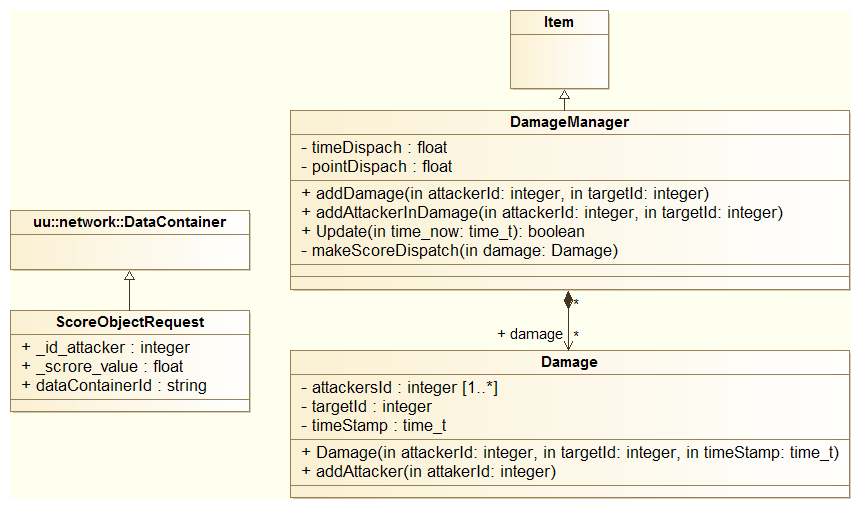
Au moment de la création d’une session, 5 NPC sont instanciés dans la fonction Game::\_OnUI.

Votre lead (qui sait qu’il aura à gérer les dégâts directs entre joueurs), vous demande de créer un manager dédié à la gestion des dégâts. Ce manager devra centraliser les demandes et gérer le scoring.

1. Proposez une architecture pour ce manager. Faites les schémas que vous jugerez utile pour expliquer votre système.

Nous avons décidé que la décision des points ce ferais chez l’hôte de la partie. Du fait que l’explosion des bombes est synchronisée chez tous les joueurs, nous avons également décidé que la décision de points s’appuierait exclusivement sur la réalité de l’hôte. Grâce à la synchronisation des bombes cela ne posera aucun problème.

Voici le diagramme de classes des éléments que nous avons ajoutés. *DamageManager* est le manager des dégâts faire sur une personne et décerne les points. *Damage* est la liste des personnes qui touche une cible dans un laps de temps. *ScoreObjectRequest* est le message de répartition des points.



Le *DamageManager* est créé par l'hôte de la session dans *Game::\_OnUI*. Seul l'hôte l'instancie et gère la répartition des points. On a rajouté dans *Game*, un pointeur de *DamageManager*. Ce pointeur sert à déterminer si on est le garant des points, et à la communication avec le manager. Le *DamageManager* est aussi ajouté dans la liste des *entities* pour l’utilisation de l’*update()*.

En plus de cela, nous avons ajouté les fonctions d’envois et de réception des messages de score (*ScoreObjectRequest*) dans *Game::OnNetwork()*.

Pour le bon fonctionnement du manageur, nous avons désactivé la gestion des dégâts des bombes si l’émetteur n’est pas l’hôte. Comme cela, l’hôte ne reçois pas de doublons de dégâts, les *hit()* des bombes en local et les messages provenant des joueurs. Par contre, l’hôte doit envoyer l’information pour que les joueurs gère ces dis dégâts entres les *Character*s.

Pour gérer correctement le scoring, nous avons ajouté à la classe *Player* une variable contenant son score. Comme cela, tous les joueurs peuvent connaitre le score de tout le monde. De plus, nous avons ajouté sur la classe *Bomb*, l’id du *Player* qu’il l’a posé. Sinon, nous aurions été incapables de savoir à qui remettre les points sur une bombe touche quelqu’un.

1. Faites le développement du manager et n’oubliez pas de commenter le code.
2. Validez la feature scoring en plaçant les deux joueurs à côté du même NPC et en activant la bombe quasi simultanément. Capturez une vidéo (par exemple avec FRAPS). Servez-vous en pour valider le fonctionnement.

Pour faciliter la manipulation et validé le fonctionnement, nous avons effectués deux tests avec des temps de décision pour le manager différents. Le premier test a un temps de 2 secondes et, le second a un temps de 450 ms.

Nous pouvons voir clairement sur les vidéos que les points sont bien répartis entre les joueurs. Les points sont affichés dans le log, dans un message d'erreur entouré de retour à la ligne pour mieux les reconnaitre.

1. Pour une meilleure compréhension, vous pouvez afficher le scoring ou le sortir en log.
2. Activez NEWT, et testez avec plusieurs latences (allez-simple) : 80ms, 150ms, 250ms, 1000ms. Que constatez-vous dans chaque cas? Comment l’expliquez-vous? Est-ce problématique et pourquoi?

Il n'y a aucun problème avec la latence en ce qui concerne les bombes. Les bombes sont répliquées sur le serveur hôte et on utilise uniquement ses bombes locales pour définir les points à répartir. Si deux joueurs posent une bombe au même instant, le décompte avant l'explosion sera adapté pour faire exploser les bombes au même timestamp.

Il faudrait avoir une latence de plus de 4000ms (décompte initial de la bombe) avant de voir la bombe exploser en retard. Si cela s’aggrave encore, en dessus de 4449ms (décompte + temps avant répartition des points), le joueur n'aura pas les points qu'il aurait dû avoir.

1. Est-ce que votre manager est fonctionnel en cas de paquets perdus ? Expliquez pourquoi ? S’il vous manque des informations allez au paragraphe 1b.

Il y a trois cas où cela pose problème:

* Si la trame de réplication de bombe a été perdue lors de la réplication vers l'hôte. La bombe du joueur sera présente dans sa réalité, mais pas dans celle de l’hôte, du coup son attaque ne sera jamais prise en compte.
* Si la trame de réplication des dégâts n'est pas envoyée vers les clients, alors le *Character* en question sera mort partout sauf chez celui qui n'a pas reçu la trame. Ce *Character* ne pourra plus mourir car il est déjà mort chez l'hôte et les bombes du client ne lui feront rien.
* Si la trame de répartition des points est perdu en allant chez le client, alors le tableau des scores ne sera pas le même chez toutes les personnes. L'Hôte aura cependant toujours le bon tableau.

Le Game designer passe tester votre prototype et croit remarquer qu’avec 150ms de latence (aller simple) le NPC ne meurt pas instantanément chez les joueurs.

* 1. Est-ce le cas ? Et pourquoi ?

Oui, la bombe explose partout au même timestamp mais l'information de dégâts et de mort est reçu 150ms plus tard chez les clients. Donc, le NPC meurt 150ms après que la déflagration de la bombe le touche.

* 1. Si oui, comment pouvez-vous corriger cette impression ?

On pourrait quand même tuer le NPC au moment où la bombe le touche. Mais dans ce cas, il faut le ranimer s'il est toujours vivant chez l'hôte, et ne pas lancer l'animation de mort quand on reçoit la bonne trame cette fois.