UNIVERSITÉ DE SHERBROOKE

Battle for Pangora

Rapport Jouabilité en jeux vidéos

Équipe : Violet Murder

Table des matières

1 Introduction **3**

2 Système de combat **3**
2.1 Description . . . 3
2.1.1 Machine à états . . . 3
2.1.2 Les attaques . . . 3
2.1.3 Les défenses . . . 5
2.1.4 Cycle de dégat . . . 5
2.1.5 La mise en buffer . . . 6
2.2 Flexibilité et adaptabilité . . . 6
2.2.1 Data Driven . . . 6
2.2.2 Progression du jeu et Équilibrage . . . 6
2.2.3 Caractéristiques drivable . . . 7
2.2.4 Lien avec les animations . . . 7

3 Points de contrôles **8**
3.1 Vitesse de capture . . . 8
3.2 Vitesse de soin . . . 9

1 Introduction

Dans Battle for Pangora, nous utilisons principalement 2 systèmes de gameplay intéressant dont nous aimerions parler ici.
L'un est un système de combat qui se veut aussi simple et flexible que profond.
L'autre est un système de points de contrôle à la base même des conditions de victoire du jeu.

2 Système de combat

2.1 Description

Nous avons décidé de créer un jeu dynamique où le combat au corps à corps serait le centre du gameplay.
Pour rendre cette expérience aussi intéressante et passionnante que possible, nous nous sommes dès le début inspiré d'une architecture sous la forme d'une machine à état proposé par Adrian Sotello.
Vous pouvez trouver l'adresse ici : http://www.adriansotelo.com/Blog.html

2.1.1 Machine à états

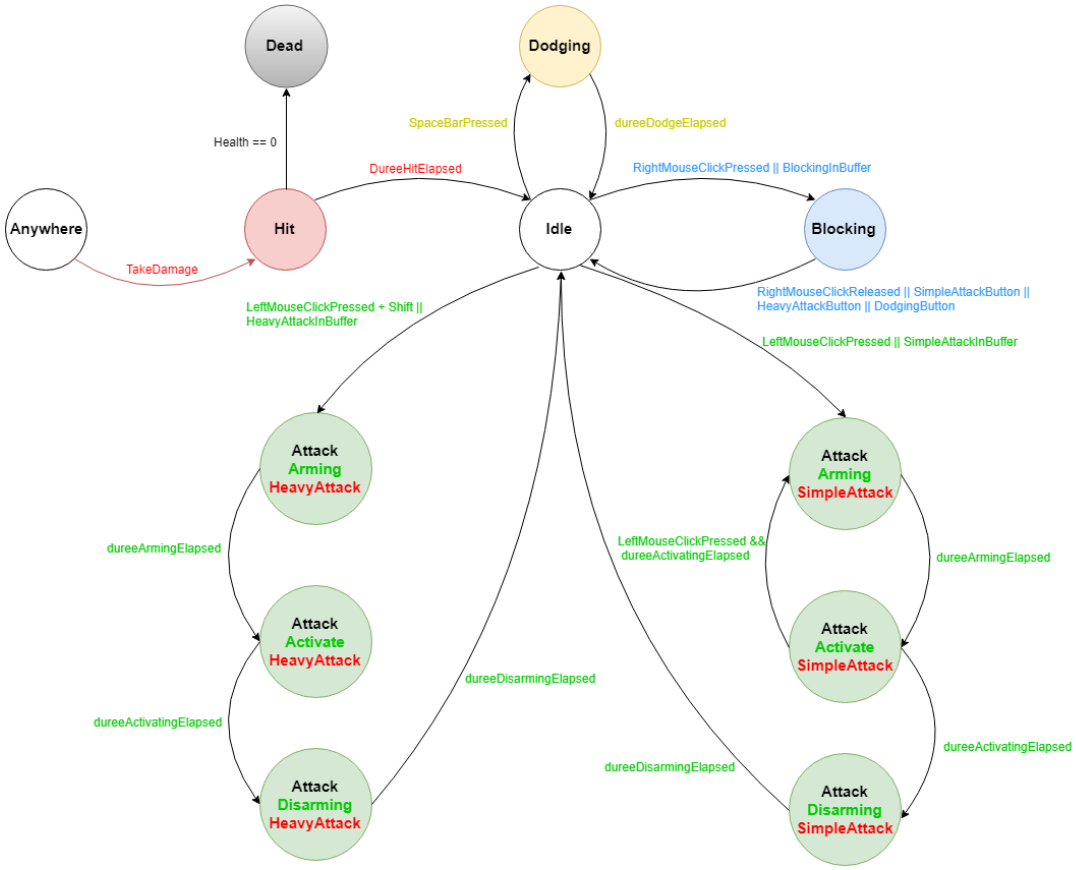
Nous avons donc utilisé une machine à état dont voici un schéma :

Ainsi notre personnage peut, actuellement, effectuer 4 actions différentes :
- lancer une attaque simple
- lancer une attaque lourde
- faire une roulade et esquiver
- passer en mode parade

Á chaque fois que le personnage termine une action, il retourne alors dans l'état Idle d'où il pourra à nouveau effectuer d'autres actions. Dans l'état Idle, le passage à un autre état est toujours instantané ce qui permet de donner un bon sentiment de réactivité au joueur.
On remarquera l'existence de l'état Hit pouvant être atteint depuis n'importe quelle autre état, et ce sans la volonté du joueur. Il s'agit bien entendu de l'état dans lequel se trouvera le joueur après avoir reçu un coup. Il devra y rester plus ou moins longtemps en fonction de la puissance du coup reçu !

2.1.2 Les attaques

En ce qui concerne les attaques, celles-ci sont décomposées en 3 phases :
- la phase d'armement : c'est le moment où le personnage lève son bras pour se préparer à frapper. Elle a une importance capitale dans le gameplay car, de par sa nature inoffensive (on ne fait pas de dégats pendant cette phase), elle permet



de prévenir l'adversaire que nous nous appretons à lancer une attaque et donc lui permettre de réagir. Permettant ainsi d'avoir une interaction entre les deux joueurs.

— la phase d'activation : c'est le moment où les boîtes de collision des attaques sont actives, si un autre joueur entre en contact avec celle-ci alors il subira des dommages ! Cette phase est souvent courte pour mettre l'accent sur un moment unique de danger, cela permet de donner plus d'importance à chaque coups, tout en rendant chaque coup plus rapide, rendant le jeu plus dynamique.

— la phase de désarmement : c'est le moment après l'attaque où le personnage est vulnérable pour avoir mis de l'énergie dans son coup. Elle est importante pour permettre aux adversaires ayant réussi à esquiver la phase d'activation d'aller chercher une récompense en attaquant le joueur s'étant rendu vulnérable.

Il y a une différence importante entre l'attaque simple et l'attaque lourde. L'attaque simple est faite pour pouvoir être enchaîné tandis que l'attaque lourde représente un pic de puissance. Ainsi, si le joueur le désire il peut enchaîner les attaques simples, créant ainsi des combos de coups. Cela a pour effet de déclencher une attaque différente à chaque fois et surtout d'ignorer la phase de désarmement des attaques, ceci dans le but de permettre aux joueurs de plus facilement rentrer dans le flow lorsqu'ils affrontent des ennemis peu coriaces mais nombreux tel que les sbires.

2.1.3 Les défenses

Un personnage possède deux façons de se défendre :

— la parade : celle-ci est un état qui peut être maintenu indéfiniment par un personnage. Tant que le personnage est dans cet état, il ne peut plus être passé dans l'état Hit par les attaques adverses. Cependant, aucune parade n'est parfaite, et donc le personnage laissera quand même passer un certain nombre de dégats à traver sa garde. Ce n'est donc pas une solution à long termes et les joueurs devront donc trouver des moyens de sortir de cette impasse, sous peine de mort ! On notera que le passage en mode parade est instantané pour récompenser des réactions rapides des joueurs.

— la roulade : cet état fait foncer le personnage en avant pendant une certaine durée à une certaine vitesse (plutôt rapide). Pendant cette période le personnage est complètement invulnérable et ne peut être ni passé dans l'état Hit par des attaques adverses, ni subire le moindre dégats de celles-ci. Cependant, une fois une roulade lancé, il faudra aller jusqu'au boût de celle-ci, nous rendant vulnérable à la sortie car il y a un cooldown sur la parade et nous devons attendre un peu avant de pouvoir en lancer une autre.

Il existe cependant des attaques spéciales dans le jeu pouvant ignorer la parade et la roulade. Certaines ignorent spécifiquement la parade, c'est le cas par exemple d'un jet d'acide corrosif.
Certaines ignorent à la fois la parade et la roulade, c'est le cas par exemple d'un feu de camp !

2.1.4 Cycle de dégat

Voici un exemple d'une séquence typique de notre système de combat :

— le héros appui sur son clique gauche
— il passe en mode Attaque/Armement
— après son temps d'armement il passe en mode Attaque/Activation
— la boîte de collision autour de sa hâche est alors crée
— la boîte de collision entre en collision avec un sbire adverse
— mais celui-ci était en mode parade, il ne subit donc que 50% des dégats
— et n'est pas projeté en arrière ni passé en état Hit
— après le temps d'activation, le joueur a à nouveau appuyé sur son clique gauche
— il passe alors en mode Attaque/Armement de sa deuxième attaque : c'est un combo
— après son temps d'armement de sa deuxième attaque il passe en mode Attaque/Activation
— la boîte de collision entre en collision avec un sbire adverse
— mais celui-ci n'était plus en mode parade !
— il subit alors 100% des dégats, est projeté en arrière, et est en état Hit pendant une durée, l'empêchant de contre-attaquer pendant cette période
— le héros continue ensuite son acharnement sanguinaire dans la joie et la bonne humeure.

2.1.5 La mise en buffer

Il subsiste cependant un problème avec l'architecture actuelle. Si les intelligences artificielles sont capables de déclencher une attaque à la première frame où elles repassent en état Idle, ce n'est pas le cas d'un joueur humain ! C'est une terrible nouvelle car cela nous empêcherai de pouvoir exécuter des combos proprement.
Pour pallier ce problème, nous avons mis en place un système de buffering.

Lorsqu'un joueur, ou une IA, désire effectuer une action, si le personnage est en état Idle alors il l'éxécute, mais sinon cette action est mise dans un buffer et sera éxécuté lorsque le personnage reviendra à nouveau dans l'état Idle. Nous permettant d'enchaîner des combos.

Pour éviter un sentiment de latence (et donc de frustration) inévitable avec ce système, le buffer n'est pris en compte que s'il a été alimenté dans les dernières 0.3s. De plus, certaines actions sont prioritaires dans le buffer. Par exemple, si un personage spam son bouton d'attaque pendant un combo mais décide finalement de parer en cliquant (et restant appuyé) une fois, mais qu'il continue un peu à spammer son bouton d'attaque alors comme la parade est une action prioritaire au buffering sur l'attaque il pourra quand même parer le coup et être satisfait !
De plus la parade est particulière car elle a une durée infinie de buffering.

2.2 Flexibilité et adaptabilité

2.2.1 Data Driven

Nous pensons que notre système de combat est relativement simple mais permet cependant une grande profondeur de jeu. En effet, par la connaissance des timings des attaques, leurs portées, leurs dégats et conséquences, les joueurs peuvent prendre l'avantage les uns sur les autres en utilisant les bonnes combinaisons d'attaques au bons moments, ainsi qu'en évitant par exemple d'affronter trop d'ennemis simultanément.

Cependant, pour permettre vraiment à notre système d'exprimer toute sa flexibilité, nous devons permettre aux designers (nous en l'occurrence) de pouvoirs facilement et rapidement modifier tous les facteurs de notre système de combat !
C'est la que notre système Data Driven intervient !

Nous avons essayé au maximum de laisser le contrôle de tous les paramètres importants dans les Blueprints d'Unreal, permettant ainsi de modifier les paramètres à la volée sans avoir besoin de recompiler tout le code source à chaque fois.

2.2.2 Progression du jeu et Équilibrage

Notre jeu étant un jeu en réseau asymétrique il est par nature un jeu multi-joueurs. Et dès qu'un jeu est multi-joueurs, il en devient par essence compétitif (sous peine de

tomber dans une suite sans fin d'exploit empêchant de profiter de l'expérience de jeu).
Le problème des jeux compétitifs, c'est leur équilibrage. Cet équilibrage est tellement difficile à obtenir et à maintenir qu'il semblerait utopique de vouloir générer tous les paramètres automatiquement, il y aura toujours des failles ! C'est pour cette raison que nous avons choisis de régler les paramètres de notre système de combat manuellement.

De plus, pour des raisons de consistance, nous avons également fait le choix de ne pas faire évoluer les différents personnages au fur à et mesure de la partie. Nous n'avons donc pas d'arbres de compétences ou de courbes d'expérience à montrer.

2.2.3 Caractéristiques drivable

Une des forces de notre système de combat réside dans sa flexibilité. En effet, il y a de nombreux paramètres que l'on sera capable de modifier via les blueprints, donc le data, de notre jeu :

- vie des personnages
- vitesse des personnages
- le nombre d'attaques des personnages
- à l'intérieur des attaques on peut :
  - changer les dégats
  - changer les 3 durées de l'animation de l'attaque (armement, activation, désarmement)
  - le point d'accroche sur le squelette du personnage
  - la forme de la boîte de collision
  - la poussée de l'attaque (sa force et sa durée)
  - la capacité d'ignorer la parade adverse
  - la capacité d'ignorer les roulades adverses
- à l'intérieur de la roulade on peut :
  - changer sa durée
  - changer sa vitesse
  - changer son cooldown de désactivation
- le coefficient d'absorption de la parade

Et tout cela nous donne la possibilité de créer un grand nombre de combinaisons différentes et intéressantes avec toujours le même système de combat !

2.2.4 Lien avec les animations

Un autre point que nous voulons également souligner. Il s'agit du lien avec les animations.
En effet, étant donné que nous ne sommes pas des animateurs, nous avons dû faire avec les animations à notre disposition. Et il se trouvait que régulièrement (tout le temps) les durées des animations n'étaient pas égales aux temps que nous aurions souhaité du point de vue du game design. Nous pourrions par exemple vouloir qu'une attaque soit plus lente à charger pour équilibrer le jeu.

Pour pallier ce problème nous avons, pour chaque animation, répertorié la durée que prend ce que nous considérons comme étant la durée de l'armement, respectivement pour la durée d'activation et la durée d'armement.
Ensuite, avec les durées voulus par le game design nous sommes en mesure de calculer le playrate de l'animation par partie.
Sur la partie armement, la vitesse à laquelle on joue l'animation est égale à :
$playrate = 1.0f/(tempsVoulu/tempsAnimation)$
Et nous faisons de même pour les deux autres durées, ce qui nous permet finalement de toujours avoir des animations qui correspondent exactement à nos attentes !

3 Points de contrôles

Après de longues réflexions, nous avons trouvé un point de notre gameplay qui pourrait être plus facilement sujet à du gameplay numérique !
Il s'agit des points de contrôle (PDC). L'idée est qu'ils pourraient jouer un rôle d'équilibrage dynamique de la partie pour avantager le camp (Héros ou Davros) le plus défavorisé et ainsi permettre des rebondissements de situation !

En pratique nous allons exprimer cela de deux manières.

3.1 Vitesse de capture

Il faut savoir que tous les personnages ont un taux de capture permettant de capturer plus ou moins vite un PDC. Le taux de capture de Davros est par exemple très rapide, tandis que celui d'un sbire est nul et celui d'un Héros est modéré.

Imaginons que nous ayons N PDC controlés par une des 2 équipes dans notre partie (en fait nous en avons 8, mais on peut facilement généralisé), alors si nous possédons déjà N/2 PDC nous sommes à égalité, dans ce cas nous voulons que nos taux de capture de soient pas altérés. Mais si nous sommes en-dessous de N/2 PDC alors nous aimerions avoir un avantage, tandis que si nous sommes au-dessus nous aimerions être un peu freiné.
De manière intuitive, on pourrait imaginer que l'on ne voudrait jamais pouvoir faire varier notre taux de capture de plus du double ou de la moitié, ainsi on peut définir notre taux de variation maximal valant 2.0f. Cette valeur pourrait bien entendu être changé à tous moments.
Maintenant il ne nous reste plus qu'à faire une interpolation linéaire ! On se retrouve avec la formule suivante :
$taux_{final} = taux_{base}/(2 * nbPDC_{captures}/nbPDC_{total})$
Il pourrait même être encore plus intéressant d'appliquer un carré sur cette formule pour que plus on est loin de la moyenne, plus le système aide les joueurs à redresser la pente ! D'ailleurs le carré n'est qu'une idée, il peut s'agir d'un exposant quelconque ! Voici la nouvelle formule :
$taux_{exposant} = ((taux_{final}/taux_{variationMax})^{exposant}) * taux_{variationMax}$
où $taux_{variationMax} = |taux_{base} - taux_{base_{max}}|$

On remarquera qu'il s'agit d'une formule multiplicative, cela a notemment pour intérêt que, même si les sbires ont un taux de capture de 0, quoi qu'il se passe il ne pourront jamais capturer un point, ce qui nous arrange bien !

3.2 Vitesse de soin

Une des particularités de nos PDC est qu'ils permettent de soigner les personnages de leur faction qui se situent dessus.
Toujours dans le but de rééquilibrer la partie, on pourrait appliquer le même raisonnement que précédemment avec pour quantifier le soin reçu par secondes.
Ainsi il y aurait un taux de soin de base, une variation de soin maximum et un exposant nous permettant de faire varier notre courbe.
Avec bien sur, 1 pour valeur dans l'exposant signifiant que l'on souhaite garder une interposation linéaire.

Conclusion

Ainsi nous avons à la fois appliqué les principes du Data Driven ainsi que les techniques de gameplay numérique à notre projet tout en essayant au maximum de conserver l'identité de notre jeu qui reste avant tout un environnement multi-joueurs nécessitant de pouvoir adapter le gameplay rapidement et efficacement :)