# Création d'un jeu de combat Java

## Introduction

Le jeu de combat au tour par tour est un genre qui a su captiver de nombreux joueurs grâce à son approche stratégique et réfléchie. Contrairement aux jeux d'action en temps réel, ce type de jeu permet aux joueurs de planifier leurs actions à l'avance, offrant ainsi une expérience de jeu plus tactique. Les mécanismes essentiels incluent la gestion des points de vie, l'utilisation d'attaques spéciales et la prise de décisions basées sur les capacités des personnages et les mouvements de l'adversaire.

Dans un jeu de combat au tour par tour, chaque joueur a la possibilité de jouer à son tour, ce qui crée une dynamique d'interaction où la stratégie joue un rôle clé. Les joueurs doivent évaluer les forces et les faiblesses de leurs personnages, ainsi que celles de leurs adversaires, afin de maximiser leurs chances de victoire. La mécanique de jeu repose souvent sur des éléments tels que le choix d'attaques, la défense, et l'utilisation de compétences spéciales, rendant chaque tour crucial pour l'issue du combat.

Le développement d'un tel jeu en Java nécessite une solide compréhension des concepts de programmation orientée objet. Les classes représentant les personnages sont fondamentales dans la structure du jeu. Chaque personnage peut être modélisé par une classe qui contient des attributs tels que les points de vie, la force d'attaque et les compétences spéciales. Les méthodes de cette classe permettent de gérer les actions durant le combat, comme attaquer, défendre ou utiliser des objets.

En intégrant des mécanismes de jeu variés et en développant des classes bien définies pour les personnages, les développeurs peuvent créer une expérience immersive et engageante. Les décisions stratégiques des joueurs, combinées à la profondeur des mécanismes de jeu, garantissent que chaque affrontement est unique et captivant.

## But du Jeu

L'objectif principal du jeu de combat au tour par tour est de vaincre tous les adversaires en utilisant une variété d'actions stratégiques pendant les combats. Chaque affrontement nécessite une réflexion minutieuse, car les joueurs doivent choisir leurs mouvements en fonction de la situation actuelle sur le champ de bataille. Cela inclut l'attaque, la défense, ainsi que l'utilisation de compétences spéciales et d'objets, chacun ayant un impact direct sur l'issue du combat.

La structure de combat au tour par tour crée un environnement où chaque joueur a la possibilité d'agir à son tour, favorisant une approche tactique. Pendant le tour de chaque joueur, celui-ci peut évaluer les actions de son adversaire et ajuster sa stratégie en conséquence. Cette dynamique permet également d'introduire des éléments de prévision, où les joueurs doivent anticiper les mouvements de leurs adversaires pour maximiser leur efficacité.

Les différents types d'actions disponibles ajoutent une couche de complexité au jeu. Par exemple, un joueur peut choisir d'attaquer avec une attaque puissante qui inflige des dégâts importants, mais cela pourrait le laisser vulnérable à une contre-attaque. D'un autre côté, un joueur peut choisir de se défendre, réduisant ainsi les dégâts qu'il subira au prochain tour, mais cela peut également signifier renoncer à infliger des dégâts à son adversaire.

Ainsi, le but du jeu ne se limite pas simplement à vaincre l'adversaire, mais également à prendre des décisions judicieuses qui influenceront l'issue de chaque combat. Chaque tour devient une opportunité de déployer des stratégies uniques, rendant le processus de jeu captivant et imprévisible. Le véritable défi réside dans la capacité du joueur à adapter son plan de bataille face à des adversaires variés et à leurs propres choix d'actions.

## Difficultés Rencontrées

Le développement du jeu de combat au tour par tour a présenté plusieurs défis significatifs qui ont nécessité des ajustements constants et une réflexion approfondie. Parmi les principales difficultés rencontrées, la gestion des personnages a été un aspect crucial. Chaque personnage doit être unique, avec des compétences et des attributs qui influencent ses performances en combat. Trouver un équilibre entre les différentes classes de personnages pour éviter les déséquilibres a nécessité des tests rigoureux et des ajustements de statistiques.

Un autre défi majeur a été l'équilibrage des mécanismes de combat. Cela inclut la détermination des niveaux de puissance des attaques, ainsi que des effets des compétences spéciales. Si certaines attaques sont trop puissantes, cela peut rendre les combats prévisibles et ennuyeux, tandis que des attaques trop faibles peuvent frustrer les joueurs. L'équilibrage est une tâche délicate qui demande de prendre en compte les retours des utilisateurs et les données de jeu pour ajuster les mécanismes en conséquence.

La gestion des actions adverses a également posé des problèmes. Les joueurs doivent être en mesure de réagir aux mouvements de l'adversaire de manière fluide et stratégique. Créer une intelligence artificielle (IA) qui prenne des décisions judicieuses et réalistes a été un défi technique, car il faut concevoir des algorithmes capables d'évaluer la situation et d'agir en conséquence tout en évitant des comportements prévisibles.

Enfin, l'interaction utilisateur a constitué un autre obstacle. Assurer une interface utilisateur claire et intuitive est essentiel pour que les joueurs puissent se concentrer sur la stratégie plutôt que sur la navigation dans le jeu. Des éléments tels que les menus, les indications visuelles des actions possibles, et le retour d'information sur les décisions prises doivent être soigneusement conçus pour garantir une expérience de jeu fluide. Les tests utilisateurs ont été cruciaux pour identifier les points de friction et améliorer cette interaction.

## Classes

Dans la structure du jeu de combat au tour par tour, la classe principale est la classe **Personnage**, qui représente tous les types de personnages jouables. Cette classe contient des attributs essentiels tels que les points de vie (PV), la force d'attaque, la défense, et une liste de compétences spéciales. Les méthodes associées permettent aux personnages d'effectuer des actions telles qu'attaquer, défendre, ou utiliser des objets.

public abstract class Personnage implements Attaquable{

protected String nom;

protected int pointsDeVie;

protected int degats; // Dégâts de base

protected int défense; //défense de base

protected int niveau;

protected int experience;

public Personnage(String nom, int pointsDeVie, int degats, int défense) {

this.nom = nom;

this.pointsDeVie = pointsDeVie;

this.degats = degats;

this.défense=défense;

this.niveau = 1;

this.experience = 0;

}

public void recevoirDegats(int degats) {

défense -= degats;

if(défense <= 0) {

pointsDeVie -= degats;

System.***out***.println(nom + " a reçu " + degats + " points de dégâts. Points de vie restants : " + pointsDeVie);

}

}

public boolean estVivant() {

return pointsDeVie > 0;

}

public String getNom() {

return nom;

}

public void setPointsDeVie(int pointsDeVie) {

this.pointsDeVie = pointsDeVie;

}

public void gagnerExperience(int xp) {

experience += xp;

if (experience >= 100) {

niveau++;

experience = 0;

System.***out***.println(nom + " a atteint le niveau " + niveau + " !");

}

}

### Sous-classes de Personnage

1. **Guerrier** : Cette sous-classe se spécialise dans les combats rapprochés. Avec des points de vie élevés et une forte puissance d'attaque, les Guerriers sont souvent en première ligne des batailles. Leur méthode unique, **attaquePuissante()**, permet d'infliger des dégâts supplémentaires, mais avec un coût en points d'action.
2. public class Guerrier extends Personnage{
3. public Guerrier(String nom) {
4. super(nom, 100, 10,60);
5. }
6. *@Override*
7. public void attaquer(Personnage cible) {
8. System.***out***.println(nom + " attaque " + cible.getNom() + " !");
9. cible.recevoirDegats(degats);
10. }
11. *@Override*
12. public void utiliserCompetence(Personnage cible) {
13. System.***out***.println(nom + " utilise sa compétence spéciale : Coup de rage !");
14. cible.recevoirDegats(degats \* 2);
15. }
16. *@Override*
17. public void gagnerpointsdevie(int HP) {
18. System.***out***.println(nom+"libère sa puissance intérieure, récupérant sa vitalité avec une explosion de force !");
19. this.pointsDeVie+=HP;
20. }
22. }
23. **Mage** : Les Mages excellent dans l'utilisation de la magie. Ils possèdent une grande variété de compétences magiques, mais des points de vie relativement faibles. Leur méthode **lancerSort()** leur permet de causer des dégâts à distance ou d'apporter des soins à leurs alliés.

public class Mage extends Personnage {

public Mage(String nom) {

super(nom, 80, 15,20);

}

*@Override*

public void attaquer(Personnage cible) {

System.***out***.println(nom + " lance un sort sur " + cible.getNom() + " !");

cible.recevoirDegats(degats);

}

*@Override*

public void utiliserCompetence(Personnage cible) {

System.***out***.println(nom + " invoque une tempête magique !");

cible.recevoirDegats(degats + 10);

}

*@Override*

public void gagnerpointsdevie(int HP) {

System.***out***.println(nom+"canalise l'énergie mystique, régénérant sa vitalité grâce à une aura de pouvoir arcane !");

this.pointsDeVie+=5\*HP;

}

1. **Voleur** : Agile et rusé, le Voleur se concentre sur les attaques furtives et les manœuvres d'évasion. Sa méthode **attaqueFurtive()** lui permet d'infliger des dégâts critiques lorsqu'il attaque par surprise.
2. public class Voleur extends Personnage{
3. public Voleur(String nom) {
4. super(nom, 70, 12,40);
5. }
6. *@Override*
7. public void attaquer(Personnage cible) {
8. System.***out***.println(nom + " attaque furtivement " + cible.getNom() + " !");
9. cible.recevoirDegats(degats);
10. // Chance d'une attaque critique
11. if (Math.*random*() < 0.2) {
12. System.***out***.println(nom + " inflige un coup critique !");
13. cible.recevoirDegats(degats);
14. }
15. }
16. *@Override*
17. public void utiliserCompetence(Personnage cible) {
18. System.***out***.println(nom + " utilise sa compétence spéciale : Attaque rapide !");
19. cible.recevoirDegats(degats \* 3);}
20. **Bête** : Cette sous-classe représente des créatures puissantes et sauvages. Les Bêtes ont des attaques physiques dévastatrices, mais peuvent également utiliser des capacités spéciales telles que **rugir()** pour effrayer l'adversaire et réduire sa défense.
21. public class Beast extends Personnage{
22. int Strike;
24. public Beast(String nom) {
25. super(nom, 40, 75,75);
26. }
27. *@Override*
28. public void attaquer(Personnage cible) {
30. System.***out***.println(nom + " Lance Morsure Sauvage sur " + cible.getNom() + " !");
31. cible.recevoirDegats(degats \* 2);
32. }
34. *@Override*
35. public void utiliserCompetence(Personnage cible) {
37. System.***out***.println(nom + " utilise sa compétence spéciale : Hurlement Ancestral !");
38. cible.recevoirDegats(degats \* 5);
39. }
40. *@Override*
41. public void gagnerpointsdevie(int HP) {
42. System.***out***.println(nom+"canalise sa force primitive, retrouvant une énergie féroce qui la rend plus redoutable que jamais !");
43. this.pointsDeVie+=HP;
45. }
47. }
48. **Robot** : Les Robots sont des personnages technologiques avec des attributs équilibrés. Leur méthode **surveillance()** leur permet de détecter les mouvements ennemis, augmentant leur chance d'éviter les attaques.
49. public class Robot extends Personnage{
51. public Robot(String nom) {
52. super(nom,90,40,100);
53. }
54. *@Override*
55. public void attaquer(Personnage cible) {
56. System.***out***.println(nom + " Déclenche Frappe Mécatronique sur " + cible.getNom() + " !");
57. cible.recevoirDegats(degats + cible.défense);
58. }
59. *@Override*
60. public void utiliserCompetence(Personnage cible) {
62. System.***out***.println(nom + " Active Rayon Plasma Surpuissant !");
63. cible.recevoirDegats(degats\*3);
64. }
65. *@Override*
66. public void gagnerpointsdevie(int HP) {
67. System.***out***.println(nom+"active son module d'auto-réparation, restaurant ses systèmes et optimisant ses performances pour le combat !");
68. this.pointsDeVie+=2\*HP;
70. }
72. **Chevalier** : Les Chevaliers combinent force et défense, souvent en servant de protecteurs pour les autres personnages. Leur méthode **protéger()** leur permet de réduire les dégâts subis par un allié lors des attaques adverses.
73. public class Chevalier extends Personnage{
74. public Chevalier(String nom) {
75. super(nom,90,75,40);
76. }
77. *@Override*
78. public void attaquer(Personnage cible) {
80. System.***out***.println(nom + " Invoque la Lance du Destin " + cible.getNom() + " !");
82. cible.recevoirDegats(degats);
83. }
84. *@Override*
85. public void utiliserCompetence(Personnage cible) {
86. System.***out***.println(nom + " Invoque le Coup de Tonnerre " + cible.getNom() + " !");
87. cible.recevoirDegats(degats + cible.degats);
88. }
89. *@Override*
90. public void gagnerpointsdevie(int HP) {
91. System.***out***.println("Active la guérison divine pour régénérer");
92. this.pointsDeVie+=HP+50;}

### Classe Play02

La classe **Play02** est responsable de la gestion du flux de jeu. Elle coordonne les tours des personnages, gère l'initialisation des combats, et assure le suivi des états des personnages (comme les PV et les effets de statut). Les méthodes de cette classe incluent **demanderAction()**, pour interroger le joueur sur l'action à effectuer, et **mettreAJourStatut()**, pour actualiser les conditions de chaque personnage après chaque tour.

Ensemble, ces classes et sous-classes créent un cadre robuste pour les mécaniques de jeu, permettant des stratégies variées et une expérience de jeu immersive.

## Fonctionnalités

Le jeu de combat au tour par tour intègre plusieurs fonctionnalités essentielles qui enrichissent l'expérience de jeu et apportent une dynamique unique aux affrontements. Parmi ces éléments, le choix du personnage, le système de combat, les compétences spéciales et le système de points de vie jouent un rôle fondamental.

### Choix du Personnage

Le choix du personnage est crucial, car chaque personnage possède des attributs et des compétences distincts. Les joueurs peuvent sélectionner un Guerrier, un Mage, un Voleur ou toute autre classe, ce qui influence leur style de jeu. Ce choix stratégique permet aux joueurs d'adapter leur approche en fonction des forces et faiblesses de leurs adversaires, favorisant ainsi une grande diversité dans les combats.

Personnage joueur;

switch (choix) {

case 1:

joueur = guerrier;

break;

case 2:

joueur = mage;

break;

case 3:

joueur = voleur;

break;

case 4:

joueur = Beast;

break;

case 5:

joueur = Robot;

break;

case 6:

joueur= chevalier;

break;

default:

System.***out***.println("Choix invalide. Le guerrier est sélectionné par défaut.");

joueur = guerrier;

break;

}

### Système de Combat

Le système de combat est basé sur des tours alternés, où chaque joueur peut exécuter une action. Cela crée une dynamique où la réflexion et la prévision sont clés. Les joueurs doivent anticiper les mouvements de leur adversaire tout en planifiant leurs propres stratégies, ce qui rend chaque tour plein de tension et d'excitation.

### Compétences Spéciales

Les compétences spéciales ajoutent une couche supplémentaire de stratégie. Chaque personnage dispose d'aptitudes uniques qui peuvent influer sur le cours du combat. Par exemple, un Guerrier peut infliger des dégâts massifs avec une attaque puissante, tandis qu'un Mage peut soigner ses alliés ou infliger des dégâts à distance. L'utilisation judicieuse de ces compétences peut renverser le cours d'un affrontement, incitant les joueurs à bien réfléchir à leur utilisation.

### Système de Points de Vie

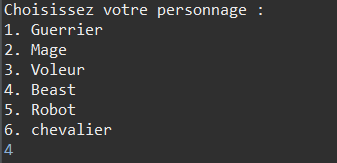
Le système de points de vie (PV) est fondamental pour la survie des personnages. Chaque action, qu'il s'agisse d'attaquer ou de défendre, impacte les PV, créant ainsi un enjeu constant. Les joueurs doivent gérer de manière stratégique leurs PV, en équilibrant entre attaque et défense pour maximiser leurs chances de victoire. Ce mécanisme renforce l'aspect tactique du jeu, où les décisions prises peuvent avoir des conséquences immédiates sur l'issue du combat.

En somme, ces fonctionnalités interconnectées créent une expérience de jeu riche et immersive, où chaque action peut potentiellement changer le cours des événements. Les joueurs sont constamment en train de s'adapter et de développer des stratégies, rendant chaque affrontement unique et captivant.

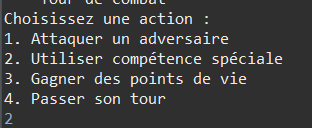
## Tests

Pour garantir la fonctionnalité du jeu de combat au tour par tour, une série de tests rigoureux a été mise en place, couvrant divers aspects tels que la sélection du personnage, l'exécution des combats, l'efficacité des compétences spéciales, et les actions des adversaires. Chaque test a été conçu pour vérifier non seulement l'intégrité du code, mais également l'expérience de jeu globale.

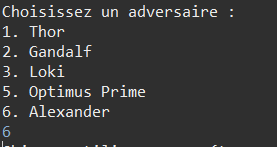
**Choix Du Caractéres :**



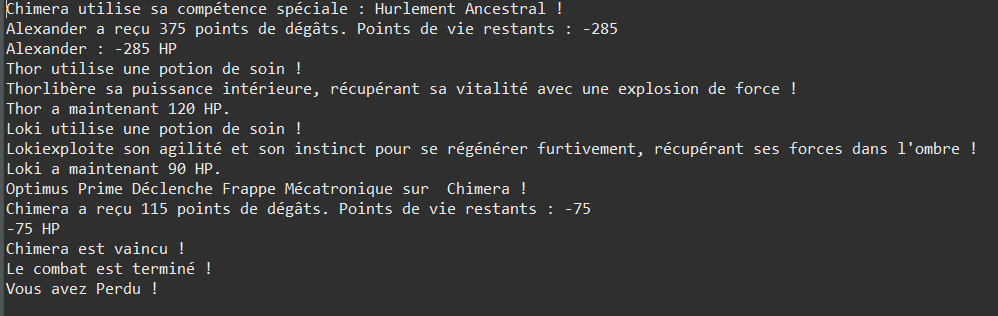
**Tour Du Combat :**



**Choix D’Adversaire :**



**Boucle Du Combat :**



### Sélection du Personnage

Le premier test a concerné la sélection du personnage. Il a été crucial de s'assurer que chaque personnage pouvait être choisi sans problème et que les attributs spécifiques à chaque classe étaient correctement affichés. Des scénarios de test ont été élaborés pour vérifier que les joueurs pouvaient facilement naviguer à travers le menu de sélection et que les changements apportés aux attributs des personnages se reflétaient immédiatement dans les combats. Des utilisateurs ont été invités à tester l'interface, et leurs retours ont permis d'identifier des points d'amélioration.

### Exécution des Combats

Les combats eux-mêmes ont également fait l'objet d'une attention particulière. Des tests ont été réalisés pour vérifier que le système de tours fonctionnait correctement, en s'assurant que chaque joueur avait bien l'opportunité d'agir à son tour. Les interactions entre les actions des joueurs ont été analysées pour garantir qu'elles se déroulaient comme prévu, sans bogues ou comportements inattendus. Les résultats des tests ont montré que le système de combat était fluide et intuitif, renforçant l'expérience immersive.

// Boucle de combat

while ( joueur.estVivant() &&( guerrier.estVivant() || mage.estVivant() || voleur.estVivant() || Beast.estVivant() || Robot.estVivant() || chevalier.estVivant())) {

System.***out***.println("\n--- Tour de combat ---");

System.***out***.println("Choisissez une action :");

System.***out***.println("1. Attaquer un adversaire");

System.***out***.println("2. Utiliser compétence spéciale");

System.***out***.println("3. Gagner des points de vie");

System.***out***.println("4. Passer son tour");

int action = scanner.nextInt();

### Efficacité des Compétences Spéciales

Un autre aspect essentiel a été l'évaluation des compétences spéciales. Chaque compétence a été testée pour s'assurer qu'elle fonctionnait comme prévu, infligeant les dégâts appropriés ou apportant les effets escomptés. Des scénarios ont été créés pour simuler diverses situations de combat, et l'efficacité des compétences en fonction des classes de personnages a été minutieusement vérifiée. Ce processus a permis d'identifier des déséquilibres potentiels, contribuant à un ajustement précis des valeurs et des effets.

### Actions des Adversaires

Enfin, la réaction des adversaires a été testée pour garantir que l'intelligence artificielle (IA) se comportait de manière réaliste et stratégique. Des tests ont été effectués pour s'assurer que l'IA pouvait évaluer la situation en fonction des actions des joueurs et répondre de manière appropriée. Ces tests ont inclus des simulations de combats où les mouvements de l'IA ont été analysés afin de garantir qu'ils ajoutaient de la profondeur et du défi au jeu.

Ces différents tests ont permis non seulement de valider la fonctionnalité du jeu, mais aussi d'améliorer l'expérience utilisateur, garantissant que chaque élément du jeu fonctionne harmonieusement et contribue à une expérience de jeu enrichissante.

Personnage [] adversaire= { guerrier,mage,voleur,Beast,Robot,chevalier};

for(Personnage cible : adversaire) {

if(cible.estVivant() && cible != joueur) {

int actionAdversaire = new Random().nextInt(4);

switch(actionAdversaire) {

case 1:

Personnage Target= *selectRandomTarget*(cible,guerrier,mage,voleur,Beast,Robot,chevalier,joueur);

if(Target!=null) {

cible.attaquer(Target);

System.***out***.println(Target.pointsDeVie+" HP ");

if(!Target.estVivant()) {

System.***out***.println(Target.getNom()+" est vaincu !");

break;

}

}

case 2:

Personnage Target2= *selectRandomTarget*(cible,guerrier,mage,voleur,Beast,Robot,chevalier,joueur);

if(Target2!=null) {

cible.utiliserCompetence(Target2);

System.***out***.print(Target2.pointsDeVie+" HP ");

if(Target2.estVivant()) {

System.***out***.print(Target2.getNom()+" est vaincu !");

}}

break;

case 3:

System.***out***.println(cible.getNom() + " utilise une potion de soin !");

cible.gagnerpointsdevie(20); // Heals 20 points of health (adjust as needed)

System.***out***.println(cible.getNom() + " a maintenant " + cible.pointsDeVie + " HP."); break;}}

## Conclusion

Le développement du jeu de combat au tour par tour a été une expérience enrichissante qui a permis d'acquérir des compétences précieuses en matière de programmation orientée objet. Tout au long du processus, nous avons appris l'importance de structurer le code de manière modulaire, en utilisant des classes et des sous-classes pour représenter les différents personnages et leurs interactions. Cela a permis non seulement de faciliter la gestion des attributs et des méthodes, mais aussi d'améliorer la lisibilité et la maintenabilité du code.

L'un des principaux enseignements tirés de cette expérience est la nécessité d'un équilibrage minutieux des mécanismes de jeu. Chaque personnage doit avoir des forces et des faiblesses bien définies pour garantir une expérience de jeu équitable et stimulante. Cela a impliqué des tests approfondis et des ajustements constants, permettant de comprendre comment les différentes compétences interagissent entre elles et influencent le cours des combats.

De plus, la gestion de l'intelligence artificielle (IA) s'est révélée être un défi majeur. Concevoir une IA capable de prendre des décisions stratégiques tout en restant imprévisible a nécessité une réflexion approfondie et des essais itératifs. Cela a mis en lumière l'importance d'un bon design de jeu, où chaque élément doit contribuer à l'expérience globale et à l'engagement des joueurs.

Pour l'avenir, plusieurs améliorations pourraient être envisagées pour enrichir le jeu. L'ajout de nouveaux personnages et de compétences uniques pourrait diversifier davantage les stratégies disponibles. De plus, l'intégration de scénarios narratifs ou d'événements aléatoires durant les combats pourrait rendre chaque affrontement encore plus captivant. Enfin, l'optimisation de l'interface utilisateur et de l'expérience de jeu globale pourrait également contribuer à rendre le jeu plus accessible et engageant pour un public plus large. En somme, cette aventure de développement a ouvert la voie à de nombreuses possibilités d'amélioration et d'enrichissement du jeu.

1. **Jeux en Java**  
   Udemy propose plusieurs cours sur le développement de jeux en Java, qui incluent des projets pratiques. Ces cours peuvent aider à acquérir des compétences pratiques tout en apprenant les concepts théoriques nécessaires.

Ces ressources peuvent fournir un cadre solide pour comprendre les principes du développement de jeux et la programmation orientée objet, tout en offrant des exemples pratiques et des conseils de la part d'experts dans le domaine.