



**MATHIEU MOREL VALENTIN CLERGUE  
LUKAS BLOUIN ANTOINE IMBERT  
PEDRO MEZQUITA FERNANDEZ**



# **Remerciements**

## **Université**

Nous souhaitons en premier lieu remercier l'Université Clermont Auvergne et plus particulièrement le département Informatique d'Aubière pour nous avoir mis à disposition du temps ainsi que des salles dans le but de réaliser des réunions hebdomadaires. Nous remercions également la DSI pour avoir installé tous les logiciels nécessaires aux projets.

## **Professeur Tuteur**

Nous remercions aussi notre professeur (M. Chafik Samir) qui a accepté de nous accompagner en tant que tuteur ainsi que pour nous avoir guidés tout au long de ce projet via des réunions hebdomadaires pendant lesquelles nous structurions notre travail et faisions le point.

## **Sophie Le Corgne**

Nous remercions tout particulièrement Mme Sophie Le Corgne pour son aide à la relecture de notre rapport.

# Sommaire

<b>I/ Introduction</b>	<b>4</b>
1. Contexte	4
2. Sujet	4
<b>II/ Présentation du jeu</b>	<b>5</b>
1. Le Plateau	5
2. Tour de jeu	5
3. Les Différentes Cartes	6
3.1 Nain	6
3.2 Mine	7
3.2.1 Les rencontres	7
3.2.2 Fin de mine	7
3.2.3 Bonus	7
3.2.4 Trésor spécial	8
3.3 Trophée	8
<b>III/ Analyse &amp; Développement</b>	<b>9</b>
1. Jeu	9
1.1 Modèle	9
1.1.1 Technologie	9
1.1.2 Conception	10
Base du jeu	10
Fabrique de Carte	10
1.1.3 Contraintes	13
1.2 Interface de jeu	14
1.2.1 Technologies	14
1.2.2 Conception	15
1.2.3 Contraintes	17
2. Site Internet	18
2.1 Hébergement web	18
2.1.1 Définition des besoins	18
2.1.2 Description de notre offre	18
2.2 Back-end	19
2.2.1 Technologies	19
2.2.2 Conception	19
PHP	19
BDD	21
2.2.3 Contraintes	22
2.3 Front-end	22
2.3.1 Technologies	22
Tailwindcss	23
2.3.2 Conception	24
2.3.3 Contraintes	26
3. Communication / serveur (page web)	27

3.1 Technologies	27
3.2 Conception	27
3.3 Contrainte	27
4. Pentesting	28
4.1 Injection HTML	28
4.2 XSS (Cross-Site Scripting)	28
<b>V/ Bilan technique</b>	<b>31</b>
1. Site internet	31
1.1 Hébergeur	31
1.2 Front-End	31
1.3 Back-End	31
2. Jeu	32
2.1 Modèle	32
2.2 Partie graphique	32
3. Communication Site Internet/Jeu	32
<b>V/ Conclusion</b>	<b>33</b>
<b>VI/ Resume</b>	<b>34</b>
<b>VII/ Webographie</b>	<b>35</b>
<b>VIII/ Annexes</b>	<b>37</b>

# I/ Introduction

## 1. Contexte

Étant tous étudiants à l'IUT de Clermont-Ferrand en seconde année, notre cursus intègre un projet tutoré qui se doit de valider un ensemble de compétences techniques.

Pour définir le sujet du projet, nous avions le choix entre :

- Un sujet proposé par un professeur de l'IUT
- Un sujet proposé par nous même et validé par l'équipe enseignante

Nous avons donc choisi de réaliser un site web permettant de jouer à un jeu de cartes dans l'univers médiéval fantasy depuis un PC.

## 2. Sujet

Ainsi, nous pouvons nous demander : « Comment pouvons-nous développer une solution permettant aux différents joueurs de TunHell de jouer quelle que soit leur localisation ? »

En effet, pour jouer à TunHell, il est nécessaire que tous les joueurs soient présents dans un même lieu, étant donné que le jeu est un jeu de cartes.

La première idée est de créer une version numérique du jeu hébergée sur un serveur et accessible depuis un site internet. Cette idée n'est pas nouvelle, plein de jeux connus fonctionnent de cette manière comme "Gartic Phone" (téléphone arabe avec des dessins) ou bien "agar.io".

La seconde idée est de faire une application à installer sur les appareils des joueurs sous la forme d'un client lourd comme "League of Legends" par exemple.

Pour savoir quelle forme est la plus adaptée, nous nous sommes basés sur le comportement d'un utilisateur lambda. Un utilisateur va chercher la simplicité, il faut donc que le joueur ait accès au jeu avec le minimum d'effort et de connaissances en informatique. Si l'interface n'est pas intuitive et nécessite beaucoup d'actions de la part de l'utilisateur pour lancer une partie, il va finir par être découragé.

En conséquence, l'option qui semble être la plus adaptée pour permettre à un joueur de TunHell de jouer depuis n'importe où, est de passer par l'intermédiaire d'un site web. De cette manière le joueur n'aura pas à installer une application à chaque fois qu'il change d'appareil et cela réduira le nombre d'actions à effectuer par les joueurs pour lancer une partie.

Ce rapport se décompose en 6 grandes parties, nous commencerons par vous expliquer l'ensemble des règles du jeu de carte Tunhell avant de vous faire part de l'analyse de notre sujet jusqu'à sa conception. Au cœur de cette partie, nous observerons l'ensemble des modules qui composent notre attendu final avant de poursuivre sur les réalisations concrètes à l'aide d'un bilan technique.

Enfin, nous conclurons notre rapport avant de vous livrer une version synthétique en Anglais ainsi que d'autres ressources comme notre documentation UML en annexe.

## II/ Présentation du jeu

TunHell est un jeu de cartes jouable de 2 à 4 joueurs, initié sur Kickstarter en 2015 par Pixie Games. Le jeu consiste à gérer une équipe de nains pour miner et récolter des ressources dans un univers médiéval fantastique.

Le but de ce jeu est d'être le joueur à extraire le plus de ressources, pour cela le joueur peut placer des nains avec différentes facultés autour d'une mine afin d'en extraire les ressources.

### 1. Le Plateau

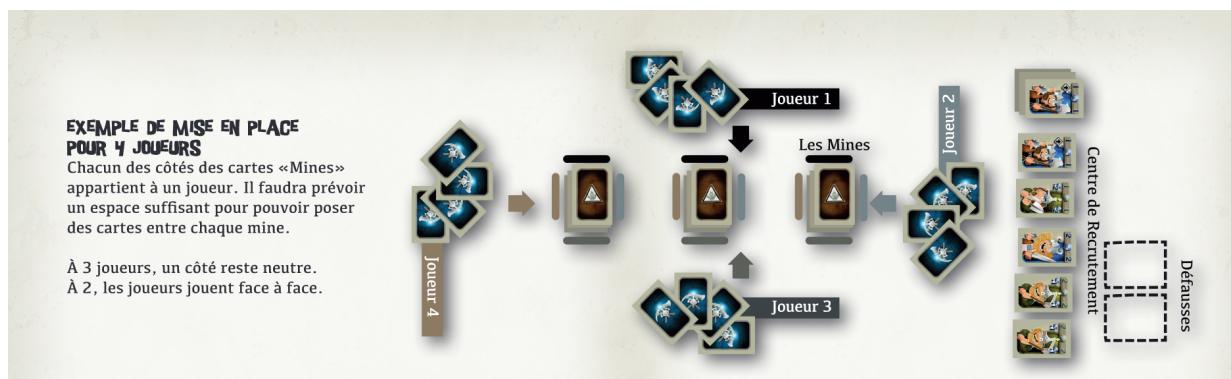


Figure 1 : Plateau de jeu Tunhell

Le jeu est composé de 44 cartes "nain", 60 cartes "mine", 4 cartes "fin de mine", 2 cartes "trophée".

Les cartes sont disposées de la manière suivante :

- 3 mines avec pour chaque tas : 1 carte "fin de mine" et 20 cartes "mine"
- 6 cartes "nain" par joueurs
- Le reste des nains est mis dans le centre de recrutement
- Chaque joueur possède un tas butin, qui correspond à toutes les richesses accumulées lors de la partie (trophée compris)

### 2. Tour de jeu

Un tour se passe de la manière suivante :

- Le joueur a le choix entre ajouter un nain dans sa main OU jouer un nain de sa main sur l'une des 3 mines
- Ensuite, les effets des cartes posées par le joueur sur les mines s'activent

Un joueur ne peut avoir que 6 cartes maximum dans sa main.

La fin du jeu arrive lorsqu'il n'y plus de cartes dans les mines, dans ce cas-là, les trophées sont décernés aux joueurs (celui qui a tué le plus de monstres par exemple).

Ensuite les joueurs regardent leurs butin et calculent leurs scores, celui qui a le plus haut score gagne.

### 3. Les Différentes Cartes

#### 3.1 Nain



Figure 2 : Les différents types de cartes nain

Les cartes "nain" possèdent 2 valeurs, la première est en rapport avec leurs types et la deuxième est un bonus.

**Exemple :** La carte verte au-dessus possède 2 points de minage et 1 point de combat.

Ensuite, ces cartes peuvent avoir une particularité :



Le symbole flèche indique qu'une fois recrutée, elle doit être immédiatement posée sur une mine.

Figure 3 : Carte nain avec le symbole "flèche"



Le symbole stop indique qu'elle doit être la seule carte posée sur la mine, donc toutes les autres cartes présentes dans la mine sont défaussées.

Figure 4 : Carte nain avec le symbole "stop"

Il existe 4 types de nains :

- Les guerriers affrontent les rencontres (monstres)
- Les éclaireurs permettent au joueur de voir les "x" prochaines cartes de la mine
- Les piocheurs permettent de retourner les cartes de la mine
- Les dynamiteurs permettent d'enlever tous les guerriers autour d'une mine

## 3.2 Mine

### 3.2.1 Les rencontres



Figure 5 : Exemple de cartes “rencontres”

Les rencontres sont des monstres présents dans les mines qui possèdent une valeur de combat et une valeur de richesse. Une fois un monstre vaincu, la carte va dans le tas butin du joueur. On peut faire diverses rencontres avec des rats, des trolls, des dragons...

### 3.2.2 Fin de mine



Figure 6 : Les cartes “Fin de Mine”

Les cartes fin de mine sont des cartes présentent à la fin de chaque de mine, chacune a sa particularité. “La porte de derrière” ne fait rien et donne une de richesse, “la grosse araignée poilue” est un monstre à affronter et “la salle de trône” maudit votre jeu et vous devez défausser toutes les cartes de votre main. Une fois la carte mise dans vos richesses, la mine est finie et le joueur doit passer à une autre mine.

### 3.2.3 Bonus



Figure 7 : Les cartes “Bonus”

Les cartes bonus sont au nombre de 3 : “la bière de la bravoure”, “nawak, l'épée de bravoure” et “la vieille pioche”. Elles peuvent, une fois associées à un nain, soit augmenter les points d'attaque, soit les points de minage, soit les deux.

### 3.2.4 Trésor spécial



Figure 8 : Les cartes “Trésor Spéciaux”

Il existe 3 types de trésors spéciaux : “Le fantôme de Grodhüre”, “Les 2 anneaux uniques”, “Le cœur d’or”. Le fantôme vous oblige à défausser une carte “nain”, le cœur d’or fait de même mais rejoint le butin. Les anneaux quant à eux doivent être obligatoirement mis dans la main du joueur ne pouvant être placés dans le butin du joueur.

### 3.3 Trophée



Figure 9 : Les cartes “Trésor”

Les trésors sont décernés aux joueurs remplissant certaines conditions à la fin du jeu. Elles rejoignent ensuite le butin dudit joueur. "L'employé du mois" est décerné à celui qui a miné le plus de carte "Terre" et le "Dératiseur" à celui qui a tué le plus de nains.

Pour toute interrogation ou incompréhension, le manuel du jeu est disponible en annexe.

# III/ Analyse & Développement

L'analyse du projet a été une partie importante de ce dernier, en effet nous n'avions jusqu'ici jamais conçu de jeu à destination des navigateurs web. Il nous a fallu approfondir le sujet afin de déterminer quels langages de programmation et quels frameworks nous allions devoir utiliser.

Si l'on synthétise les technologies utilisées dans le projet, nous avons :

- HTML5 / CSS
  - Tailwind CSS
  - Chart JS
- PHP 8
- MariaDB (BDD)
- JavaScript
  - NodeJS
  - Jest
  - TypeScript
  - matter.js
  - Colyseus

## 1. Jeu

Le but est de faire un jeu web, quand on pense à ce type de jeux, on pense à Adobe Flash Player mais il n'est plus très utilisé aujourd'hui et possède de nombreuses failles de sécurité.

Ensuite il y a Unity qui pourrait tout à fait être adapté, mais son fonctionnement particulier rendrait impossible de finir le jeu dans les temps. Donc le choix le plus évident a été JavaScript, étant très utilisé, nous étions certains de trouver des frameworks capables de nous aider pour le développement.

Nous avons décidé d'écrire le jeu en TypeScript. C'est une couche supplémentaire au JavaScript permettant d'avoir un langage typé<sup>1</sup>. De cette manière, il est plus simple de s'y retrouver avec les variables et de trouver les erreurs. Le problème de JavaScript étant que les variables sont non typées, ce qui est source d'erreur.

### 1.1 Modèle

#### 1.1.1 Technologie

Pour le back-end du jeu, nous nous sommes très vite mis d'accord sur *Node.js* qui est exactement fait pour ça. Le fait d'utiliser le même langage de programmation permet un échange d'informations entre les différentes parties du jeu plus harmonieuses. Étant confiants sur le temps de développement du jeu, nous avons utilisé "Jest", pour faire des tests unitaires, malheureusement seul un test unitaire a été écrit et permet de vérifier que les JSON sont cohérents.

---

<sup>1</sup> **Un langage de programmation typé** est un langage qui définit le type de contenu que la variable peut accepter. Un int (nombre entier) ne peut pas aller dans une variable qui n'accepte que des strings (chaîne de caractère).

### 1.1.2 Conception

Durant la conception, nous avons fait le choix de définir toutes les classes en anglais pour d'une part nous y habituer et d'autre part pour permettre une plus grande accessibilité à la compréhension du code.

Nous avons décomposé le back-end en 3 parties : la base du jeu, la fabrique de carte et le comportement du jeu (qui est énormément lié avec l'interface).

#### Base du jeu

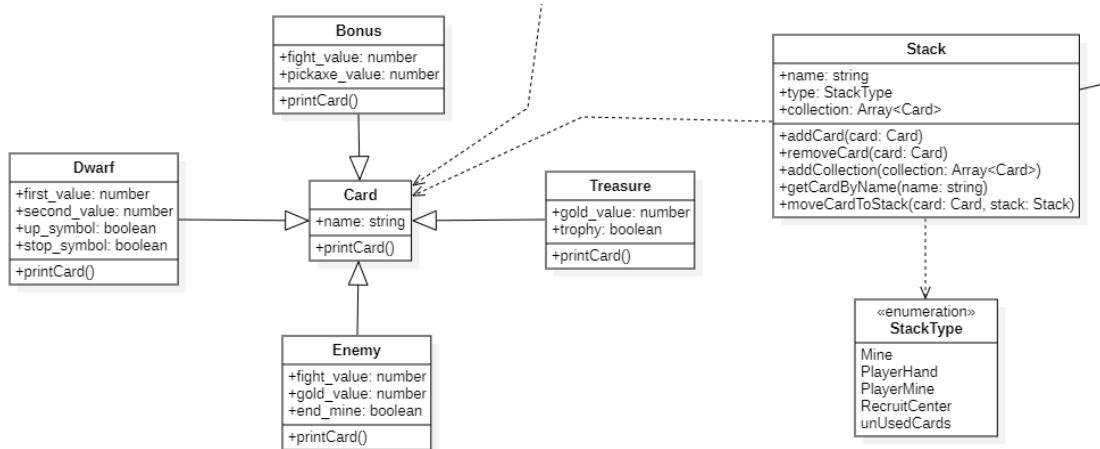


Figure 10 : Diagramme de classe des classes de “base”

Ces quelques classes définissent les différents objets que nous allons manipuler pour constituer le jeu. On y trouve Card qui se décompose en 4 sous-classes : Bonus, Dwarf, Enemy et Treasure. Nous les avons définis après avoir minutieusement les règles du jeu et déterminé quels attributs et comportements étaient à implémenter dans le jeu.

Suite à cela, pour représenter les tas de cartes, nous avons créé une classe Stack, qui en plus de représenter une liste de cartes, permet de déterminer son nom ainsi que son but dans le jeu (Mine, Main de joueur, Centre de recrutement ...).

#### Fabrique de Carte

La génération des cartes du jeu peut paraître anodine, mais elle présente de nombreuses questions sous-jacentes qui auront un impact sur le fonctionnement du jeu.

Nous avons déterminé précédemment 4 types :

- Nain
- Ennemi (ou rencontre)
- Bonus
- Trésor

**Première question** : Comment concevoir le système de génération des cartes de telle manière à ce qu'il soit extensible et maintenable facilement ?

En effet, pour la génération des cartes, on aurait pu tout simplement, les créer à la main, en instanciant les cartes une à une avec des boucles for pour le nombre voulu dans une ou plusieurs classes :

```

for(int i=0; i<5; i++)
    cards.push(new Picker(2,0,true,false))
for(int i=0; i<5; i++)
    cards.push(new Picker(4,3,false,false))

```

Figure 11 : Exemple du code pour générer des cartes

Mais cela devient vite illisible, c'est pour cela que nous avons décidé de décrire les cartes dans des JSON<sup>2</sup>, et d'écrire une classe capable de créer n'importe quelle carte de façon générique (un peu comme une recette de cuisine). De cette manière, cela réduit le code à modifier et/ou ajouter et cela simplifie l'ajout de nouvelles cartes, car il ne suffit plus qu'à l'écrire dans un JSON et créer une fabrique quand il y a un nouveau type.

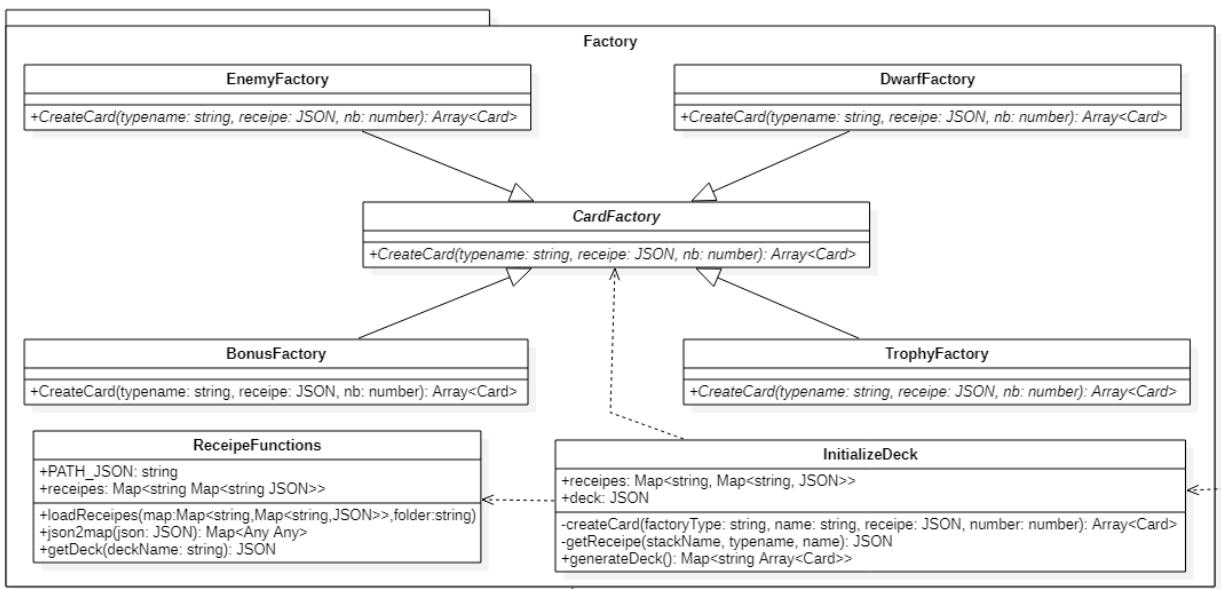


Figure 12 : Diagramme de classes du package Factory

Le package *Factory* utilise deux patrons de conception<sup>3</sup> : La Fabrique et la Façade. *InitializeDeck* est le point d'entrée du package, c'est lui qui sert de Façade aux fonctionnement interne des autres classes du package. Le module *ReceipeFunctions* lit tous les JSON et les envoie à *InitializeDeck* sous la forme d'une Map associant le nom du fichier et le JSON.

Ensuite la classe *InitializeDeck* fait corréler les “recettes” avec le JSON “Deck”, ce JSON définit le nombre, le type et l'emplacement des cartes sur le jeu, et envoie les informations aux fabriques respectives afin de générer les cartes.

À la fin, elle retourne une Map contenant 4 entrées, le nom des différents tas du jeu (Mine, Nain, Fin de mine et Trésors) associé à leur liste de cartes.

<sup>2</sup> **JSON (JavaScript Object Notation)** est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de représenter l'information d'une manière structurée.

<sup>3</sup> Les **patrons de conception** (design patterns) sont des solutions classiques à des problèmes récurrents de la conception de logiciels. Chaque patron est une sorte de plan ou de schéma que l'on peut personnaliser afin de résoudre un problème récurrent dans le code.

```

> Picker.json
{
  "Picker1": {
    "first_value": 2,           -----> Nom de la carte
    "second_value": 0,          -----> Valeur de minage
    "up_symbol": true,         -----> Valeur de minage bonus
    "stop_symbol": false,      -----> a un effet spécial ?
    "end_mine":false          -----^
                                -----> est une fin de mine ?
  },
  "Picker2": {
    "first_value": 1,
    "second_value": 2,
    "up_symbol": false,
    "stop_symbol": false,
    "end_mine":false
    etc . . .
}

```

Figure 13 : JSON définissant les cartes “Piocheur”

```

> Deck.json
{
  "Default":{               -----> Nom du deck
    "Mine":{                 -----> Emplacement sur le jeu
      "Enemy":{              -----> Type de la carte
        "Meetings":{         -----> Nom du JSON où se
          "Rat":7,            trouve la carte
          "Gobelin":4,        -----> Nom de la carte et
          "Orc":4,             nombre souhaité
          "Troll":3,
          "Dragon":3
        }
      },
      "Bonus":{              -----> Nom du JSON où se
        "Bonus":{            trouve la carte
          "Beer_of_bravery":2,
          "Nawak_sword":1,
          "Old_pickaxe":1
        }
      },
      "Treasure":{            -----> Nom du JSON où se
        "Special_treasure":{   trouve la carte
          "Unique_rings":2,
          "Grödur_ghost":2,
          "Hearth_gold":1
        }
      }
    },
    etc . . .
}

```

Figure 14 : JSON définissant le “Deck”

Cependant, la distribution des cartes des différents types n'est pas homogène sur le plateau. Par exemple, “Dwarf le chien” et “Machin” qui peuvent être trouvés dans la Mine sont considérés respectivement comme un Nain Guerrier et un Nain Piocheur dans notre code. Or contrairement à toutes les autres cartes Nain, elles se trouvent dans la Mine.

**Deuxième question :** Devons-nous générer les cartes par leurs types puis placer les cartes particulières dans leurs tas respectifs ou devons-nous générer les cartes par le tas et donc spécifier les types entre chaque tas ?

Nous avons opté, dans un premier temps, pour la 1<sup>re</sup> solution. Nous nous étions venus à la conclusion qu'il n'y aurait que quelque carte à retrouver et à re-dispatcher dans leurs tas respectifs. Malheureusement, il s'est avéré que pour les retrouver, l'algorithme devait parcourir toutes les cartes, ce qui demande une deuxième étape coûteuse en ressources et complexifie le code.

C'est pour cela que nous avons fait marche arrière et restructuré les JSON pour qu'ils génèrent les cartes par tas et non par type (comme vous pouvez le voir sur les exemples précédents).

### 1.1.3 Contraintes

Durant la conception des classes métier, nous avons rencontré beaucoup de problèmes, notamment causés par la complexité des règles. L'une d'entre elles est la généralisation et différenciation de toutes les cartes.

Prenons l'exemple des nains, il existe 4 types de "nain" représentés par leurs symboles :



Figure 15 : Les différents types de carte "Nain"

Il serait donc logique de faire une classe "Nain" et des sous-classes pour chaque type :

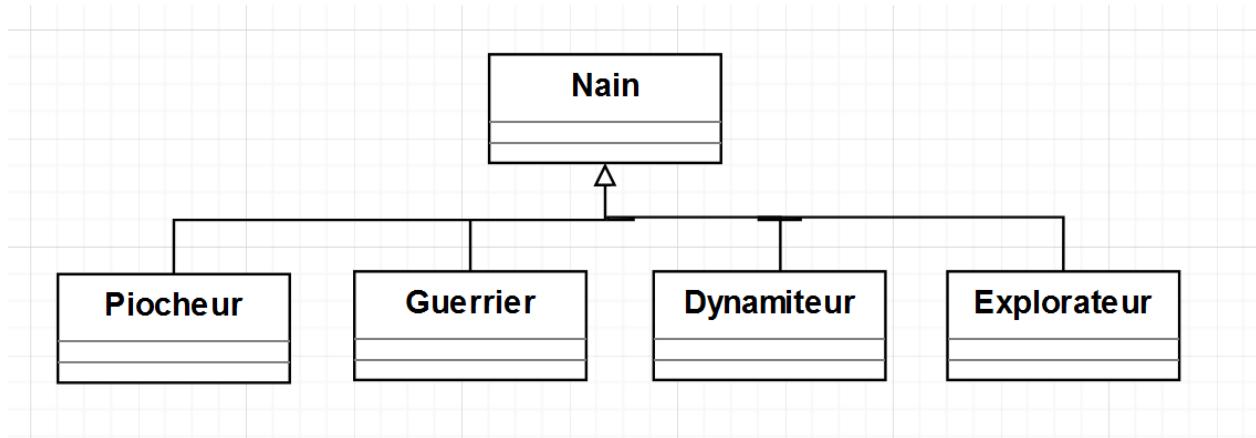


Figure 16 : Exemple de conception des classes Nain

Un nain aurait donc une valeur correspondant aux points de minages, attaques, explorations. Le contexte de cette valeur ne serait défini que par la sous-classe.

Logiquement, nous aurions alors à nous dire que chaque sous classe redéfinit le comportement en jeu de la carte. Mais étant donné que les cartes agissent sur le jeu, cela veut dire qu'une carte connaît l'ensemble du jeu pour pouvoir agir dessus, ce qui n'est pas en

adéquation avec le principe S de SOLID<sup>4</sup> : Responsabilité unique (Single responsibility principle)<sup>5</sup>.

De ce fait, nous avons décidé de séparer le comportement de la carte, ce qui veut dire que tout ce qui différencie un piocheur d'un guerrier, est le nom de sa classe.

Plusieurs choix s'offraient à nous pour nous permettre de reconnaître le "sous-type" de la carte.

Nous avions le choix entre :

- Garder les sous-classes de Nain et donc de regarder l'appartenance de l'objet, mais cela aurait été superflu étant donné que ni les méthodes, ni les attributs de Nain ne changent
- Utiliser un attribut "sous-type" dans Nain

C'est donc pour la deuxième solution que nous avons opté. Néanmoins, devons-nous utiliser une chaîne de caractères ou une énumération ?

Nous avons fait le choix d'utiliser une chaîne de caractères pour éviter d'avoir à faire la correspondance entre la chaîne de caractères définissant le type dans le JSON et l'énumération correspondante et cela nous évite d'avoir à ajouter à l'énumération le type à chaque nouveau type.



De plus, un Nain peut avoir en plus de sa valeur (minage, combat...), une deuxième valeur d'un autre type.

Figure 17 : Exemple d'une carte "Nain" avec une deuxième valeur

## 1.2 Interface de jeu

### 1.2.1 Technologies

Pour l'interface du jeu, nous sommes partis à la recherche d'un moteur graphique pouvant fonctionner sur une page web. Après quelques recherches, nous sommes tombés sur des moteurs de jeu tel que *matter.js*, *melon.js*, *phaser* et plein d'autres<sup>6</sup>. Il a été difficile de trancher, surtout qu'il en existe aussi en *Kotlin* tel que *KorGE* (Kotlin Game Engine).

Vous vous demandez sûrement pourquoi ne pas écrire un moteur de jeu nous même ? Étant donné que nous avons très peu d'expérience dans la création d'un jeu et du langage JavaScript, nous allons utiliser un moteur de jeu web déjà fonctionnel. L'avantage est qu'il ne sera pas nécessaire d'en écrire un par nous même et cela nous permet de réduire la charge de travail qui est conséquente.

<sup>4</sup> **SOLID** est un acronyme mnémonique utilisé en programmation orientée objet, qui regroupe cinq principes de conception destinés à produire des architectures logicielles plus compréhensibles, flexibles et maintenables.

<sup>5</sup> **Single responsibility principle** (ou principe de responsabilité unique) consiste à s'assurer qu'une classe n'a qu'une seule responsabilité.

<sup>6</sup> Github recensant les moteur de jeu JS : <https://github.com/collections/javascript-game-engines>

Afin de ne pas s'éparpiller sur de nombreuses technologies, nous avons décidé de prendre un moteur de jeu JavaScript.

*melon.js* et *phaser* n'étant peu ou plus maintenus, nous sommes partis sur *matter.js* qui propose un moteur de jeu 2D avec de la physique, ce qui nous aurait éventuellement permis de faire des animations dans le jeu.

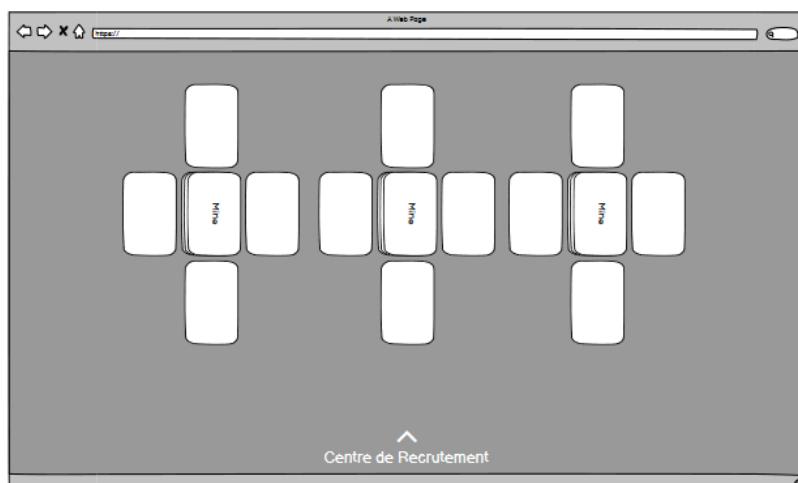
### 1.2.2 Conception

Lorsque nous avons fait le Wireframe<sup>7</sup> du site internet, nous avons aussi imaginé à quoi pourrait ressembler une partie de Tunhell.



Figure 18 : Wireframe du lobby du jeu

Une fois que la personne a créé une partie, un code de partie est créé afin de permettre aux autres joueurs de rejoindre la partie. Une fois que les 3 autres joueurs sont présents sur le lobby, le créateur de la partie peut lancer cette dernière.



<sup>7</sup> Le **Wireframe** est une maquette de l'interface, c'est un outil très utilisé pour la conception et le design de la structure d'une application.

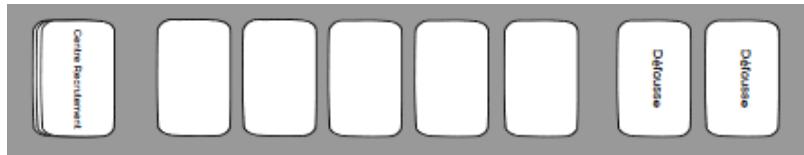


Figure 19 : Plateau de jeu sur PC

Une fois la partie commencée, les joueurs ont sur leur téléphone, leur main et leurs richesses. Sur l'ordinateur où la partie a été créée le plateau s'affiche. Le plateau représente les différents tas de carte comme la mine, le centre de recrutement et la défausse. Les joueurs peuvent aussi faire le choix de cacher le centre de recrutement afin de mieux voir le plateau.

Il est également possible d'afficher le plateau sur d'autres appareils en supplément du principal.

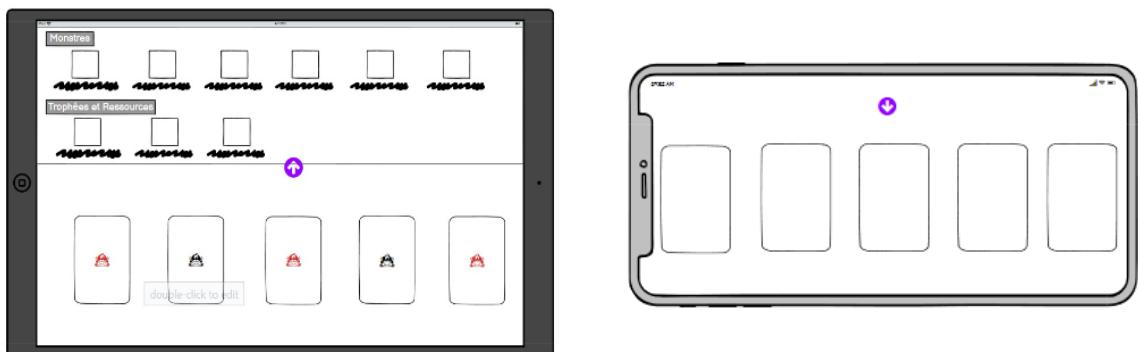


Figure 20 : Plateau de jeu sur des appareils mobiles

Si jamais les joueurs n'ont pas de smartphones, ou ne souhaitent pas utiliser cette fonctionnalité, il est possible d'afficher tout le jeu sur le PC.

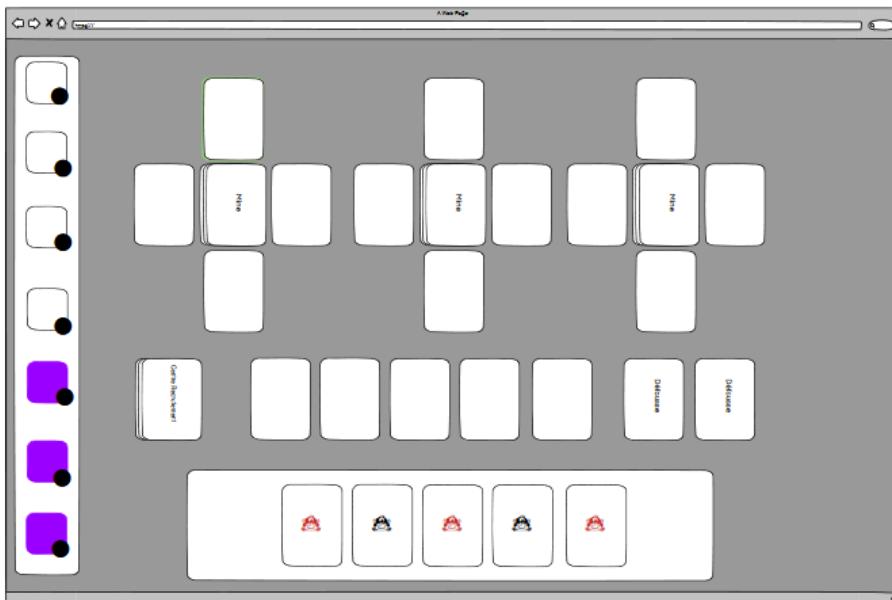


Figure 21 : Plateau de jeu avec la main des joueurs

Dans ce cas-là, les informations de chaque joueur sont affichées tour par tour. Le problème de ce mode-là est que chaque joueur peut voir le jeu de l'autre, ce qui peut influencer le jeu.

### 1.2.3 Contraintes

Malheureusement par manque de temps, nous n'avons pas pu réaliser cette partie du projet. À la place, nous avons conçu une interface simpliste sur terminal avec des questions afin de déterminer les actions du joueur.



Figure 22 , 23 : Capture d'écran de l'interface du jeu

Dans cette capture d'écran les cartes sont affichées par leur nom et type. Ensuite nous proposons au joueur de choisir entre prendre une carte ou jouer une carte. Une fois ce choix fait, le modèle agit en conséquence et affiche une ou plusieurs questions ou met fin au tour si aucune action n'est nécessaire.

Si nous le souhaitions (cela n'a pas encore été fait), nous pourrions intégrer le jeu de cette manière sur le site. Le jeu serait moins attrayant mais fonctionnel.

## 2. Site Internet

Il nous fallait un site internet permettant d'accueillir le jeu et d'expliquer comment jouer à notre jeu, pour cela nous sommes partis sur *PHP* et *MariaDB* pour le Back-end et *HTML/Tailwind CSS* pour le Front-end.

### 2.1 Hébergement web

Afin d'héberger notre site, nous avions besoin d'un hébergeur web permettant d'à la fois contenir la partie esthétique (pages de présentation) ainsi que la partie contenant le jeu en lui-même (modèle).

#### 2.1.1 Définition des besoins

Nous avions donc besoin d'un hébergeur pouvant supporter les technologies citées précédemment.

Cet hébergeur doit également être capable de supporter la création et le déroulement de plusieurs parties en même temps par plusieurs joueurs disposant chacun d'une connexion internet différente. Le débit offert par l'hébergeur devenait alors un point crucial car il devait être suffisant afin de ne pas entraver l'expérience utilisateur.

#### 2.1.2 Description de notre offre

Nous avons de ce fait opté pour l'hébergeur "Minestrator" que l'un de nos membres du projet avait déjà utilisé et expérimenté avant. Cet hébergeur répondait aux attentes que nous nous étions fixées en termes de technologies, il ne restait plus qu'à mener quelques tests afin de s'assurer qu'il répondait aux attentes concernant le débit proposé.

Nous avons donc mené 2 tests mineurs dans ce domaine :

- **Un test de débit FTP :**  
Sur un fichier unique de 4 Go avec la fibre (D: 1 GB/s | U: 0,4 GB/s)  
On obtient : Débit ascendant : 8.1 MB/s et Débit descendant : 62.3 MB/s
- **Un test de DOS<sup>8</sup> :**  
Cela a pour but de mesurer le nombre de connexions simultanées possible via HTTP/HTTPS. Le test a été réalisé avec High Orbit Ion Cannon, nous constatons que le site web n'est pas vulnérable aux attaques DOS (paramètre du logiciel : 6 threads et utilisations des scripts)

Ensuite nous avons effectué une analyse du serveur afin de déterminer concrètement ce que propose le serveur.

Notre offre ne nous permettait pas de nous connecter en SSH dans le serveur, nous avons cherché une alternative pour nous permettre d'utiliser Git par exemple.

Nous avons également créé un script PHP qui nous a permis d'exécuter des commandes bash dans le serveur, nous avons donc appris que :

- Notre serveur n'a pas de serveur SSH installé

---

<sup>8</sup> **DOS** (Deny Of Service) est une attaque qui consiste à faire des requêtes en boucle à un serveur dans le but de le faire tomber en panne.

- Nous pouvons exécuter des scripts shell
- Nous pouvions utiliser git directement sur le serveur (mais nous n'avons pas fait cela, car ceci nous pose un problème de sécurité puisque nos clefs privées pour le git resteraient en accès public)

Finalement, tous ces tests nous ont permis de connaître le débit réel du serveur et de déterminer qu'il ne sera pas possible d'utiliser git afin de cloner le projet directement sur le serveur. Nous utiliserons à la place, le protocole FTP (File Transmission Protocol) pour copier les fichiers si nécessaire.

## 2.2 Back-end

### 2.2.1 Technologies

Pour ce qui est du back-end<sup>9</sup>, nous avons choisi d'utiliser le langage de programmation PHP car l'on venait tout juste de l'apprendre. Contrairement à JavaScript qui nous était inconnu jusqu'alors. Cela nous a permis un développement du site plus facile et rapide.

Le PHP gère la redirection des pages du site, la connexion et déconnexion d'utilisateurs ainsi que l'envoie des données relatives au compte d'un utilisateur connecté.

Pour le choix de la Base de Donnée (BDD) nous avons opté pour MariaDB, un système de gestion de base de données que nous savons utiliser. N'ayant pas de grosses quantités de données à traiter, ce système nous est amplement suffisant.

### 2.2.2 Conception

#### PHP

Pour le backend nous avons utilisé le patron MVC (Modèle-Vue-Contrôleur) qui structure le code PHP.

Le patron MVC est un patron de conception en PHP qui permet de séparer les différentes classes en 3 rôles principaux :

- Gestion du modèle, avec les classes métier ainsi que les gateway pour la communication avec la BD.
- Vue, qui va afficher le contenu PHP et qui va interagir avec l'utilisateur.
- Contrôleur, qui va séparer les différentes actions en fonction de l'utilisateur. Il y a deux types de contrôleurs :
  - Un frontController qui va valider les requêtes des utilisateurs puis appeler le contrôleur d'actions correspondant.
  - Les contrôleurs d'actions qui vont permettre d'appeler les méthodes correspondantes selon l'action demandée par l'utilisateur.

Cette structure nous donne une grande flexibilité en cas d'ajout de vues ou de fonctionnalités, elle nous aide aussi lors de la connexion avec la base de données et par sa sécurité grâce à la simple implémentation d'une classe de vérification.

---

<sup>9</sup> Le backend est toute la partie que l'utilisateur ne voit pas, mais qui lui permet de réaliser des actions sur un site ou une application.

Concernant la connexion entre le PHP et la base de données, nous avons utilisé PDO (PHP Data Object) qui est un pilote de communication entre PHP et les bases de données.

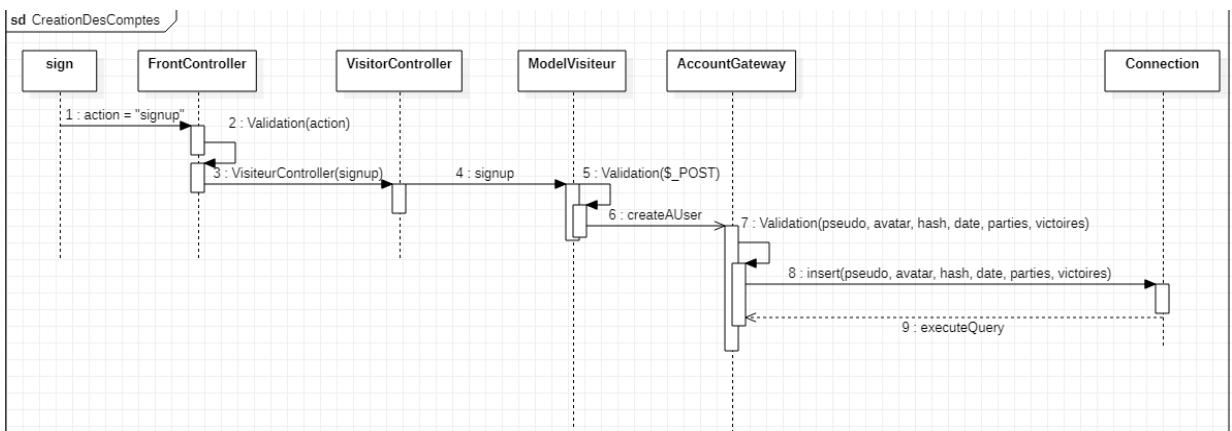


Figure 24 : Diagramme de séquence détaillant la création des comptes utilisateur

Ce diagramme montre les appels dans le code qui sont effectués lors de la création d'un compte utilisateur. Lorsqu'un visiteur (c'est-à-dire, une personne qui n'est pas connecté avec un compte utilisateur) rempli le formulaire et clic sur le bouton, la page appelle index.html avec une variable PHP appelé "action" avec une valeur "signup", les données de création de comptes sont envoyés avec la méthode POST, le FrontController va récupérer la variable "action" et selon la variable appeler le contrôleur correct (dans le cas de "signup" notre FrontController va appeler le VisitorController) ainsi qu'il va vérifier la valeur passée en paramètre, le VisitorController appelle donc la fonction signup et cette fonction va appeler le ModelVisiteur. Le ModelVisiteur va tout d'abord vérifier les valeurs données par l'utilisateur et après appeler la méthode createAUser de AccountGateway, qui va être en charge de vérifier une dernière fois les valeurs données par l'utilisateur pour après appelée la base de données et créer l'utilisateur.

Cette procédure nous donne une bonne organisation des classes, où chaque classe PHP possède sa propre tâche unique, mais elle nous facilite aussi la modification du code et la sécurisation de la base de données car nous pouvons vérifier les différentes données à des différentes étapes selon le rôle de la classe correspondante.

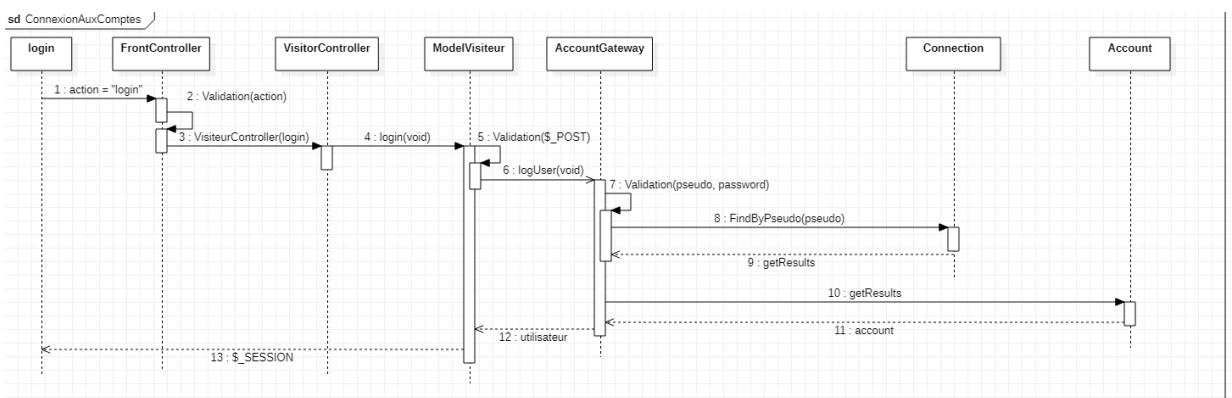


Figure 25 : Diagramme de séquence détaillant la connexion aux comptes utilisateur

Ce Diagramme de séquence nous montre les appels effectués dans le code lorsqu'un utilisateur se connecte. En premier l'action "login" est transmise au frontController qui va alors valider si l'action est valide puis initialiser un visiteurController. Un modelVisiteur va être initialisé à son tour puis va valider les variables récupérées dans les champs du formulaire de connexion. Ensuite une gateway liée à la classe account va communiquer avec la base de donnée pour vérifier que l'utilisateur est bien existant. Si l'utilisateur existe bien, on récupère ses informations dans la base de donnée, on crée une instance d'utilisateur que l'on renvoie au modelVisiteur. Pour finir, le modelVisiteur met à jour la variable de session dans le navigateur puis la page d'accueil est affichée.

Le principe est simple, nous avons un site internet sur lequel figurent des informations comme les règles du jeu, la possibilité de création de parties personnalisées, etc. Pour accéder à notre site un joueur le recherche sur un moteur de recherche

Depuis la page principale, il peut voir les informations principales comme les règles du jeu, les informations juridiques, une page de contact, ou encore un tutoriel pour créer une partie. Il a la possibilité de rejoindre une partie grâce à un code à 4 chiffres, mais peut aussi créer un compte ou se connecter pour créer sa propre partie. Si un utilisateur est connecté, il a accès à une page de profil où il peut consulter ses statistiques de jeu, modifier son pseudo ou son mot de passe.

En effet, une personne qui possède un compte peut créer une partie personnalisée qui doit accepter 4 personnes afin de lancer la dite partie.

Cette personne sera le maître du jeu de la partie. En lançant une partie, cela créera un salon d'attente dans lequel des personnes avec ou sans comptes pourront rejoindre avec un code de partie.

Ce code de partie peut être saisi dans un formulaire disponible sur la page d'accueil. Ainsi, la personne le rentrant rejoindra le salon d'attente et une fois que 3 autres personnes auront rejoint cette salle d'attente, le maître du jeu pourra lancer la partie.

Si la personne ayant rentré le code s'est connectée depuis un téléphone, il n'aura, sur son écran (une fois la partie lancée) que la main de son joueur, c'est -à -dire ces propres cartes et non le plateau au complet.

S'il s'est connecté depuis un ordinateur, il aura un plateau de jeu ainsi que sa main.

## BDD

Pour sauvegarder et gérer les différents utilisateurs nous avons décidé de créer une base de données très simple, tout d'abord nous avons pensé à ajouter une table "Accounts" pour les utilisateurs qui vont se connecter avec un compte, puis nous avons aussi ajouté une table "Games" qui gardera en mémoire, différentes informations sur les parties s'étant déroulées.

C'est ainsi que nous avons créé une base de données qui, en restant minimaliste, nous est très utile pour notre projet.

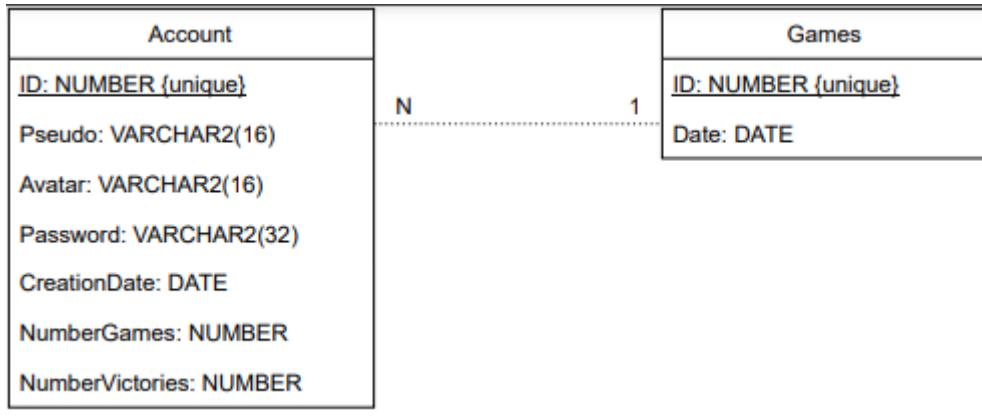


Figure 26 : Modèle conceptuel des données (MCD) de notre Base de Données

Ce MCD est relativement simple , mais l'on peut quand même y retrouver toutes les informations nécessaires à une gestion de compte rapide.

Dans un premier temps, nous avons travaillé le modèle de base de données en local en reproduisant le MCD que nous avions conçu auparavant, puis nous l'avons exporté et importé dans le serveur.

### 2.2.3 Contraintes

La réalisation et conception de la partie PHP se sont déroulées sans problème particulier. Cependant, nous verrons que n'est pas forcément le cas de la partie front-end.

## 2.3 Front-end

### 2.3.1 Technologies

Pour ce qui est du Front-end<sup>10</sup>, HTML/CSS et JavaScript sont les différentes parties qui composent notre landing page, qui est notre site de présentation du jeu.

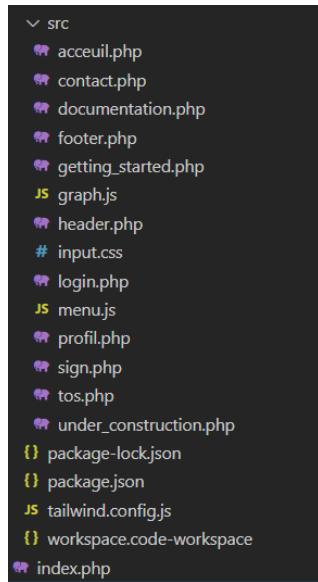
Concernant les frameworks nous avons :

- Tailwindcss : est un framework CSS qui simplifie la mise en page et raccourcit le temps de développement, il est utilisé dans de nombreux projets comme le top 10 de Netflix
- ChartJS : est une bibliothèque JavaScript permettant de réaliser des diagrammes en camembert ou en colonne facilement

---

<sup>10</sup> **Le front-end** c'est la partie du code qui est reçue par le client, par client nous désignons le navigateur internet. Il s'agit finalement des éléments du site web que l'on aperçoit à l'écran et avec lesquels on pourra interagir.

## Tailwindcss



Tailwindcss utilise un fichier JavaScript de configuration qui nous permet de sélectionner l'emplacement des vues.

De plus, il est fourni avec des composants de bases qui sont des classes CSS préconçues.

En effet, pour que Tailwindcss nous génère un fichier .css, nous devons installer le client sur notre machine.

Après avoir lancé la commande d'observation :

```
npx tailwindcss -i src/input.css -o dist/output.css  
--watch
```

Le client observe les changements à chaque enregistrement dans l'ensemble de nos fichiers. De ces changements, il embellit le fichier d'output.

Figure 27 : Structure du projet .php

Tailwindcss n'est pas utilisé par Netflix sans raison, en effet nous y voyons 4 avantages :

- Une bonne connaissance du CSS. En effet, un utilisateur de Tailwindcss doit savoir utiliser les éléments de base comme les grid, les flexbox, ...

```
<p class="text-xs text-orange-500">Étudiant</p>
```

- Le temps de développement : c'est un atout majeur. En effet, ce système de classe bas niveau n'offre pas de composants tout pré-réalisés comme Bootstrap mais juste des composants de bases comme flex, mt-{x} (margin top),
- Enfin, concernant le responsive<sup>11</sup>, Tailwindcss part de la vue smartphone pour montrer jusqu'à la vue écran large avec un système de breakpoint. Ce système permet de simplifier l'écriture du responsive et de développer dans un temps record.

```
<div class="grid gap-8 grid-cols-1 md:grid-cols-2  
lg:grid-cols-3">
```

---

<sup>11</sup> Le **Responsive** Design adapte la mise en page d'un site en fonction de la résolution d'écran. L'objectif est bien sûr de faciliter la lecture et la navigation de l'internaute quel que soit le type de terminal utilisé.

```

require_once(" .grid {
    display: grid;
section class=" }
                ju


Voici un exemple de génération de grille css.



### 2.3.2 Conception



```

usecaseDiagram
    actor "Utilisateur" as U
    actor "Utilisateur Connecté" as UC
    usecase "Se créer un compte" as SC
    usecase "Se connecter" as SC
    usecase "Rejoindre une partie" as ROP
    usecase "Créer une partie" as CUP
    usecase "Se déconnecter" as SD
    usecase "Supprimer son compte" as SSC
    usecase "Consulter ses statistiques" as CSS
    usecase "Changer Informations" as CI
    usecase "Mot de passe" as MP
    usecase "Pseudo" as P

    U --> SC : 
    U --> ROP : 
    U --> CUP : 
    UC --> SD : 
    UC --> SSC : 
    UC --> CSS : 
    UC --> CI : 
    UC --> MP : 
    UC --> P : 

    SC <<-->> ROP : <<Include>>
    ROP <<-->> CUP : <<Include>>
    SD <<-->> SSC : <<Include>>
    SD <<-->> CSS : <<Extends>>
    SD <<-->> CI : <<Extends>>

```



The diagram shows the use cases for the TunHell website. It includes two actors: 'Utilisateur' and 'Utilisateur Connecté'. The 'Utilisateur' actor is associated with three use cases: 'Se créer un compte', 'Se connecter', and 'Rejoindre une partie'. The 'Utilisateur Connecté' actor is associated with six use cases: 'Créer une partie', 'Se déconnecter', 'Supprimer son compte', 'Consulter ses statistiques', 'Changer Informations', 'Mot de passe', and 'Pseudo'. There are several associations between the use cases, including 'include' relationships from 'Se connecter' to 'Rejoindre une partie' and from 'Rejoindre une partie' to 'Créer une partie', and 'extends' relationships from 'Se déconnecter' to both 'Supprimer son compte' and 'Consulter ses statistiques'.



Ce diagramme de cas d'utilisation recense l'ensemble des actions utilisateurs ayant un compte ou pas. Dans cette partie, nous allons observer comment nous avons agencé le Front-End



Figure 28 : Diagramme de cas d'utilisation du site internet



The image shows a comparison between a real website screenshot and a wireframe. The left side shows a mobile device screen with a wireframe overlay. The right side shows a desktop browser window with a wireframe overlay. Both screens display the TunHell homepage, which features a main message 'Toi aussi rejoins une partie de TunHell' with buttons for 'Affichage d'un plateau', 'Affichage d'un plateau et de la mo', 'Code de la partie PLA-101', and 'Rejoindre'. The right side also shows a decorative illustration of various cartoonish characters.



Figure 29, 30 : Comparaison de la page d'accueil réel avec celle du wireframe



L'objectif premier de ce projet est de rejoindre une partie en ligne. Il fallait que le formulaire pour rejoindre une partie apparaisse en première page. Sur le format mobile ainsi que sur le format PC, un utilisateur peut donc rejoindre une partie en rentrant un code.



24


```

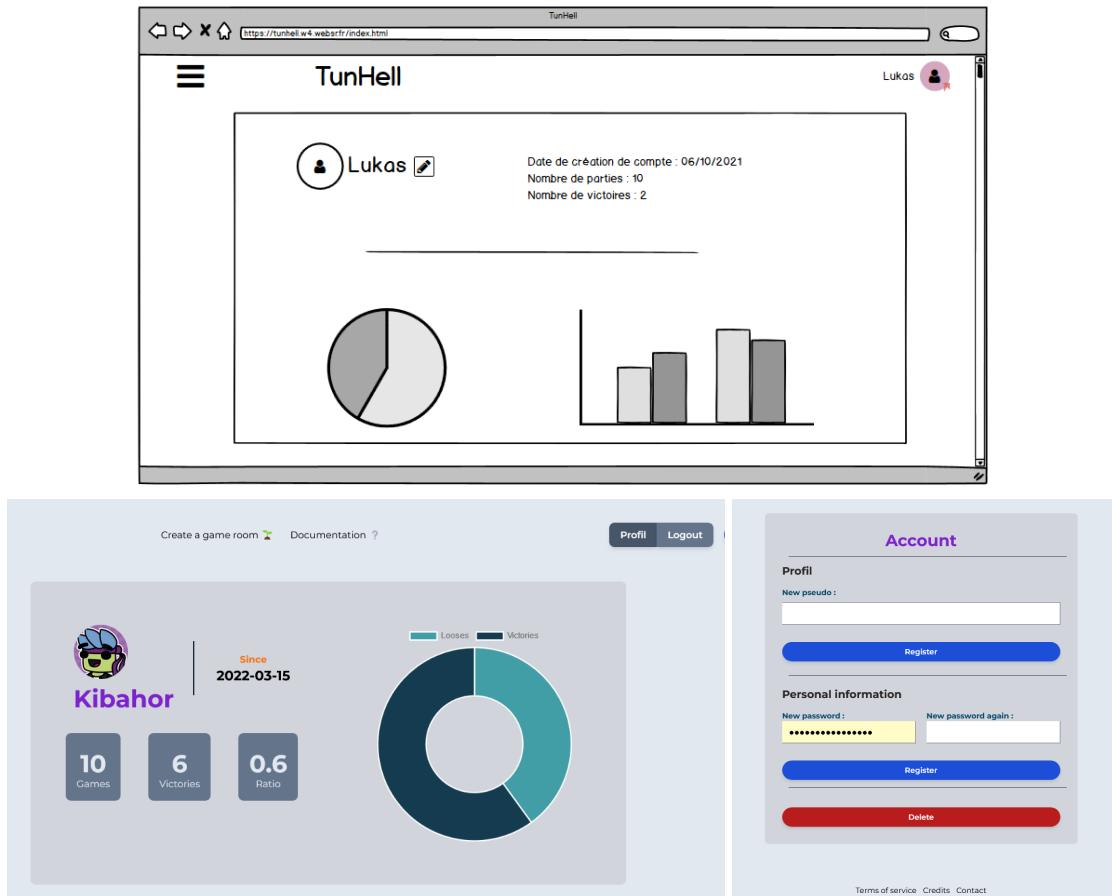


Figure 31, 32, 33 : Comparaison de la page de profil utilisateur du site avec celle du wireframe

Concernant la page relative au profil, les statistiques sont visibles par le biais de données et d'un diagramme. Si le nombre de parties est nul, le diagramme est inexistant. En revanche, sur la même page, il existe un onglet "Account" qui permet à un utilisateur de modifier son profil.

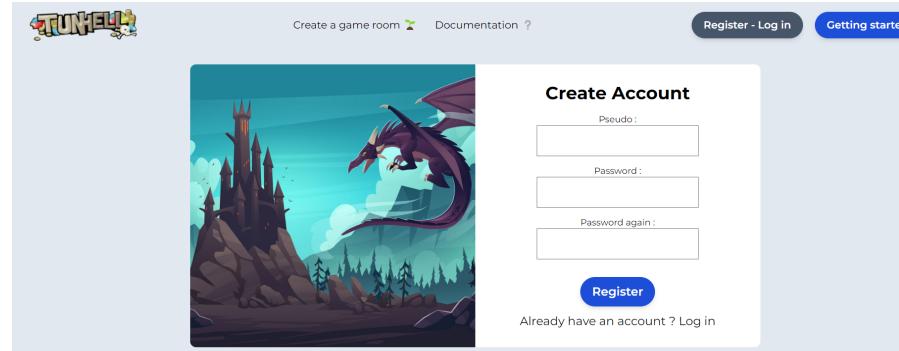


Figure 34, 35 : Comparaison entre le wireframe et le rendu finale sur la page de login

Enfin, la dernière grosse partie présente sur le diagramme de cas d'utilisation est la partie relative à la connexion. Là, en revanche, entre les attendus et les réalisations, les formulaires ont été bien pensés et mis à part la mise en page sont identiques en tous points.

### 2.3.3 Contraintes

La contrainte principale fut la gestion du responsive, la première version du site, (cf figure 19, 20 & 21) n'est pas adaptée pour les smartphones et est réalisée sans Tailwindcss. L'utilisation de ce framework nous permettrait de gagner en temps de développement avec les breakpoints fournis par le framework.

Une autre contrainte fut la contrainte juridique. En Europe, nous avons le RGPD qui nous oblige à informer les utilisateurs des informations collectées ainsi que des cookies. Pour notre part, afin d'être en règle, nous ne stockons nos informations que sur les bases de données de notre hébergeur.

### 3. Communication / serveur (page web)

#### 3.1 Technologies

Il ne restait plus qu'à trouver le moyen de communiquer entre le serveur et le client, cette partie a été la plus dure à résoudre. Il nous fallait un moyen d'acheminer les informations provenant des classes métier (Back-end) jusqu'au client (Front-end) et de plus nous devions avoir un système permettant de gérer plusieurs parties de 4 joueurs.

Le projet étant déjà assez conséquent, nous nous sommes dirigés vers le framework *JavaScript* nommé *Colyseus*. Ce framework nous permet ainsi de gérer plus facilement les différentes parties avec un système de codes de parties , mais facilite aussi la communication avec le client.

#### 3.2 Conception

Le framework Colyseus est un framework qui permet de créer des "salles" dans les jeux en JavaScript/TypeScript pour permettre la création des parties multijoueurs, pour cela nous avons besoin d'un serveur dédié aux jeux pour qu'il puisse traiter les différentes connexions et différents utilisateurs. N'ayant pas eu l'occasion de réaliser cette partie du projet, nous ne pourrons pas vous en expliquer davantage.

#### 3.3 Contrainte

Le but initial dans ce projet était la création des salons multijoueur, nous avons rencontré différents problèmes lors de notre tentative d'implémentation de Colyseus dans le projet. Dans cette tentative de communication entre Colyseus et le serveur web nous avons rencontré des problèmes de compréhension par rapport au fonctionnement de Colyseus et nous ne sommes pas parvenus à les faire communiquer.

Nous avons commencé à nous documenter pour apprendre à utiliser Colyseus et une fois que nous avons approfondi le sujet, nous nous sommes rendus compte que nous avions sous-estimé la complexité de Colyseus. Commencer son implémentation à une semaine de l'échéance du projet est irréalisable et nous manquons de temps pour l'implémenter ou chercher des alternatives.

## 4. Pentesting<sup>12</sup>

Une fois que le site internet était prêt pour le déploiement sur notre serveur web (notamment la partie PHP) nous avons effectué quelques tests de sécurité avec les vulnérabilités les plus communes dans les serveurs web.

Nous avions donc préparé une liste de 17 attaques potentielles dont 9 ont été testées. La plupart de ces attaques ont échoué, ce qui montre que notre site web était bien protégé sauf pour deux attaques majeures.

### 4.1 Injection HTML

Notre site était vulnérable aux injections HTML<sup>13</sup>, comme nous pouvons voir ci-dessous lorsque l'on envoie une balise HTML dans le formulaire de connexion et que nous envoyons la requête de connexion. La page interprète la balise HTML ce qui a pour effet d'ajouter un élément non voulu sur le site.

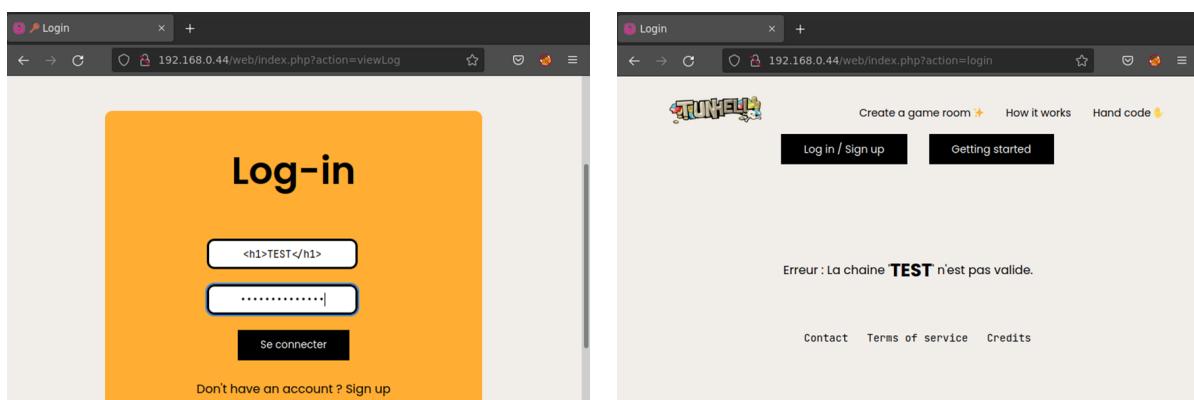


Figure 36 & 37 : Captures d'écrans montrant l'injection HTML

Si cette faille en elle-même n'est pas très dangereuse, elle peut être signe d'une faille XSS (Cross-Site Scripting)<sup>14</sup> qui est beaucoup plus dangereuse.

### 4.2 XSS (Cross-Site Scripting)

Nous avons donc essayé d'utiliser cette faille pour injecter du code JavaScript :

<sup>12</sup> **Pentesting**, signifiant “test de pénétration” est une simulation d’attaque informatique consistant, par un ensemble d’outils et de méthodes, d’évaluer la sécurité d’un système informatique.

<sup>13</sup> **Une injection HTML** est une faille de sécurité consistant à mettre du code HTML dans champs pouvant accepter du texte, afin qu'il soit interprété par le site. Cela a pour effet d'ajouter des éléments non voulus au site.

<sup>14</sup> **XSS (Cross-Site-Scripting)** est une faille de sécurité qui permet à un attaquant d'injecter dans un site web un code malveillant. Ce code sera alors exécuté par les victimes et permet aux attaquants de contourner les contrôles d'accès et d'usurper l'identité des utilisateurs.

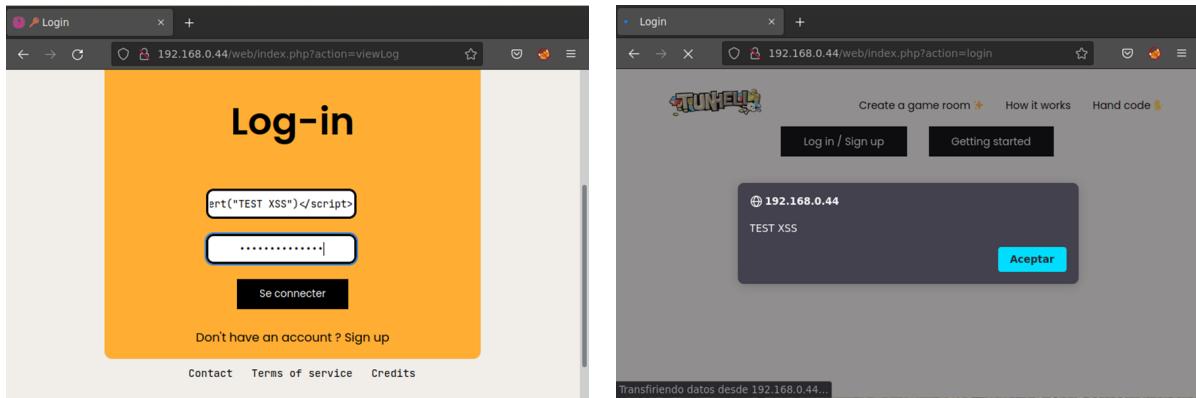


Figure 38, 39 : Captures d'écran montrant une injection de script

Comme nous pouvons le voir ci-dessus, notre site est bel et bien vulnérable aux attaques XSS. En effet, nous avons réussi à injecter un script qui affiche une popup avec pour contenu "TEST XSS" alors qu'il n'est pas présent de base. Cette vulnérabilité est très dangereuse, car dans notre exemple nous avons essayé de recevoir des informations des autres connexions / utilisateurs en utilisant le lien de la page d'erreur.

#### **Voici un exemple d'une personne voulant exploiter la faille XSS :**

Tout d'abord nous créons un serveur HTTP sur une machine virtuelle en écoute sur le port 8080 avec python.



Figure 40 : Capture d'écran montrant le démarrage du serveur HTTP

Une fois le serveur lancé nous allons injecté ce script sur la page de login:

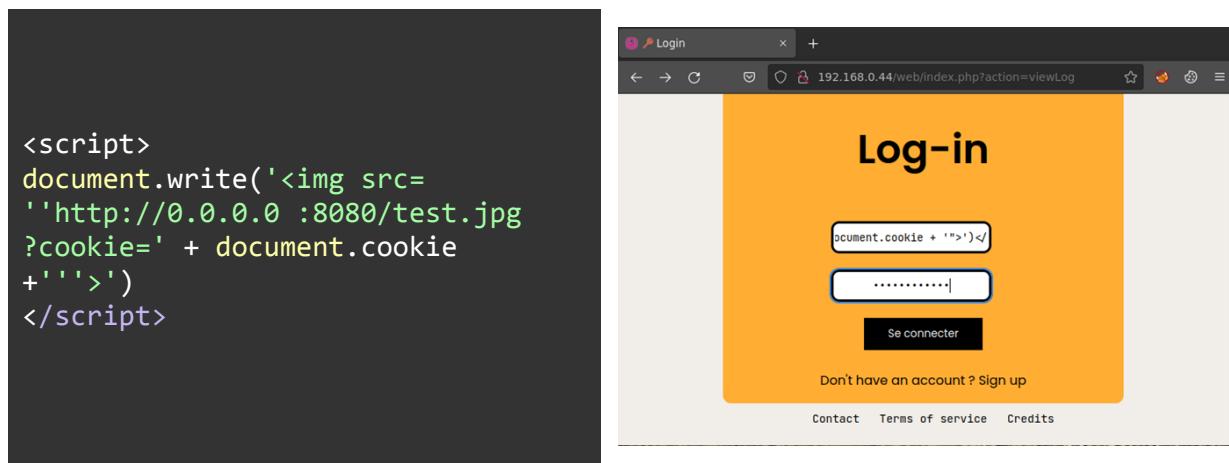


Figure 41, 42 : Script injecté + Capture d'écran montrant comment le script a été injecté

Ce script aura pour effet de demander à notre site internet d'aller chercher, sur notre serveur HTTP précédemment créé, une image qui n'existe pas. Ce qui a pour conséquence d'envoyer une commande GET sur notre serveur et comme dans la requête on envoie aussi les cookies, notre serveur recevra les cookies de l'utilisateur.

Une fois que nous avons envoyé cette requête nous pouvons retourner sur le serveur pour voir s'il reçoit bien la requête.

```
> python -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
127.0.0.1 - - [24/Feb/2022 22:36:39] code 404, message File not found
127.0.0.1 - - [24/Feb/2022 22:36:39] "GET /test.jpg?cookie=PHPSESSID=uhfuk7v6
693akn630g424221mu HTTP/1.1" 404 -
```

Figure 43 : Capture d'écran des logs du serveur HTTP

Et comme nous pouvons le voir, il reçoit les requêtes avec le cookie de session. Les cookies qui ont pu être récupérés permettront d'usurper l'identité de la personne à qui l'on vient de récupérer les cookies.

Nous rechargeons la page d'erreur et notre serveur reçoit à nouveau la requête, mais si nous envoyons cette page d'erreur sur un autre navigateur mais que la page d'erreur ne s'affiche plus, nous pourrons nous assurer que la page web ne sauvegarde pas ces pages d'erreur.

Une fois avoir identifié le problème détecté dans notre code, nous l'avons corrigé en changeant la page d'erreur pour qu'elle n'affiche plus les données, après une tentative d'attaque. Une fois fait, si on réessaie , on remarque bien que l'attaque échoue, nous avons donc corrigé la faille et sécurisé notre site contre ce type d'attaques.

## V/ Bilan technique

### 1. Site internet

#### 1.1 Hébergeur

Concernant le déploiement du site, nous avons sélectionné et loué un hébergeur web chez Webstator.

Le site est à sa deuxième version déployée et fonctionnel, vous pouvez y accéder à l'adresse suivante : **https://tunhell.w4.websr.fr**

#### 1.2 Front-End

Actuellement, nous avons un site fonctionnel avec un système de comptes. Les utilisateurs peuvent créer des comptes, accéder à leur page de profil.

Sur cette dernière, ils peuvent visualiser leurs statistiques et leur ratio, peuvent modifier leur mot de passe ou leur pseudo.

Le site est entièrement fonctionnel et responsive, il s'adapte à n'importe quelle taille d'écran. Ainsi, si nous devons faire le bilan sur les technologies utilisées pour la partie front-end, l'utilisation de Tailwindcss et de ChartJS à impacté positivement le projet, nous permettant d'être plus esthétique, rapide et responsive.

#### 1.3 Back-End

La partie Back-end nous permet actuellement de gérer le système de compte entièrement fonctionnel, et de stocker l'ensemble des entrées en base de données. Concernant la persistance, Le PHP nous permet d'appliquer le SCRUB avec l'ensemble de nos utilisateurs. En revanche concernant le système de parties en multijoueur, nous n'avons pas l'ensemble des statistiques d'une partie

Pour ce qui est relatif à notre base de données, nos deux tables Games et Account ont étaient implémentés. N'ayant pas de système de partie en multijoueurs, la table Games n'est pas exploitée jusqu'à aujourd'hui.

PHP nous à permis de réaliser un site fonctionnel ainsi qu'une gestion du côté serveur plutôt simple et fluide.

## **2. Jeu**

### **2.1 Modèle**

À l'heure actuelle, le jeu est entièrement jouable en console selon les règles officielles du jeu (trouvable sur le Site ainsi que dans les annexes). Les joueurs sont cependant obligés de jouer sur un seul et unique terminal, ce qui peut amener à des tricheries , car ces derniers sont alors capables de voir la main du joueur courant.

### **2.2 Partie graphique**

Malheureusement, la partie graphique n'a pu être menée par manque de temps. Le jeu reste toutefois parfaitement jouable et bien que cette partie n'ait pas été faite, nous disposons des ressources graphiques nécessaires à cette partie.

## **3. Communication Site Internet/Jeu**

La partie de connexion entre le site internet et le jeu n'a pas été implémentée, L'équipe a eu du retard sur la communication entre les deux parties.

Néanmoins, des recherches ont été faites dans l'objectif de simplifier et rendre plus rapide le temps de développement grâce à l'utilisation de Colyseus.

Des tests ont été faits en local, mais nous sommes confrontés à un problème dû à notre manque de connaissances du framework et des outils présents sur l'hébergeur.

## V/ Conclusion

Durant ces 2 semestres, nous avons appris de nouvelles technologies, à gérer un projet avec une équipe de 5, à gérer les dates butoir, mais aussi à gérer les obstacles tout du long de ce parcours, ainsi que résoudre et apprendre des problèmes et des conflits internes que nous avons rencontrés.

Le projet n'est pas fini, cela peut s'expliquer par le fait que le projet était dès le départ trop ambitieux, au vu du temps disponible et des conditions de travail. Nous avons sous-estimé la charge de travail, la capacité à appréhender et utiliser de nouvelles technologies ainsi que le temps nécessaire pour développer les différentes parties du jeu. Ce manque de temps peut aussi s'expliquer par notre charge de travail quotidienne, ainsi que les différents autres projets à mener à bien en parallèle.

Néanmoins, nous avons bien avancé sur le projet ; il nous a permis de toucher à de nombreux domaines de l'informatique, du réseau au développement en passant par la conception à la sécurisation d'un projet. Mais aussi d'autres domaines tels que l'UX design ou, la législation (RGPD). Il nous a aussi permis de comprendre les différents processus de développement d'un grand projet ainsi que les différents problèmes que l'on pourrait rencontrer, à l'avenir, dans notre vie professionnelle.

## VI/ Resume

Our project consists of an online browser game inspired by the Kickstarter's project called "Tunhell" in which we would create a webpage which anyone could access and where anyone could play an online and multiplayer version of the original card game.

The original card game is a card game for 2 to 4 players in which we control a team of dwarves to mine and gather resources in a medieval fantasy world. The goal of the game is to collect more resources than the other players, to achieve that the player can place the dwarves in one of the different card piles (called mines) where they will "mine" resources but also face several dangers where the player must strategically place their dwarves taking into account their unique statistics and attributes.

Our project was divided into two subprojects, the website and the game model. For the website part we made it compatible with most of the screen resolutions used by computers and with mobile devices such as smartphones and tablets, the website also has a detailed manual where we explain how to play the game, a register and login screens so people can create their own accounts to play the game and access their personal game statistics. The website is secured against the most common web attacks that could put at risk the users and the database.

For the game model side we have a game that runs in terminal, even if we didn't implement a visual interface nor the online multiplayer mode the game stills fully functional and most of the original card game mechanics are implemented.

After 2 academic periods (5 months) we didn't finish implementing the game to the website, but we finished the rest of the website with an eye-catching visual facet, a simple, effective and secured database. We also couldn't implement the online multiplayer mechanics in the game model, but we still have a solid console game that can be played by multiple players, even if they must be in the same room.

This project helped us understand the process of web and game development and the different problems we can face. We learned how to overcome those issues and solve them, as well as how to organize a team and how to structure work, so we can advance a project effectively. Now we know a bit more about how web games work and all the facets of its development.

# VII/ Webographie

## 1. Langages, Frameworks & Interpréteurs

### 1.1 Langages

JavaScript : <https://developer.mozilla.org/fr/docs/Web/JavaScript>

TypeScript : <https://www.typescriptlang.org>

PHP : <https://www.php.net>

MariaDB : <https://mariadb.org>

### 1.2 Frameworks

Colyseus : <https://www.colyseus.io>

matter.js : <https://brm.io/matter-js/>

phaser.js : <https://phaser.io>

melon.js : <https://www.melonjs.org>

KorGE : <https://korge.org/>

Github Collection - JavaScript-game-engines :

<https://github.com/collections/javascript-game-engines>

Chart JS : <https://www.chartjs.org/docs/latest>

Tailwind : <https://tailwindcss.com>

### 1.3 Interpréteurs & Interfaces de gestion

NodeJS : <https://nodejs.org/fr/>

phpmyadmin : <https://www.phpmyadmin.net>

### 1.4 Logiciels

Visual Studio Code : <https://code.visualstudio.com>

Atom : <https://atom.io>

Balsamiq : <https://balsamiq.com>

SQLMap : <https://sqlmap.org>

Draw.io : <https://draw.io>

StarUML : <https://staruml.io>

Parrot OS (suite de logiciel de pentesting) : <https://www.parrotsec.org>

## 2. Auto Formation

Le Designer du Web - Codez des graphiques facilement avec Chart.js ! :

<https://www.youtube.com/watch?v=NSCJ9jIUnSI>

Chaine Youtube Net Ninja :

<https://www.youtube.com/channel/UCW5YeuERMmlnqo4oq8vwUpg>

Raddy - Tailwind CSS v.3.0 Crash course :

<https://www.youtube.com/watch?v=V5HZOmkPSs>

Alsacreation - Tricks css : <https://www.alsacreations.com/>

CSS Tricks - Tutoriel: <https://css-tricks.com/>

Graphikart : <https://grafikart.fr>

Exploit DB : <https://www.exploit-db.com>

### 3. Hébergeurs

Webstrator : <https://webstrator.fr>

OVH : <https://www.ovhcloud.com/fr/>

### 4. Inspirations

TunHell (cartes + règles) : <https://www.kickstarter.com/projects/1655335494/tunhell>

Projet de PHP de Mathieu Morel et Mathis Ribémont :

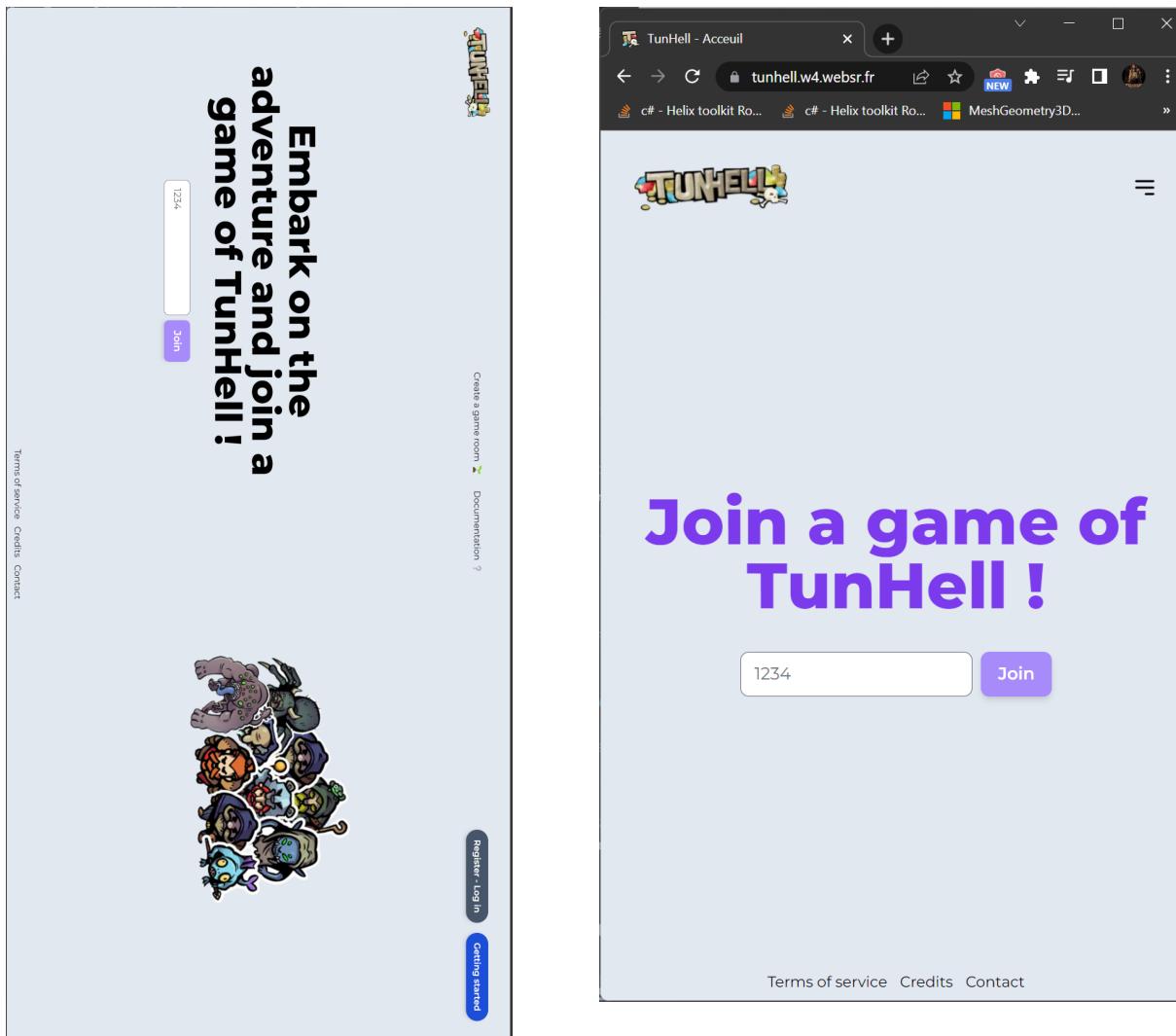
<https://gitlab.iut-clermont.uca.fr/maribemont/toutdouxliste>

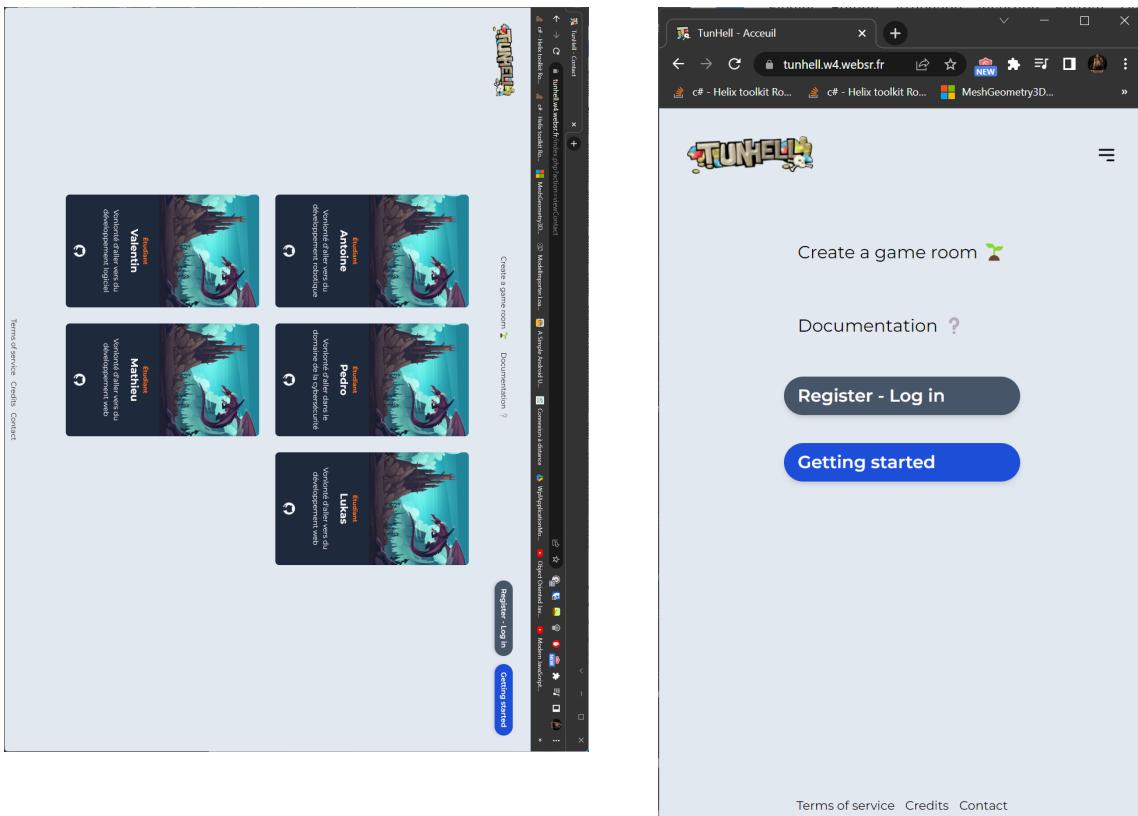
### 5. Autres

Cnil (Commision national de l'informatique et des libertés) : <https://www.cnil.fr/>

# VIII/ Annexes

## Site Internet





**Website Terms and Conditions of Use**

**1. Terms**

By accessing the Website you are agreeing to be bound by these Website terms and conditions of use. No part of the materials on this site may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of the author.

The materials contained in this Website are protected by copyright and trademark law.

**2. Use license**

Permission is granted to temporarily download one copy of the materials on TunHell's Website for personal, non-commercial viewing only. This is the grant of a license, not a transfer of title, and under this license you may not:

- modify or copy the materials;
- use the materials for any commercial purpose or for any public display;
- attempt to reverse engineer any software contained on TunHell's Website;
- remove any copyright or other proprietary notices from the materials or software;
- transfer or make available the materials to another person or "mirror" the materials on any other server;
- attempt to permanently download the materials on TunHell's Website or any portion thereof for permanent storage;
- attempt to circumvent any restrictions of any of these restrictions; Upon termination, you will let TunHell to terminate upon violation of any of these restrictions. Upon termination, all rights to the materials in your possession, whether in printed or electronic format, these Terms of Service has been created with the help of the Terms Of Service Generator.

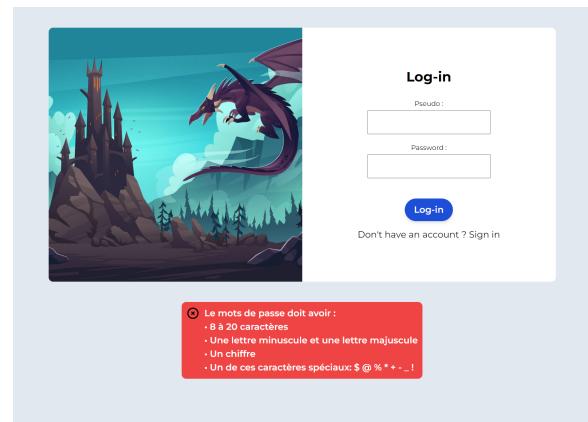
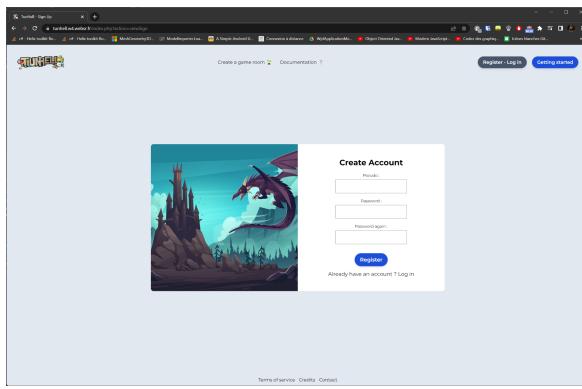
**3. Disclaimer**

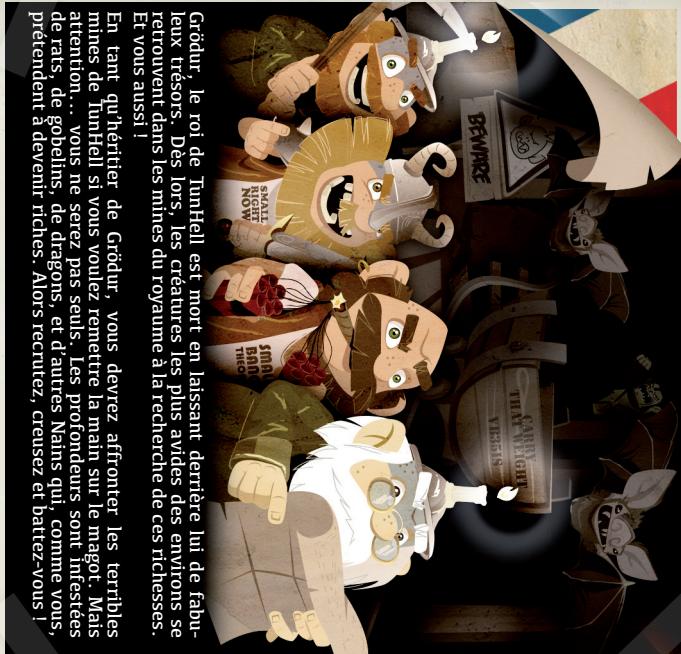
All the materials on TunHell's Website are provided "as is". TunHell makes no warranties, may be expressed or implied, therefore negates all other warranties. Furthermore, TunHell does not make any representations concerning the accuracy or reliability of the use of the materials on its Website or otherwise relating to such materials or any sites linked to this Website.

**4. Limitations**

TunHell or its suppliers will not be held accountable for any damages that will arise with the use or inability to use the materials on TunHell's Website, even if TunHell or an authorize representative of this Website has been notified, orally or written, of the possibility of such damage. Some jurisdiction does not allow limitations on implied warranties or limitations of liability for incidental damages, these limitations may not apply to you.

**5. Revisions and Errata**





## INTRODUCTION

Grödur, le roi de TunHell est mort en laissant derrière lui de fabuleux trésors. Des lors, les créatures les plus avides des environs se retrouvent dans les mines du royaume à la recherche de ces richesses. Et vous aussi !

En tant qu'héritier de Grödur, vous devrez affronter les terribles mines de TunHell si vous voulez remettre la main sur le magot. Mais attention... vous ne serez pas seuls. Les profondeurs sont infestées de rats, de gobelins, de dragons, et d'autres Nains qui, comme vous, prétendent à devenir riches. Alors recrutez, creusez et battez-vous !

TunHell est un jeu qui se joue de 2 à 4 joueurs. Chaque joueur est à la tête d'une équipe de nains et va tenter d'amasser plus de richesses que ses adversaires. À votre tour, vous pourrez recruter de nouveaux Nains puis les envoyer creuser dans la mine pour y dénicher des trésors et combattre des ennemis qui vous rapporteront des points de victoires. Le joueur qui en obtiendra le plus sera déclaré vainqueur !

La règle est divisée en deux parties : une règle d'initiation qui vous permettra de commencer à jouer rapidement et de vous familiariser avec le système de jeu ; et le jeu complet. La règle du jeu complet est identique, mais introduit de nouveaux éléments. Il s'agit du vrai jeu !

## MATÉRIEL

	• 44 cartes «Nain»
	• 60 cartes «Mine»
	• 4 cartes «Fin de Mine»
	• 2 cartes «Trophée»

• La présente règle

## RÈGLES D'INITIATION

### MISE EN PLACE



1- Commencez par éarter les cartes «fin de mine», «trophée» et toutes les autres cartes «Mine» portant le symbole (+) en bas à gauche. Ces cartes sont remises dans la boîte et ne seront pas utilisées dans les règles d'initiation.

2- Écartez aléatoirement 7 cartes «Mine» supplémentaires que vous rangez également dans la boîte. Ceci permet d'avoir des configurations différentes à chaque partie.

3- Constituez 3 paquets égaux de cartes «Mine» face cachée au milieu de la table. Disposer-les de manière à ce que chaque joueur puisse poser ses cartes de chaque côté des paquets. Ces tas représentent les 3 mines que vous pourrez explorer (voir exemple page suivante).

4- Mélangez les cartes «Nain» et distribuez-en 4 face cachée à chaque joueur.

5- Disposez le reste du paquet sur le côté, face visible et révélez les 5 premières cartes en constituant une ligne. Ces 5 cartes représentent le «centre de recrutement».

6- Prévoyez une défausse pour les cartes «Nain» et une autre pour les cartes «Mine». La défausse est l'endroit où sont posées les cartes jouées au fil du jeu. Nous vous conseillons de défausser les cartes face cachée afin de les distinguer des cartes en jeu.

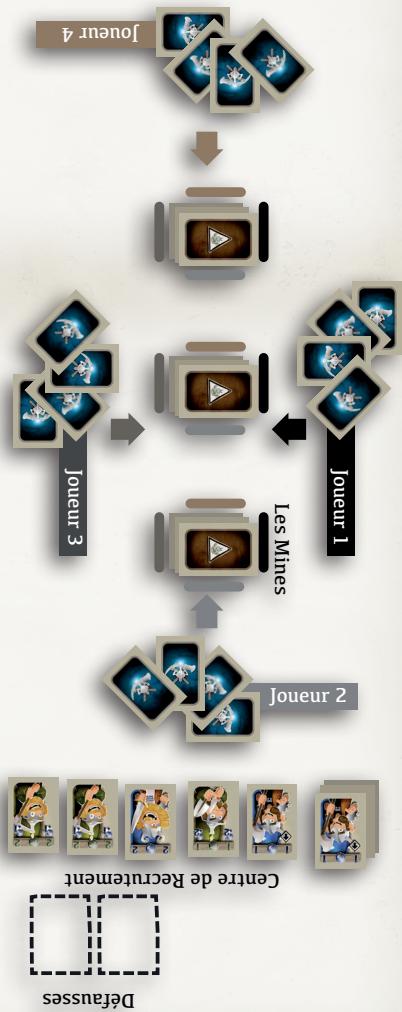
Le joueur ou la joueuse ayant la plus longue barbe commence, puis on joue chacun son tour dans le sens horaire.

## EXEMPLE DE MISE EN PLACE

### POUR 4 JOUEURS

Chacun des côtés des cartes «Mines» appartient à un joueur. Il faudra prévoir un espace suffisant pour pouvoir poser des cartes entre chaque mine.

À 3 joueurs, un côté reste neutre.  
À 2, les joueurs jouent face à face.



## TOUR DE JEU

Lors de son tour de jeu, un joueur doit faire une action parmi les 2 possibles :

- A - recruter un Nain

OU

- B - jouer un Nain et appliquer son effet.



Seules les cartes portant le symbole ci-contre font exception et doivent être posées au moment où elles sont recrutées.

Une fois l'action réalisée, c'est au tour du joueur suivant.

### A - RECRUTER UN NAIN

Un joueur peut choisir de recruter un Nain. Il fait alors son choix parmi les 5 cartes «Nain» disponibles dans le centre de recrutement et l'ajoute à sa main. La carte choisie est remplacée par la première carte de la pile afin d'avoir toujours 5 cartes disponibles sur la table. Un joueur peut avoir au maximum 6 cartes en main. Il ne peut donc pas effectuer cette action s'il a déjà atteint ce maximum.

Lorsque la pile est vide, on mélange à nouveau les cartes «Nain» qui ont été défaussées pour en former une nouvelle.

Attention : Le joueur **ne peut jamais** prendre la première carte de la pile de cartes «Nain».

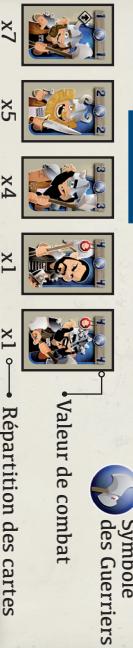


### B - JOUER UN NAIN

### JOUER UN GUERRIER

Un joueur peut poser une et une seule carte «Nain» face visible à son tour de jeu. Il existe 4 types de «Nain» : Guerrier, Piocheur, Éclaireur ou Dynamiteur. Ils se différencient par leur couleur et le symbole en haut de la carte. Chacun a un effet particulier.

### SYNTHÈSE



Symbol des Guerriers

Valeur de combat

Répartition des cartes

Un Guerrier permet :

- de combattre des ennemis ;
- et de compliquer l'accès d'une mine à ses adversaires.

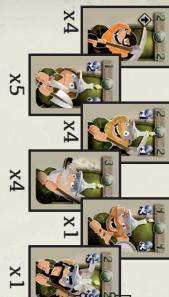
Lorsqu'un joueur joue un Guerrier, il le pose face visible, de son côté (voir exemple de mise en place), devant la mine de son choix. Sa valeur de combat s'ajoute à celles de ses autres Guerriers présents devant cette même mine s'il y en a déjà.





Les Guerriers qui portent ce symbole sont fiers, ils n'ont besoin de personne ! Si le joueur qui pose une de ces cartes avait d'autres Guerriers devant cette mine, ils sont aussitôt défaussés. Le joueur ne pourra pas ajouter d'autres Guerriers à cette mine tant que cette carte sera présente.

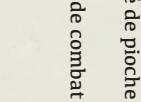
## JOUER UN PIOCHEUR



Symbol des Piocheurs  
x3



• Capacité de pioche  
Bonus de combat

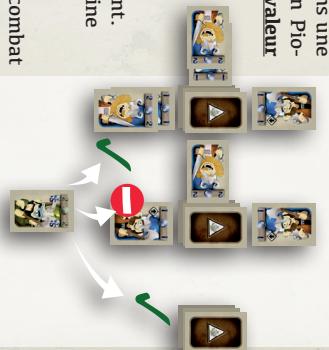


• Valeur d'exploration  
x2  
x3

Symbol des Éclaireurs  
x3



Le Piocheur permet de creuser dans une mine. Un joueur ne peut poser un Piocheur dans une mine que si la valeur de combat de ses Guerriers présents sur cette mine est au moins égale à celle des Guerriers adverses présents dans cette même mine. On compare avec chaque autre joueur individuellement. Il peut bien sûr être posé sur une mine où il n'y a aucun guerrier.



**⚠️ Attention :** Ici, le bonus de combat du Piocheur ne s'ajoute pas à celle des Guerriers. (Voir exemple).

Le Piocheur peut alors piocher dans cette mine autant de cartes que sa capacité de pioche le permet tant que rien ne l'en empêche (voir chapitre suivant : «dans la mine !»). Les cartes sont révélées une à une face visible à l'ensemble des joueurs et les événements résolus au fur et à mesure.

**Le Piocheur est toujours défaussé à la fin du tour.**

## JOUER UN ÉCLAIREUR

Symbol du Dynamiteur  
x3



Le Dynamiteur permet de regarder - selon sa valeur d'exploration - les 3 ou 5 premières cartes d'une des 3 mines, sans les dévoiler à ses adversaires. Un joueur peut choisir n'importe quelle mine sans avoir à tenir compte des Guerriers présents. Les cartes sont ensuite reportées dans le même ordre, face cachée, à leur place initiale. S'il reste moins de cartes dans la mine que la valeur d'exploration de l'Éclaireur, la différence est perdue et n'est pas reportée sur une autre mine. **L'Éclaireur est défaussé à la fin du tour.**

## JOUER UN DYNAMITEUR



Symbol du Dynamiteur  
mine de son choix. Ils sont tous défaussés.

**⚠️ Attention :** Tous les Guerriers sont défaussés, même ceux du joueur qui a posé le Dynamiteur.

**Le Dynamiteur est défaussé à la fin du tour.**

## DANS LA MINE !

Lorsqu'un Piocheur creuse, il peut trouver des trésors, des ennemis ou encore ne ramasser que de la terre !

### LA TERRE

Lorsqu'un Piocheur retourne une carte «Terre», il ne se passe rien... Le joueur récupère la carte et la pose devant lui. Les cartes ainsi ramassées constituent le butin du joueur. Le joueur continu de piocher si la capacité de son piocheur le lui permet encore.

**x18**

### LES TRÉSORS

Lorsqu'un Piocheur découvre un trésor, le joueur le récupère et la pose devant lui dans son butin. Un trésor vaut 1 point de victoire en fin de partie. Le joueur continue de piocher si la capacité de son piocheur le lui permet encore.

## LES RENCONTRES

Symbol  
des Rencontres

Rat	Gobelin	Orc	Troll	Dragon	Valeur de combat
x7					
x4					
x4					
x3					
x3					
					→ Points de victoire

Lorsqu'un Piocheur rencontre un ennemi, il y a obligatoirement affrontement. Pour vaincre un ennemi, un joueur doit défausser, parmi ses Guerriers posés devant cette mine, des cartes dont les valeurs de combat également au moins la valeur de combat de l'ennemi (pour un Dragon qui vaut 6 en combat, il faudra, par exemple, se défausser de 2 Guerriers valant 3).

Il peut aussi utiliser le bonus de combat du Piocheur. Mais dans ce cas, Le Piocheur est défausse et ne pourra plus continuer à piocher. Le joueur est obligé de combattre un ennemi qu'il rencontre et de le vaincre s'il le peut.

• Si l'ennemi ne peut être vaincu, le joueur actif doit défausser toutes ses cartes «Nain» (Guerriers et Piocheurs) posées devant cette mine et l'ennemi est remis à sa place face cachée. C'est au joueur suivant de jouer.

• Si l'ennemi est vaincu, le joueur récupère la carte et la pose devant lui dans son butin. Certaines rencontres donnent des points de victoire. Si le Piocheur n'a pas été défausssé lors de la rencontre, il continue de piocher si sa capacité de pioche le lui permet encore.

⚠ Attention : Si après avoir pioché la dernière carte d'une des mines il reste des Guerriers devant cet emplacement vide, ils sont défausssés.

**FIN DU JEU**

Le jeu prend fin immédiatement lorsque 2 des 3 tas de cartes «Mine» sont vides.

# JEU COMPLET



Le jeu complet se joue exactement de la même manière que la version d'initiation. La principale différence est l'ajout de cartes que les joueurs pourront ramasser en piochant dans les mines. Parmi ces «Trouailles», il y a des bonus qui aideront les joueurs dans leurs explorations, des trésors particuliers, mais aussi le fantôme du roi Grôdr qui, lui, ne vous fera pas de cadeau ! De plus, au fond de chaque mine se trouve maintenant une carte spéciale qui pourra être un monstre terrifiant, une salle hantée, ou encore une petite porte bien dissimulée dans l'ombre...

## MISE EN PLACE

1- Mélangez toutes les cartes «Mine» et écartez-en 9 au hasard. Mélangez les cartes «Fin de Mine» et écartez-en 1. Ces cartes ne seront pas utilisées pour cette partie et sont remises dans la boîte.

2- Constituez 3 paquets égaux de cartes au milieu de la table en répartissant les cartes «Mine» sur chacune des cartes «Fin de Mine». Ces tas représentent les 3 mines dans lesquelles les joueurs pourront engager leurs Nains.

3- Placez les 2 Cartes «Triophées» sur le côté de l'aire de jeu.

Le reste de la mise en place est identique à la règle d'initiation à deux exceptions près :

**TOUR DE JEU**

Le tour de jeu est en tout point identique à la règle d'initiation à deux exceptions près :

Chaque joueur fait alors le total des points de victoire des cartes de son butin. Celui qui a le plus de points de victoire est déclaré grand vainqueur. Enfin grand...

• Le premier joueur à ramasser de la terre obtient le trophée «Employé du Mois» (il a bien travaillé). Il le conserve devant lui tant qu'il a plus de carte «Terre» dans son butin que les autres joueurs. Dès qu'un autre joueur obtient autant ou plus de «Terre», il s'empare alors du trophée.



- Le premier joueur à terrasser un Rat obtient **le trophée «Deratiseur»**. Il le conserve devant lui tant qu'il a plus de carte «Rat» dans son butin que les autres joueurs. Dès qu'un autre joueur obtient autant ou plus de «Rat», il s'empare alors du trophée.

## DANS LA MINE

Les cartes «Terre», «Trésors» et «Rencontres» sont résolues comme dans la règle d'initiation, mais dorénavant les joueurs pourront en plus faire des trouvailles dans les mines et récupérer des objets qui rejoindront leur main.

Ce symbole signifie que la carte une fois piochée rejoue la main du joueur. Si ce joueur a déjà 6 cartes en main, la Trouvaille est donnée au joueur suivant dans l'ordre de tour et ainsi de suite. Si tous les joueurs ont déjà 6 cartes en main, la Trouvaille est alors défaussée.

### LES TROUVAILLES :

#### MACHIN

 Il s'agit en fait de retrouvailles ! Machin était perdu dans Tun-hell depuis si longtemps qu'on a oublié comment il s'appelait... Mais maintenant il connaît les mines comme sa poche !

Il est tellement content de vous revoir qu'il est prêt à vous aider ! Il rejoue votre main et pourra être joué par la suite comme n'importe quel autre Piocheur.

#### DWOUARF LE CHIEN

 Vous trouverez un chien errant. Il est sympa, vous décidez de l'appeler Dwouarf. Il vous débarrassera facilement des Rats ! Il rejoue votre main et pourra être joué par la suite comme n'importe quel autre Guerrier.

x1 Il n'a aucune valeur de combat mais il donne un malus (-1) en combat aux cartes «Rencontre». Ainsi les Rats valent 0 et sont éliminés directement, les Gobelins valent 1, etc... Il n'est jamais défaussé lorsqu'une rencontre est vaincue, mais il est défaussé comme les autres Guerriers en cas de défaite. Le dynamiteur le fait fuir également.

#### LES BONUS :

Les 3 cartes suivantes devront être posées en même temps qu'un Piocheur et lui conféreront un bonus. Il s'agit des seuls cas où l'on peut jouer plusieurs cartes en même temps. Les Trouvailles et leurs avantages sont cumulables.

#### NAWAK, L'ÉPÉE DE BRAVOUR

 Cette épée sacrée donne un bonus en combat (+2) au Nain qui la porte.

x1

#### LA VIEILLE PIOCHE

Rien de sacré, mais toujours efficace ! elle donne un bonus en pioche (+2) au Nain qui la porte.

x1

#### LA BIÈRE DU COURAGE

 Cette divine boisson déuple les capacités du Piocheur qui la boit (x2 à sa capacité de pioche et x2 à son bonus de combat).

x2

#### LES 2 ANNEAUX UNIQUES

 Cette relique est tellement rare qu'on a préféré en forger 2 au cas où... Les Anneaux rapportent 2 points de victoire en fin de partie mais ils sont trahis, ils rejouent obligatoirement votre main et non votre butin !

x2 Attention : Votre limite de cartes en main est toujours de 6 cartes maximum.

### LES TRÉSORS SPÉCIAUX :

## LE COEUR D'OR

Le Roi Grödür avait beau être cruel et avide, il avait un cœur d'or. D'ailleurs, il le gardait jalousement au centre de la mine... C'est un trésor qui rapporte 3 points de victoire, mais il est lourd. Pour le ramener vous devrez vous défaussez d'un autre Piécheur issu de votre main. Le Coeur d'Or est alors placé sur votre butin. Si vous ne pouvez pas, il est remis à sa place face cachée et votre tour s'arrête immédiatement.



x1

## LE FANTÔME DE GRÖDUR :

Une apparition du Fantôme effraie vos Nains !

Le joueur à votre gauche défausse une carte «Nain» tirée de votre main au hasard. Si vous n'avez pas de carte «Nain», alors on en défausse une de la main du joueur suivant et ainsi de suite. Si personne n'a de carte «Nain» dans sa main (déjà, c'est louche...) rien ne se passe.



x2

Le Fantôme est ensuite défausonné.

## AU BOUT DE LA MINE...

La dernière carte de chaque pile est une carte spéciale «Fin de Mine».

## LA PORTE DE DERRIÈRE

Vous trouvez une porte dérobée qui vous permet d'entrer dans une mine en toute discrétion ! Le joueur qui la découvre pose immédiatement cette carte face à une nouvelle mine. À partir de ce moment il pourra jouer ses Piécheur sur cette mine sans tenir compte de la valeur des Guerriers adverses.

⚠ Attention : S'il reste des cartes à piocher au joueur qui découvre la porte de derrière, il les prend immédiatement dans la mine devant laquelle il aura posé la porte.

La porte ne pourra plus être déplacée.  
La porte rapporte 1 point de victoire au joueur qui l'a découverte.



## LA SALLE DU TRÔNE

On y trouve le squelette du roi Grödür, sa couronne et son sceptre. La carte rejoint le butin du joueur qui la découvre et rapporte 3 points de victoire. Mais cette salle est maudite ! Le joueur doit se défausser de toutes les cartes de sa main.



## LA GROSSE ARACHIDE POULE

Avec ses 8 pattes velues, elle vous attend tapie dans l'ombre. C'est une carte «Rencontre» dont la valeur de combat est 5 et qui rapporte 5 points de victoire.

## LE DEMON DES PROFONDEURS

Née de la rencontre d'un lézard et d'une vache, cette créature maléfique se cache tout au fond de la mine. C'est une carte «Rencontre» dont la valeur de combat est 7 qui rapporte 7 points de victoire.



## FIN DU JEU

Comme dans la règle d'initiation, le jeu prend fin immédiatement lorsque 2 des 3 paquets de cartes «Mine» sont vides.

Chaque joueur fait alors le total des points de victoire des cartes de son butin, éventuellement des trophées, des Anneaux de sa main et de la Porte dérobée qu'il a obtenue. Celui qui a le plus de points de victoire est déclaré vainqueur.

## CONSEILS

• L'exemple de mise en place est indicatif mais vous pouvez placer les mines comme vous le voulez sur la table pourvu que chacun ait la place de poser ses cartes.

• Essayez de gérer correctement vos valeurs de combat !

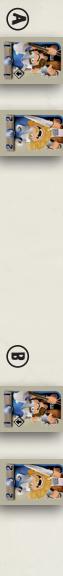
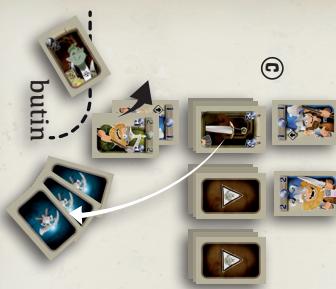
Défausser un Guerrier «3» pour vaincre un simple rat n'est souvent pas rentable !

• Vous pouvez laisser votre butin face cachée pour maintenir le suspens jusqu'à la fin du jeu, mais les cartes «Terre» et «Rat» doivent rester visibles pour que chacun puisse vérifier qui en a le plus.

• Vous pouvez raccourcir la durée du jeu en otant un peu plus de cartes «Mine» en début de partie. Assurez-vous juste que chaque tas contienne le même nombre de cartes «Mines». Selon vos envies, il faudra enlever 3, 6 ou 9 de plus que le nombre indiqué dans la mise en place.

## EXEMPLE DE TOURS DE JEU A 2 JOUEURS

**1** A- Le premier joueur (en bas) joue un Piocheur avec une capacité de pioche de 2 sur une mine.



B- La première carte qu'il dévoile est un Gobelins qui a une valeur de combat de 2. Le joueur choisit alors de défausser son Guerrier de valeur 3 pour vaincre le Gobelins (il aurait pu choisir son Guerrier 1 et son Piocheur).

Celui-ci rejoint le butin du joueur.



**2** Le deuxième joueur joue à son tour un Piocheur avec une capacité de pioche de 2 et dévoile un Orc qui a une valeur de combat de 3.

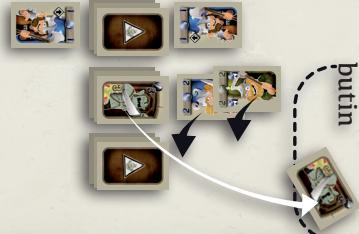
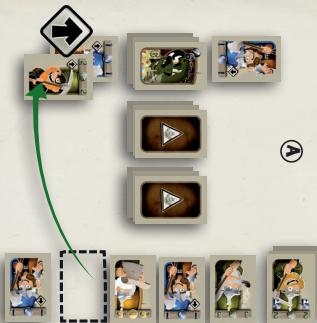
Il est alors obligé d'utiliser son Guerrier de valeur 2, et d'y ajouter le bonus de combat de son Piocheur pour obtenir une valeur de combat suffisante.

Ces cartes sont défaussées, et l'Orc rejoint le butin du joueur.

Puisque le Piocheur a été défaussé, il ne peut pas continuer à creuser.

C'est au tour du joueur suivant.

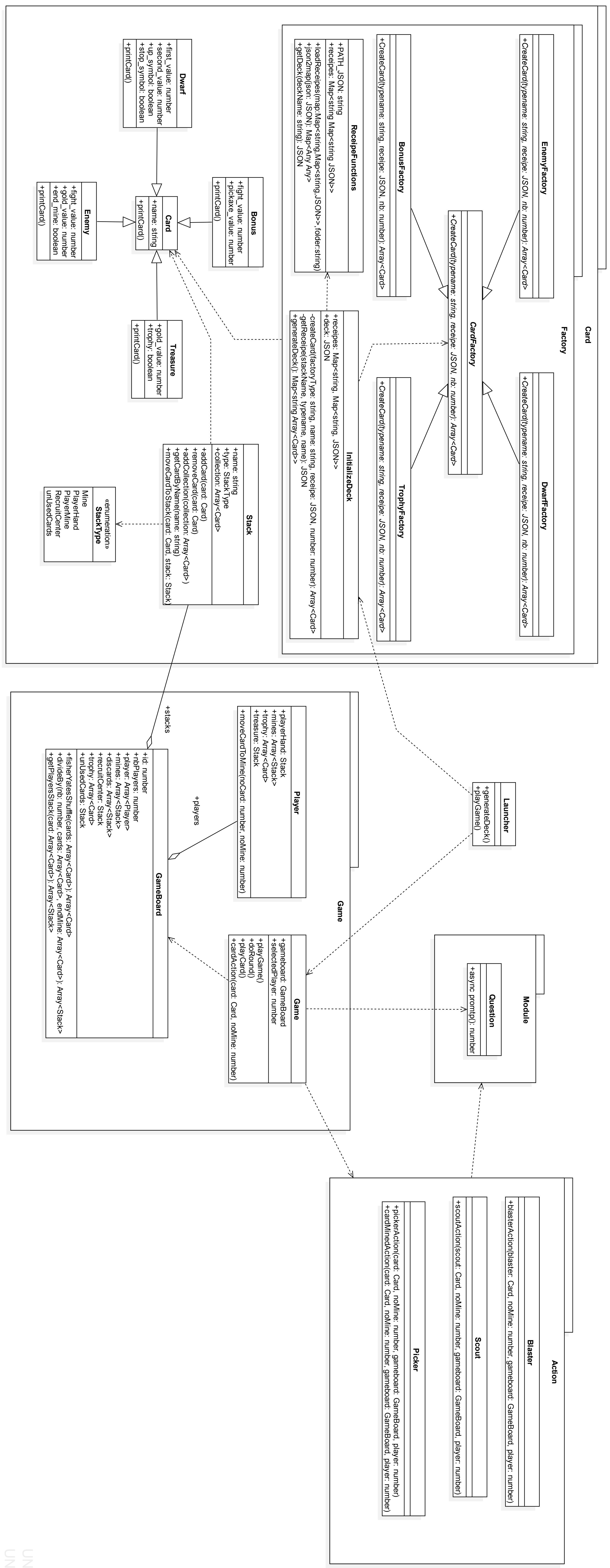
**3** A- C'est de nouveau au tour du premier joueur. Cette fois, il recrute et joue immédiatement un Piocheur portant le symbole ♦ avec une capacité de pioche de 2 et dévoile un Dragon qui a une valeur de combat de 6.



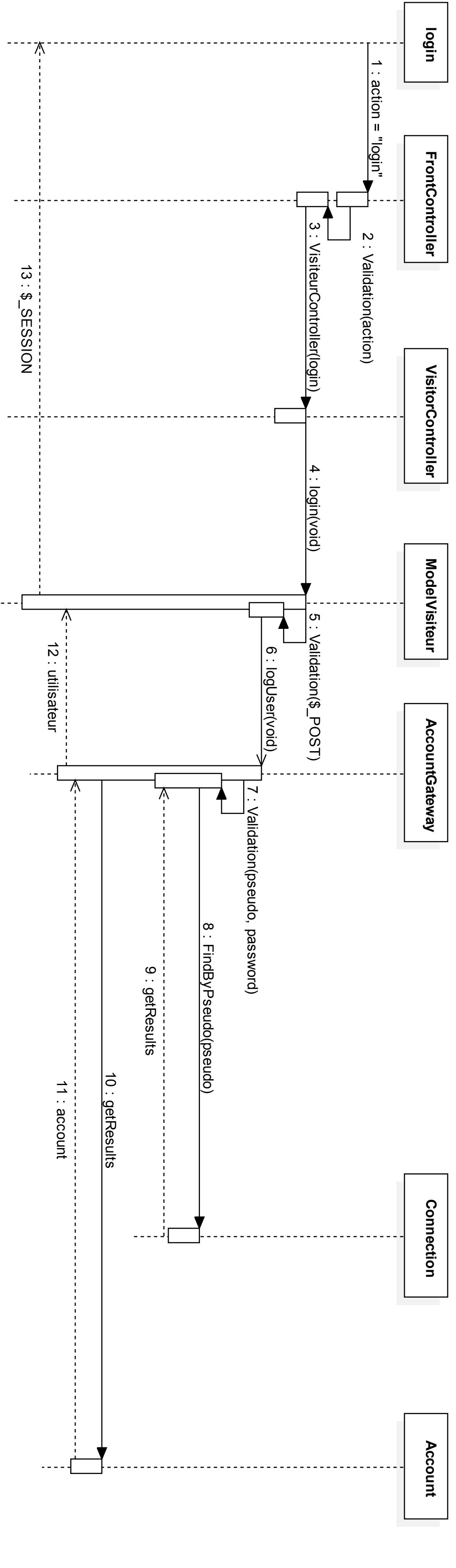
B- Aïe ! Il ne peut pas le vaincre. Ses Nains sont donc tous défaussés et le Dragon est remis face cachée sur sa pile. C'est au tour du joueur suivant.

**C**- Le Piocheur peut encore piocher une dernière carte et dévoile l'Épée. Celle-ci rejoint la main du joueur.

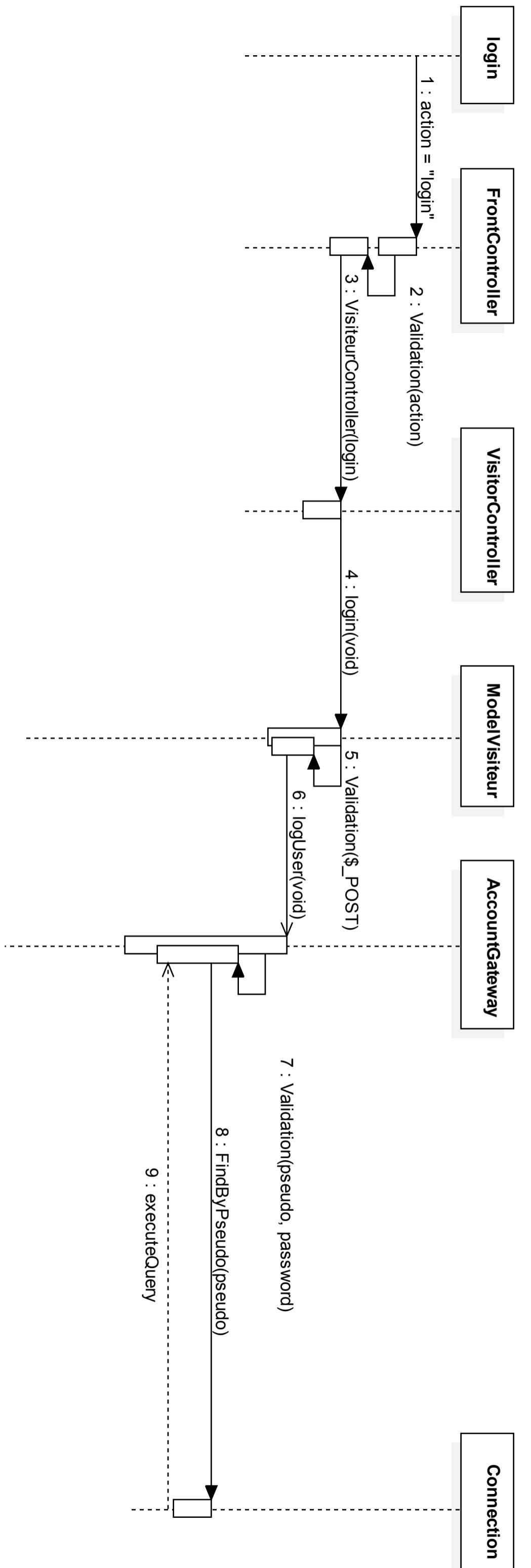
Le Piocheur est maintenant défaussé et c'est au tour du joueur suivant.

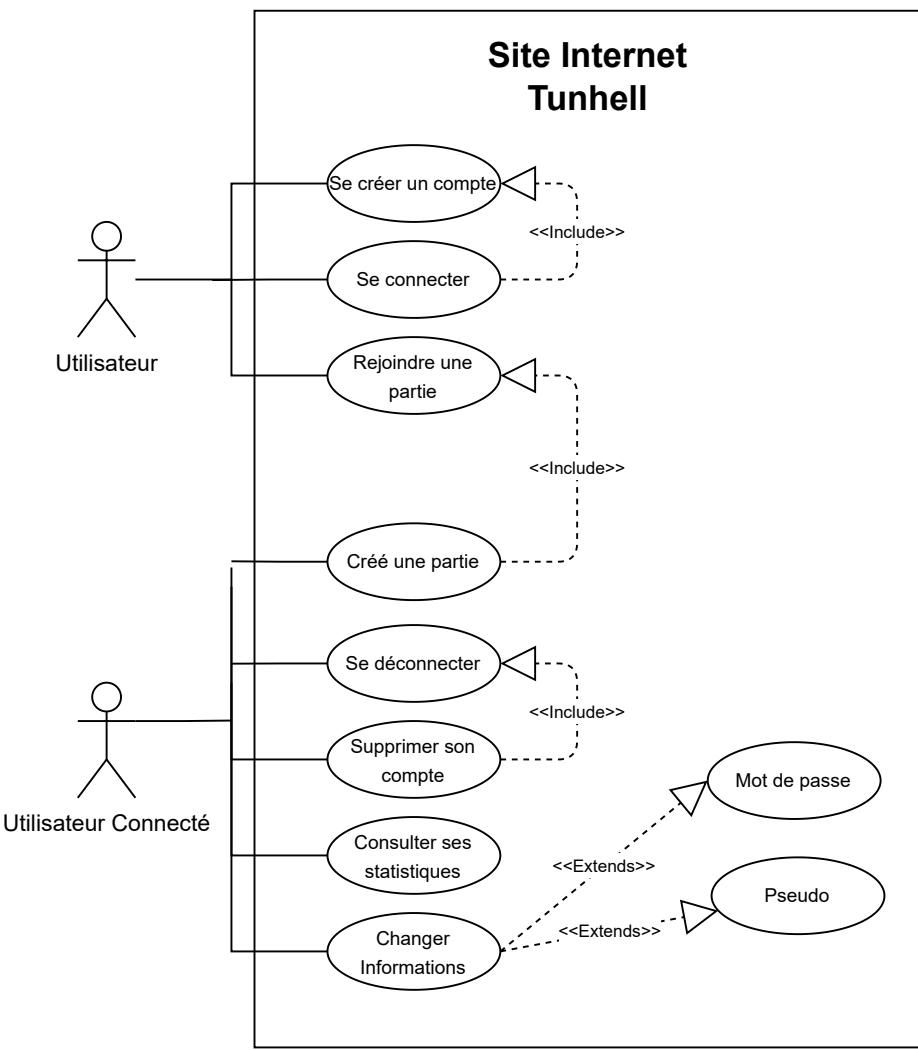


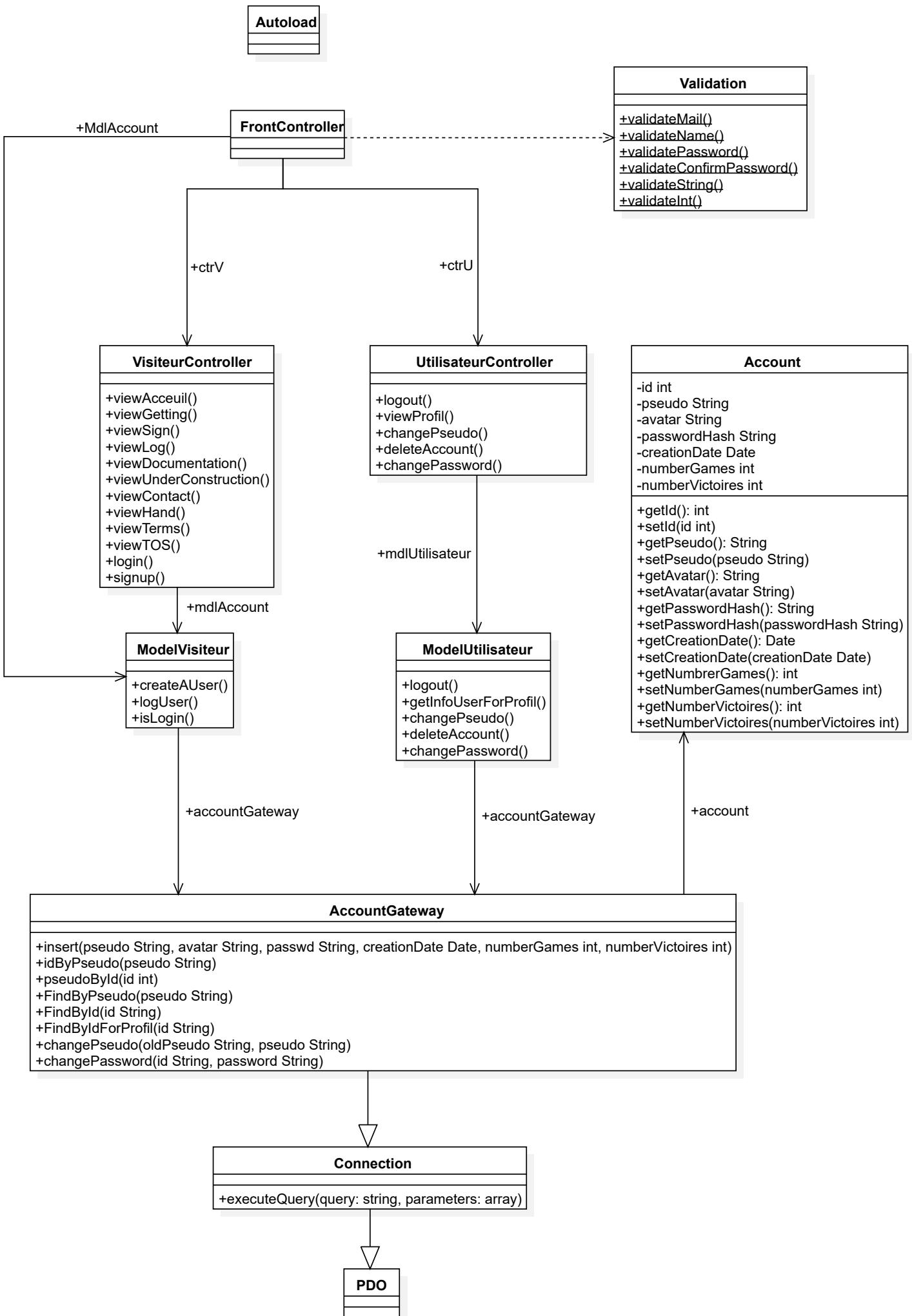
**sd ConnexionAuxComptes**



**sd CreationDesComptes**







Prévisionnel commun P3

Proj n°3 : Tunell

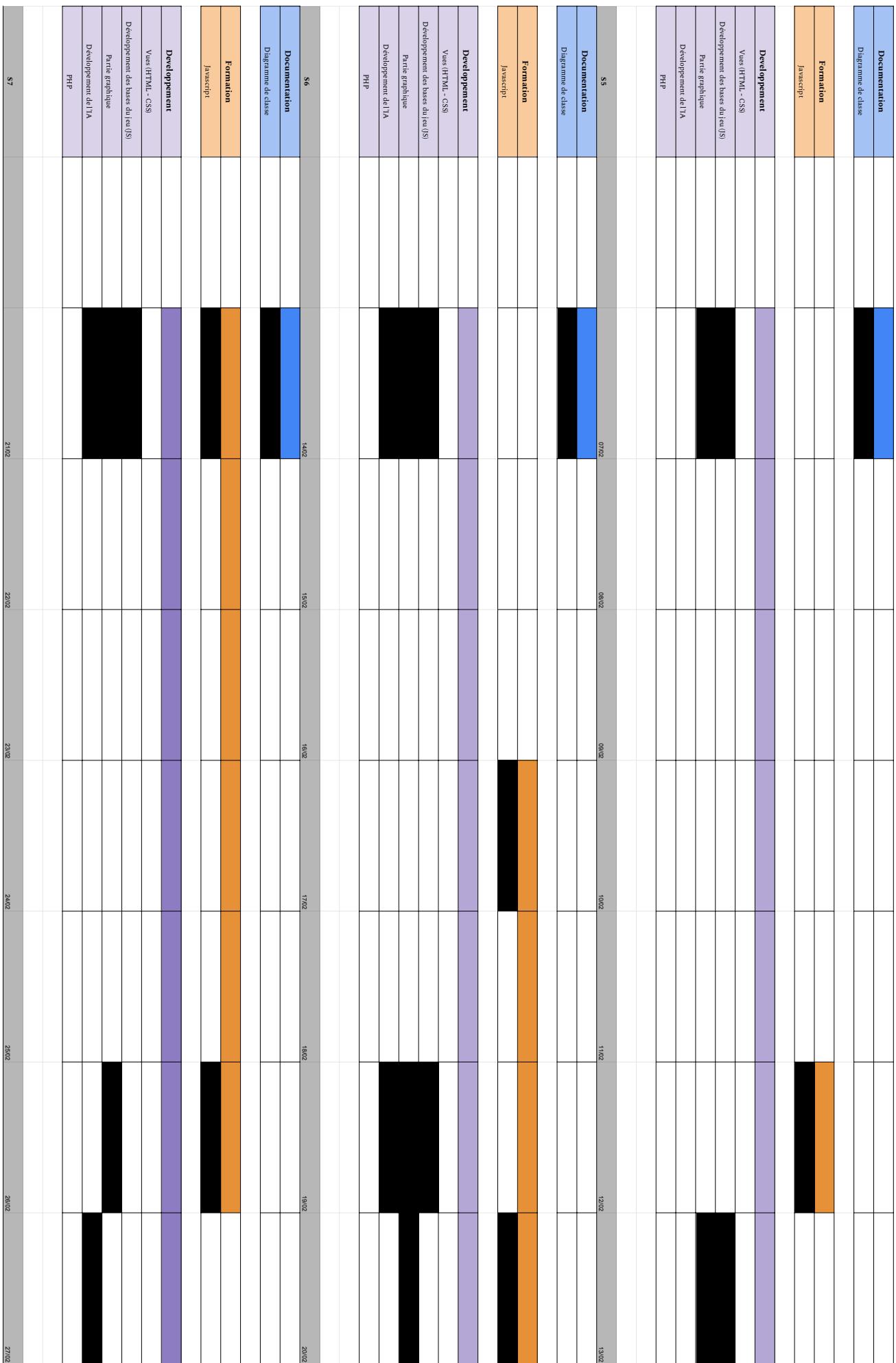
\$1

15/01

16/01





	<b>Documentation</b>			
	Diagramme de classe			
<b>Formation</b>				
Javascript				
<b>Développement</b>				
Landing Page (HTML - CSS)				
Gestion de la partie Utilisateur				
Développement des bases du jeu (JS)				
Développement de l'IA				
PHP				