# Explication Projet :

Le but de ce projet est de réaliser un tournois fictif entre une liste de personnages qui sera tiré aléatoirement (par une lotterie).

Le joueur diposera de x personnages, et pourra les placer comme il le souhaite dans le tournois.

L’ordinateur va ensuite tirer aléatoirement aussi des cartes.

Dans le combat 1vs1 \*4, si en demi finale, on affronte un mec de son camp, alors on choisit lequel envoyer et il reçoit un petit buff selon son coéquipier.

RNG(-1 à +1) sur l’atk, la def et les PV.

Les PV sont régénérés à chaque fin de tournois (donc conserver Pvmax).

On peut donc obtenir 2 « Ether », et en choisir un plutôt qu’un autre car les stats.

On peut revendre un personnage.

* Stocker une variable $stars qui donne la rareté du personnage.
* Dessiner sur la carte (le perso) le nombre d’étoiles.

Gagnant d’un duel :

**A.ATK – B.DEF - (B.ATK – B.DEF) + (A.PV – B.PV) / 4**

PV restant du gagnant :

**(Perdant.ATK – Gagnant.DEF) \* (1 + Perdant.ATK / Gagnant.ATK)**

« Si le gagnant à beaucoup d’attaque, l’ennemi à probablement mis qu’un coup voire moins. »

Idées pour la suite :

Une variable $\_SESSION[‘personnages’] qui liste tous les persos que je possède.

Nouvelle partie 🡪 session\_destroy()

Reset les personnages ennemis : $\_SESSION[‘ennemis’] = array()

Faire un écran spécial pour les summons, il sera également utilisé à chaque nouvelle partie pour faire \*6 summons.

Les 6 premières du début sont à personnages unique, autrement dit on ne peut pas avoir plusieurs fois le même perso, par contre après on peut.

C’est pour cela qu’on peut avoir un perso meilleur qu’un autre à cause de la RNG (random number generator).

On aura un système de vente de personnages, on sélectionne le/les persos à vendre et on obtient x pour chacun.

Créer un menu à droite avec la liste d’argents, les boutons summons / vendre.

Mettre en permanence une musique sur chaque page, avec les balises html du son.

- Créer des cartes dérivés (Dark Silarius, Velrod (Delvor ?) Noir, Nerio, Varox, Tuteur

* Passage sur Github du code pour voir l’avancement

## Introduction :

Ce projet à pour but de développer un jeu vidéo de style Gacha, avec une trentaine voire plus de personnages collectionnables.

Le jeu se joue sous forme de tournois, le joueur choisit quel perso envoyer selon les challengers et son but est de gagner la compétition.

A chaque compétition gagné, le joueur obtient de l’argent et la possibilité d’invoquer de nouveaux personnages, d’où l’esprit Gacha.

Finalité : . ? ? ?

## Objectifs & Livrables :

Réaliser un jeu gacha où l'on a une liste de persos 3\*,4\* et 5\* tirés aléatoirement avec de la proba. Chaque perso possède ses stats uniques (tiré aléatoire avec un score variant de plus ou moins 2-3 points).

Chaque perso peut être placé dans un tournois pour combattre.

4 ennemis sont prédéfinis par l’ordinateur à chaque nouveau tournoi et leur stats seront affichés dès le début du tournois (Donc pas tant que le joueur place ses persos).

Comme ça tu choisis pas à l'avance en fonction des stats mais de l'élément seulement.

Une fois les persos placés, le tournoi peut commencer et déterminer le vainqueur. Si le joueur a gagné depuis une manche antérieure il reçoit un bonus d'argent proportionnel.

L'argent permet de faire 1-10 summons, on collectionne des cartes et on peut en vendre (si un doublon est moins bon par contre exemple), prix proportionnel aux étoiles. En cas de combat du même camp, c'est aléa + heal, ou best stats?

## Grandes étapes de réalisation :

Les grandes étapes de réalisation du projet sont listées ci-dessous dans l’ordre :

Les sous-étapes sont décrites par des tirés indentés.

* Créer un modèle de tournois et les emplacements des persos.
  + Ecriture HTML à coup de <ul>/<li> et CSS pour mise en forme
  + CSS : li.game-top : border-bottom : 3px solid grey;
* Créer une div listant les persos (enregistré en session).
  + Div avec classe liste, et une boucle for pour chaque perso enregistrée, plus l’affichage. La boucle contient une liste de div, avec notamment l’image.
  + Une option a été ajouté pour afficher les stats dans une fenêtre en cas de survol sur le perso.
* Créer un personnage à travers une classe et ses attributs, lui attribuer un nom pour afficher son image.
  + Création de la classe personnage, avec un $nom, $atk, $def, $pv, $pvm, $elmt, $camp, avec que des GETTERS et SETTERS.
  + SETTERS avec stats rand(-1,1) sur l’atk, la def et les pvm. Et pv <= à pvm.
* Créer un générateur random de personnages, issue de la liste des persos, et les attribuer aux cases tournoi ennemi et persos du joueur.
  + Array\_rand(liste\_persos,1) pour l’index, et le perso est liste\_persos[var], on peut ensuite l’instancier en donnant son nom (avec .png par ex), son nombre d’étoiles (rareté), et la liste des joueurs dans laquelle piocher.
    - On pioche dans la liste pour retrouver les stats « de base » associés.
  + On peut attribuer un perso à une case en mettant une valeur php dans <img src= > en mettant le getter **« perso->nom() ».**
* Enregistrer chaque case du tournois dans une liste pour effectuer la progression.
  + **Tableau $persos\_totaux, avec 2 niveaux de table, [round][numero],** round correspond au numero du round, de 0 à 3 (3 = 4e = gagnant). Et numero correspond à la case du personnage dans ce round, ainsi, on ira de [0][1] à [0][8], de [1][1] à [1][4], de [2][1] à [2][2] et juste [3] car unique.
  + Le tableau contient donc une liste de personnages, les rounds 1, 2 et 3 seront créés au fil du tournois, et remplis par les persos gagnants.
* Faire combattre les personnages du premier round, sauvegarder les gagnants au round d'après.
  + Créer une fonction **combattre(perso1,perso2)** qui va tester les attributs atk(), def() et pv() des deux joueurs pour savoir le gagnant. Cette fonction retourne un entier valant 0 ou 1, correspondant au numéro de camp.
  + 0 = Joueur, 1 = Ennemi (Ordi). Ainsi, le perso ayant la valeur de **->camp()** identique à celle de **combat(J1,J2)** sera le gagnant de la manche. Donc on aura **$persos[1][$i] (donc le perso à mettre au round 2, celui d’après) qui sera égale à : = $persos[0][$i + camp]**, l’index du perso sera « sommé » par le camp. Car $i = Joueur, $i+1 = Ennemi. Car les 8 cases.
* Vérifier le camp de chaque nouveau combattant, faire les combats ou soin.
  + On vérifie si les nouveaux persos (avec round = 1), ont l’attribut **->camp()** d’identique ou non, car au round 0, on peut avoir 2 gagnants du même camp. Si c’est le cas, on ne fait pas de combat et l’un des deux persos est envoyé au round suivant, avec un petit soin pour aider.
  + On aura donc **perso[2][$i] = perso[1][rand($i,$i+1)]**.
  + Si les 2 persos ont un camp différent, on refait un combat pour déterminer le vainqueur.
* Avancer au rang 3, puis pareil la finale.
  + Au rang 2 (donc manche 3), on revérifie les camps, et combat si camp !=.
  + Au rang 3, on a donc le perso vainqueur, et avec son **->camp()** on peut déterminer s’il s’agit de la victoire ou de la défaite du joueur.
* Afficher si on a gagné ou perdu et attribuer les gains pertes.
  + On joue sur la variable session « argent » du perso, gain ou perte.
  + Spécificité : si le perso à gagné depuis le round 2, il reçoit +1 d’argent. S’il gagne depuis le round 1 (donc après les 1er combat), il reçoit +2 d’argents.
* Relancer le tournoi en reinitialisant les cases du tournoi et les ennemis.
  + On vide les tableaux sessions « ennemis » et « tournois », pour qu’aucun perso n’existe aux différents round, et que les ennemis soit regénérés.
* Relancer la partie, créer un dispositif d'invocations selon l'argent, le choix et la liste des persos possibles.
  + Boutons pour lancer et avancer dans une game, pour relancer un tournois.
  + Boutons d’invocations vérifiants si le joueur a assez d’argent ou non, et utiliser une fonction **invocation(nb)** qui, comme à la première partie, donne un certains nombre « nb » de personnages aléatoirement.
* Créer un dispositif pour vendre (relâcher) les personnages.
  + Bouton pour vendre, sélection des x personnage(s), **unset($\_SESSION[‘perso’][numero])** et variable argent++ ;
* Bonus : ajouter des illustrations, animations et sons/musiques.
  + Illustrations des persos, background, image d’invocation, victoire
  + Animations de combat, animations d’invocations, d’invocations 4\*, 5\*
  + Musiques d’ambiances, de combats, sons d’invocations, de combats.
* Bonus 2 : Utiliser les éléments pour créer des efficacités / inefficacités
  + Dd
  + D
* Mettre en place un système de Compte (Login/MDP) avec une BDD.
  + Créer une BDD avec 3 tables : Joueurs (login/mdp/argents), Cartes\_Persos(nom,atk,def,pv,elmt,…) bref la liste des persos et attributs de base (sans random). Cartes\_des\_Joueurs(idJoueur,idCartes,atk\_P,def\_P,pvm\_P,…) : liste des persos que chaque joueur possède.
  + On crée un système de co /deco, deconnexion = unset session  
    connexion = vérifie login (strtolower), mdp et password\_verify avec (mdp,mdp\_save). + Charger **$\_SESSION[‘personnages’]** des personnages dans Cartes\_des\_Joueurs, avec idJoueur = id du perso qui se connecte (log).
    - Pour charger les persos, on doit adapter la classe Perso de façon à ce que le constructeur puisse prendre en argument une requête SQL, du coup on la modifie, on envoie la requête et il cherche chaque colonne pour le mettre dans un attribut du perso.
  + Système d’inscription : Si login != login\_existants, si mdp >= 5 char, on password\_hash(mdp), et on **INSERT INTO Joueurs**. + argent init = -100
    - -100 car comme ça, si argent = -100, alors summons du début
  + Système de sauvegarde : On efface les Cartes\_des\_Joueurs pour l’id actuel, puis on INSERT INTO la table les variables de **$\_SESSION[‘personnages’]** en envoyant dans le **$requete->execute( array())** la liste des attributs de chacun de nos objets personnages. Donc **foreach($\_SESSION[‘pers..’])**.
  + Evidemment, créer une BDD SQL Gacha\_LTE contenant les 3 tables.
* Mettre en place une interface (page admin) où l’on peut ajouter/éditer des persos
  + Amélioration de **Liste\_Persos.php**, les listes « list », « liste\_4\_stars » et no\_o sont générés via requête SQL : **(‘SELECT \* FROM Cartes\_Perso… WHERE STARS = …’)**, ce qui permet de générer des listes dynamiques.
    - Pareil pour les listes non-object, avec un **SELECT NOM FROM …** .
    - Bref, les listes de persos sont maintenant dynamique car BDD.
  + .
* Afficher via une icône « détails « ? » » la liste des cartes (avec images ?) + les proba en %. **🡪 Ne nécessite que de faire une requête SQL et d’afficher tout.**
  + On crée une page PHP récupérant les listes de persos objets, on affiche le nom, les stats de références (donc tout Cartes\_Personnages), + image avec nom.png .
  + On précise la rareté (étoiles) de chaque personnages et la proba à côté.
* Retravailler la structure du code, de façon à :
  + réduire le nombre de variables temporaires
  + simplifier les noms, l’écriture, l’indentation (lisibilité)
  + factoriser le code en mettant en fonction le maximum de choses
    - combats\_du\_round() est un bon exemple
  + factoriser des morceaux du code, exemple : </form> général.

Bravo ! Vous avez fini la création du jeu !

Nombre d’heures de création effectuées (historique) :

* 12/10 : 5h : creation du tournois + image des persos.
* 13/10 : 3h : creation des insertions de persos par <select><option>.
* 14/10 : 4h : ajout background et mise des persos alliés dans le tournoi.
* 17/10  + 18/10: 14h : création du Git, du menu, de intro, de la classe Perso.
  + Création des étapes du tournois, de combat(), du tableau de persos.
  + Création des variables $\_SESSION[], gestion de $camp.
* 22/10 : 4h : Création de « délivrance personnage », ajout musique, mise au propre des invocations, du tournoi. Début création de la Gacha\_BDD en SQL.
* 23/10 : 6h : Création système de connexion/deco, inscription/sauvegarde, ajout des tables, mise au propre. En gros : création de identification.php.

+ Ajout du constructeur de Perso version SQL.

* 24/10 : 3h +4h : Mise au propre de la structure du code (variables tempos), factorisation par des fonctions (ex : combats\_du\_round($round)).
  + Création de Presentation\_Persos.php, avec la liste des persos, stats, rareté, toute l’interface (galère css) + du coup passage des liste\_persos.php en SQL, maintenant les listes sont dynamiques, donc modifiable en Real time.
* 25/10 : 5h (**+3h30 design Velrod adulte**) : Création de l’interface de création de personnages (avec nom, stats, rareté et éléments), aucune valeur manuscrite, tout est prédéfini (niveau -2, -1, 0, 1, 2), ce qui évite les abus. + Créations d’une dizaine de persos (les 5\*, Zavell \*2, + tous les persos 4\*et 3\* à partir de Zavell).
  + Velrod adulte a été créé. Il a été ajouté tranquille à la liste des 5\*.
* 27/10 : ?? : Ajout des effets élémentaires

Temps total actuel : S1 : 12/10🡪18/10 : 5+3+4+14 : 26h

S2 : 22/10 🡪 24/10 : 4+6+ 3+4 + 5 + … = 22h + …

Soit : 48h

Importance des PV a été réhaussée en combat : /3 au lieu de /4.

Rendre dynamique la proba d’avoir un ennemi 4\*, 5\*.

La proba doit augmenter selon la force du joueur.

29/10/2020 : Suite au fait que Victorine a joué :

En la voyant joué, je vois beaucoup de petits défauts à corriger et améliorer, le principale étant la quantité de PV perdu entre chaque combat, c’est une valeur pouvant être négative, car elle vaut PV - ( atk2 – def1), mais si la def du perso est supérieur à l’attaque de l’autre perso, bah… Tu regagnes des PV ?!

Donc j’ai changé en mettant def/2, ce qui est au final plus logique.

Mais c’est pas encore idéal, si les stats sont super proche, on devrait perdre davantage que juste ça.

Ex : PV/ATK/DEF : 16/9/8 vs 16/7/7, le J1 perdera7-8/2 = 3, donc 13 restants.

En vrai c’est un peu exagéré, il devrait perdre un peu plus, donc à revoir.

(sûrement selon $value\_of\_fight, si il tend vers 0, alors ecart faible donc le joueur perd plus de PV ?)

Après y a aussi l’ergonomie des boutons : quand tu fais 30 invocs, c’est chiant de pas pouvoir directement éjecter tous les 3\* par exemples, ou au moins les doublons.

Après y a aussi le fait que les éléments ne soient pas présent, ça gache l’expérience parce que c’est ça qui rend la différence de perso intéressante.

Un mauvais choix peut t’empêcher de gagner un bonus de 2, voire te donner un malus de 4, donc un perso par rapport à un autre, stats équivalent, peut avoir 4 points d’écarts pour la victoire ! **Car +2 devient -2 sur $value\_of\_fight**.

11h : Elle joue vite fait pour tester les éléments, elle aime bien, je lui expliquer les erreurs de programmation que je peux avoir fait.

Exemple : Quand je calcul la valeur de l’avantage désavantage, je fais la somme du bonus élémentaires de Perso 1 sur Perso 2, puis de Perso 2 sur Perso 1.

Sauf que l’avantage P2->P1 est un avantage « négatif » dans le score.

Si P2 a l’avantage, $value\_of\_fight va diminuer, donc $bonus = negative value.

Donc je dois dire $bonus = $bonusP1P2 **- $bonusP2P1**, pour faire simple.

Et en effet, la fonction est étudié 2 fois, P1 sur P2 pour les avantages de P1.

Et on la réutilise à l’identique mais en inversant l’ordre P2 et P1 pour calculer le bonus de l’autre.

En effet, quand je regarde l’élément de P1, je lui affecte aucune valeur négative.

Donc si je veux le faire, il faut que ce soit l’élément de P2 qui soit vu en premier.

Donc ça diminue le nombre de if, ex : feu va mettre bonus sur x et y, mais pas besoin de codé le malus X et Y.

Pourquoi ? Parce qu’il sera calculé quand on inversera les 2 éléments !