# JAVA 2 - PROJET

**Réalisé par Omnes Camille, KARTHIGESU Prousoth et Maréchal Théo**

# Introduction

Dans le cadre de notre formation Java, nous devions concevoir et développé une application de jeu vidéo en Java.

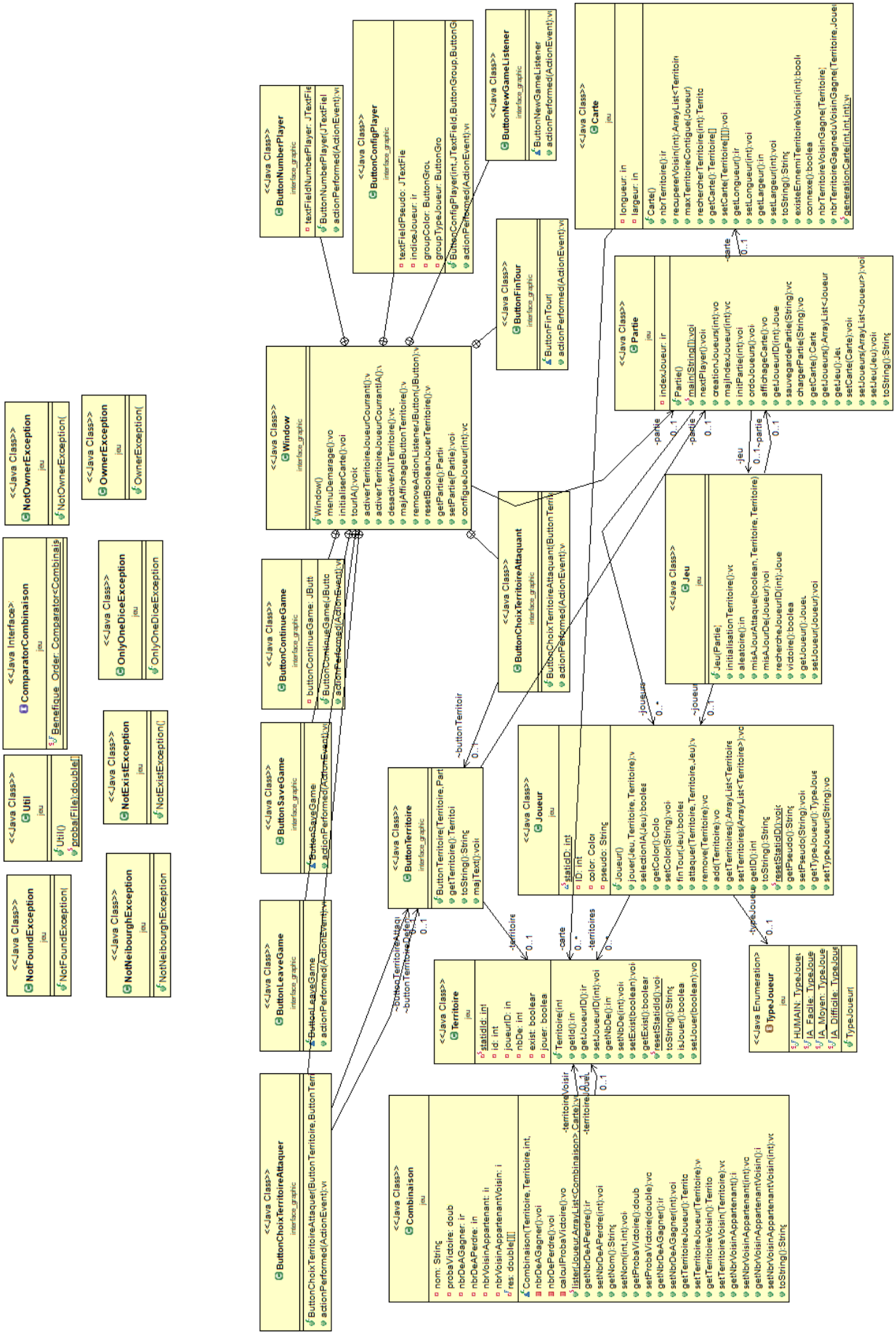
# Objectif et contexte du projet

L'objectif du projet est de développer le jeu « Dice Wars ». L'ensemble des fonctionnalités sont détaillées dans le sujet fourni. Le projet a été globalement réalisé dans le respect des consignes. Néanmoins, nous avons également décidé d'apporter quelque changement que nous trouvions plus intéressent pour l'application.

# Conception du projet

Après une analyse approfondie du sujet, nous avons tout d'abord réfléchi sur la conception de l'application. Pour cela, nous avons utilisé des outils de modélisation que ce soit pour réaliser des diagrammes de classe ou encore une maquette de l'interface de l'application.

Voici le diagramme de classe finale (version graphique) :



# Les fonctionnalités implémentées

- **Version console**

Pour la version console, il y a la possibilité de jouer avec un nombre potentiellement infini de joueur, même si c'est moins recommandé, il est recommandé de jouer avec au maximum 6 joueurs. Pour éviter les problèmes liés aux dimensions du territoire, nous avons choisi de mettre en place la génération de carte aléatoire automatiquement en début de partie. Pour cette version console, seulement des joueurs physiques peuvent s'affronter. Il n'y a pas d'IA d'implémenté dans cette version.

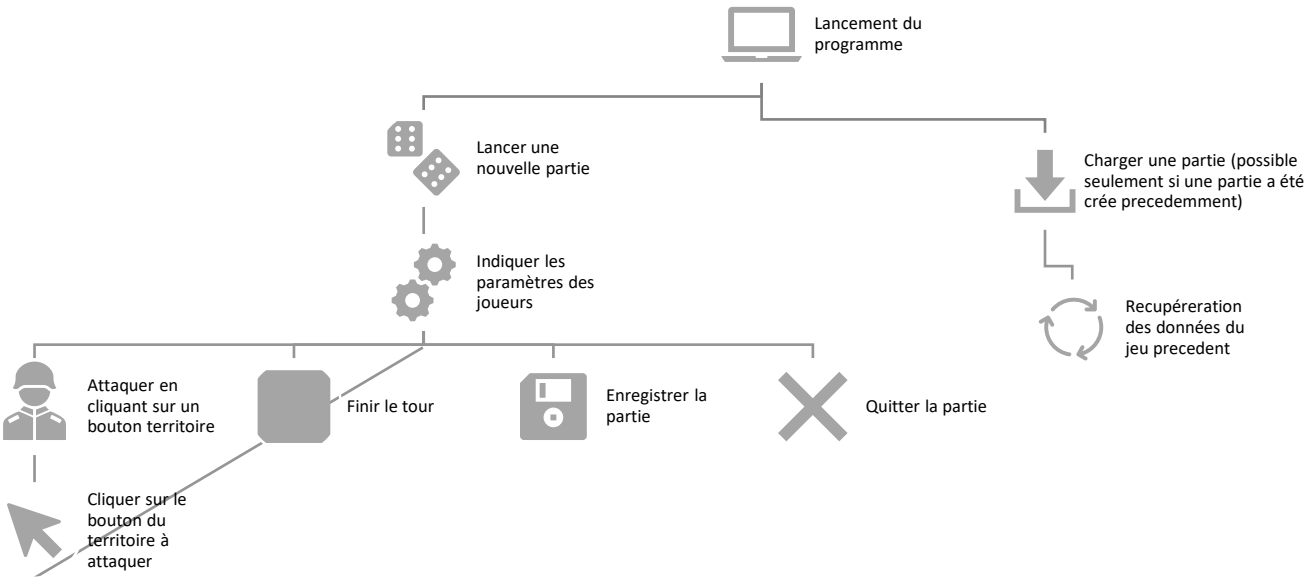
D'abord, on demande à l'utilisateur de rentrer le nombre de joueur. Une fois que le nombre de joueur est rentré, la partie va être créée. Dans la partie, il va y avoir la carte qui va être téléchargé depuis le fichier csv_type, les joueurs vont être créé, chaque territoire va être attribué au joueur, chaque dé va être attribué à chaque territoire de chaque joueur puis l'ordre des joueurs va être mélangé. Ensuite, le jeu va être créé, et le premier joueur va jouer. Pendant son tour, le joueur peut attaquer autant de fois avec tous les territoires qu'il possède ayant plus de 1 dé, et attaquer des territoires qui lui sont voisins et qui ne lui appartiennent pas déjà. Une fois que le joueur est satisfait du nombre d'attaque qu'il a fait, il peut finir son tour.

Lorsque le tour termine, le joueur courant va gagner des dés égaux au nombre maximum de territoire contiguë qu'il possède. Ces dés vont être redistribué sur les territoires du joueur de façon aléatoire. Après cette distribution, le programme va vérifier si le jeu n'est pas dans le cas d'une victoire, c'est-à-dire si tous les territoires n'appartiennent pas à la même personne. Si c'est le cas, un message de victoire va être lancé et le jeu va se terminer. Sinon, on va passer au prochain joueur qui pourra faire les mêmes actions que le joueur précédant, attaquer puis arrêter son tour.

- **Version interface graphique**

Afin de réaliser l'interface graphique, plusieurs éléments développés pour la version console ont dû être modifié. Comme par exemple toutes les fonctions qui attendaient une réponse de l'utilisateur via la console. Pour lancer l'application, il faut uniquement créer la classe Window qui appelle la classe JFrame ainsi que toutes les autres classes nécessaires à l'exécution du jeu.

Voici un schéma montrant les différentes étapes de l'interface graphique :




Pour afficher une carte avec l'ensemble des territoires, nous avons créé une classe ButtonTerritoire qui hérite de la classe JButton. Celle-ci contient un territoire correspondant à un territoire de la carte. Et la carte est affichée grâce à un JPanel disposé via un GridLayout. Ainsi, nous pouvons afficher la carte, et l'utilisateur peut directement interagir avec le territoire sur lequel il clique. La classe Window contient également plusieurs classes internet implémentant l'interface ActionListenner qui sont liées avec les boutons afficher à l'écran. C'est grâce à l'interaction du joueur avec les différents ActionListener qu'il navigue entre les différents menu ou joue la partie. Nous vous invitons à lire attentivement les use case (cf. Scénario d'exécution - Version graphique), afin de mieux comprendre son implémentation.

- **Enregistrement / chargement**

Pour gérer la sauvegarde, nous écrivons l'objet Partie dans le fichier « sauvegarde_partie » via les classes FileOutputStream et ObjectOutputStream. Pour cela nous avons tout d'abord implémenté l'interface Serializable aux classes Jeu, Joueur, Partie, Carte et Territoire. Lorsque le joueur le désire sauvegarder la partie, il suffit d'écrire l'objet partie crée initialement dans le fichier. La classe Partie contenant les autres classes précédemment nommées, c'est l'ensemble du contexte de la partie qui sera sauvegardé.

Pour charger la partie, il suffit de lire le fichier de sauvegarde via FileInputStream et ObjectInputStream afin de charger les données stockées.

❖ **Intelligence artificielle**

Le but du jeu étant de gagner tous les territoires, l'IA va vouloir en un tour gagner le plus de nouveau territoire, et les territoires les plus avantageux possible.

Un territoire avantageux est un territoire qui possède beaucoup de dés (un territoire qui a 6 dés est plus avantageux qu'un territoire avec seulement 4 dés), mais c'est aussi un territoire qui a pour voisin beaucoup de joueur appartenant déjà à l'IA. C'est-à-dire que de gagner ce territoire va faire augmenter le nombre maximal de territoire contiguë, et apporter plus de dé à la fin du tour.

Ainsi pour chaque territoire appartenant à l'IA, que l'on retrouve dans la liste de la classe Joueur territoires, on va lister toutes les combinaisons d'attaques possibles. On va alors regarder pour chaque territoire ses voisins. Si le territoire en question n'a qu'un seul dé, on passe au suivant car le territoire ne sera pas en mesure d'attaquer. Si le territoire voisin nous appartient déjà ou bien si l'un des territoires voisins n'existe pas on ne prend pas en compte ces combinaisons.

Une fois que l'on a récupéré les différentes combinaisons, il va falloir calculer la probabilité de victoire que notre territoire a sur son voisin.

Le maximum de dé étant de 8, les scores des dés seront toujours compris entre 1 et 8*6 = 48, nous avons calculé les probabilités qu'ont n dés (n allant de 1 à 8) de faire une certaine valeur. Ces calculs ont été effectués grâce à Matlab comme il était trop complexe de calculer directement dans java car la formule mathématique utilisait des coefficients binomiaux qui eux même utilise des factorielles. La formule en question est celle calculant la probabilité de réaliser la somme s avec n dés :

$$Q(s;k) = \left(\sum_{k=0}^{\frac{s-n}{6}} (-1)^k \binom{n}{k} \binom{s-6k-1}{n-1}\right)/6^n$$

Les valeurs obtenues ont ensuite été mise dans un csv. Ce csv, va être rempli dans le tableau double res[][] de la classe Combinaison qui est en final.

A chaque fois qu'une combinaison va être créée, l'AI va regarder le nombre de dé de son territoire D et celui du territoire à attaquer B. Elle va parcourir toutes les lignes du tableau res colonne D qu'elle va comparer à celle de la colonne B une à une. Dès que l'indice de la ligne de D est plus grand que celui de la colonne B, cela veut dire que la victoire est pour le territoire de l'AI. On veut donc trouver la probabilité de faire ce résultat en particulier. Soit d'avoir pour B dés le résultat res1 ET pour D dés le résultat res2. Pour déterminer cette probabilité, il suffit juste de multiplier les deux probabilités associées au fait de faire res2 avec D dés, et res1 avec B dés. Or cette possibilité de victoire n'est qu'une seule de toutes celles possible. Pour déterminer la possibilité totale de victoire, il faut faire la somme de tous les cas où res2 est plus grand que res1 pour D dés et B dés, ce qui revient à faire un OU et probabilité.

Exemple :

L'IA possède 2 dés sur son territoire, son adversaire a un dé sur son territoire.

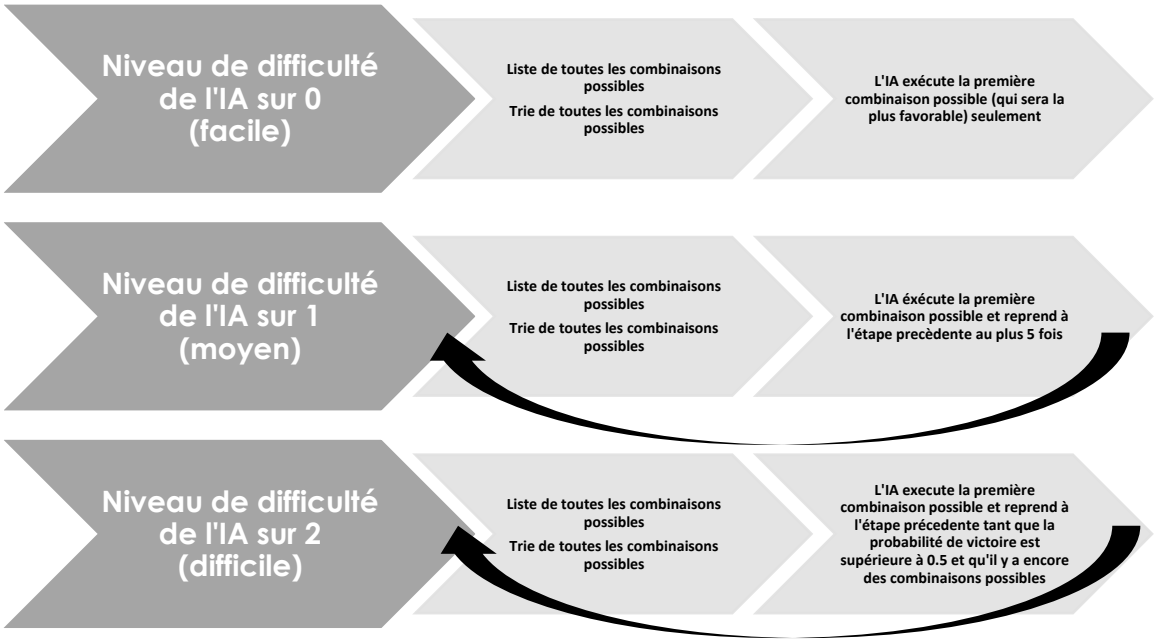
La probabilité de victoire de notre IA avec ces distributions de dé sera donc :

Probabilité pour l'ia de faire 2,3,4,5,6,7,8,9,10,11,12 * probabilité de l'adversaire de faire 1 + probabilité de faire 3,4,5,6,7,8,9,10,11,12 * probabilité de l'adversaire de faire 2 + … + probabilité de faire 7,8,9,10,11,12 * probabilité de l'adversaire de faire 6.

Une fois que la liste de toutes les combinaisons est finie et que le calcul de la probabilité de victoire est effectué, les combinaisons vont être trié. On a implémenté l'interface ComparatorCombinaison qui a un attribut et une méthode anonyme qui permet de trier les différentes combinaisons.

D'abord les combinaisons vont être trié en fonction de leur probabilité de victoire, plus la probabilité de victoire est élevée, plus la combinaison sera au début du classement. Si deux combinaisons ont la même probabilité de victoire, elles vont ensuite être classé par la quantité de dé qu'il rapporte et de dé qu'il perd (plus il en perd plus il est à la fin du classement, plus il en gagne plus il est haut). Si encore une fois, deux combinaisons sont associées aux mêmes gains, on va préférer celle qui a le plus de voisin déjà colonisé, et dont le territoire à colonisé a le plus de voisin colonisé, et trié par rapport à ses informations, qui ont été déterminés lors de la création de la combinaison.

Au moment de jouer de l'IA, dans la méthode jouer(Jeu jeu) de Joueur :




Ainsi, la difficulté d'une IA repose seulement sur sa capacité à jouer plusieurs fois ou non, et le nombre de fois qu'elle peut jouer par tour.

# Les difficultés rencontrées

Nous avons rencontré plusieurs difficultés durant la réalisation du projet. Certaines fonctionnalités étaient compliquées à réaliser algorithmiquement. Comme par exemple la fonction permettant de trouver les voisins d'in territoire ou encore celle qui donne le nombre maximal de territoire contigu d'un joueur. Certaines étaient compliqué à cause de notre manque de connaissance dans un domaine comme pour les Threads. Nous avons également eu des difficultés lorsque nous devions implémenter de nouvelles fonctionnalités à l'application. Dans certains cas, nous devions réécrire une fonction déjà implémentée, car elle entrait en conflit avec une nouvelle fonctionnalité que nous devions ajouter. Mais dans l'ensemble nous avons réussi à surmonter les difficultés et réaliser les fonctionnalités exigées.

# Scenario d'exécution

❖ **Version console**

- Lors du lancement du programme, la console nous demande tout d'abord de renseigner le nombre de joueur

```
Combien de joueur y a-t-il ?
3
```

- Ensuite le programme génère une carte et distribue l'ensemble des territoire (aux 3 joueurs ici) et des dés pour chaque territoire, puis il affiche la carte des territoires

```
Territoire [id=0, joueurID=3, nbDe=8, exist=true] || Territoire [id=1, joueurID=3, nbDe=8, exist=true] || Territoire [id=2, joueurID=3, nbDe=8, exist
Territoire [id=7, joueurID=1, nbDe=8, exist=true] || Territoire [id=8, joueurID=0, nbDe=0, exist=false] || Territoire [id=9, joueurID=3, nbDe=8, exis
Territoire [id=14, joueurID=2, nbDe=8, exist=true] || Territoire [id=15, joueurID=1, nbDe=6, exist=true] || Territoire [id=16, joueurID=3, nbDe=1, ex
Territoire [id=21, joueurID=2, nbDe=8, exist=true] || Territoire [id=22, joueurID=1, nbDe=1, exist=true] || Territoire [id=23, joueurID=3, nbDe=1, ex
Territoire [id=28, joueurID=2, nbDe=1, exist=true] || Territoire [id=29, joueurID=2, nbDe=1, exist=true] || Territoire [id=30, joueurID=1, nbDe=2, ex
```

- Puis l'application informe a un joueur (ici le joueur 2) de jouer en premier. L'ordre des joueurs et générer aléatoirement. Le joueur peut soit passer son tour en tapant « q » soit attaqué un territoire en tapant un autre caractère. Ici, on décide d'attaquer

```
tour du joueur 2
Pour commencer votre tour appuyer sur n'importe quelle touche.
Pour finir votre tour rentrer q
*
```

- Ensuite, le joueur choisi un de ses territoires pour attaquer (14) et un territoire à attaquer (15). Ici, le joueur a perdu son attaque, son territoire n'as plus qu'un seul dé. Puis il décide de mettre fin à son tour

```
Veuillez rentrer l'id du territoire attaquant
14
Veuillez rentrer l'id du territoire à attaquer
15
Score Defense
23
Score Attaque
20
false
Territoire [id=0, joueurID=3, nbDe=8, exist=true] || Territoire [id=1, joueurID=3, nbDe=8, exist=true] || Territoi
Territoire [id=7, joueurID=1, nbDe=8, exist=true] || Territoire [id=8, joueurID=0, nbDe=0, exist=false] || Territo
Territoire [id=14, joueurID=2, nbDe=1, exist=true] || Territoire [id=15, joueurID=1, nbDe=6, exist=true] || Territ
Territoire [id=21, joueurID=2, nbDe=8, exist=true] || Territoire [id=22, joueurID=1, nbDe=1, exist=true] || Territ
Territoire [id=28, joueurID=2, nbDe=1, exist=true] || Territoire [id=29, joueurID=2, nbDe=1, exist=true] || Territ
Pour finir votre tour appuyer sur q
q
```

- À la fin d'un tour, les joueurs regagnent des dés équivalents au nombre maximal de territoire contigu. Puis la carte et mis à jour

```
Fin du tour
Territoire [id=0, joueurID=3, nbDe=8, exist=true] || Territoire [id=1, joueurID=3, nbDe=8, exist=true] || Terr
Territoire [id=7, joueurID=1, nbDe=8, exist=true] || Territoire [id=8, joueurID=0, nbDe=0, exist=false] || Ter
Territoire [id=14, joueurID=2, nbDe=2, exist=true] || Territoire [id=15, joueurID=1, nbDe=6, exist=true] || Te
Territoire [id=21, joueurID=2, nbDe=8, exist=true] || Territoire [id=22, joueurID=1, nbDe=1, exist=true] || Te
Territoire [id=28, joueurID=2, nbDe=1, exist=true] || Territoire [id=29, joueurID=2, nbDe=1, exist=true] || Te
```

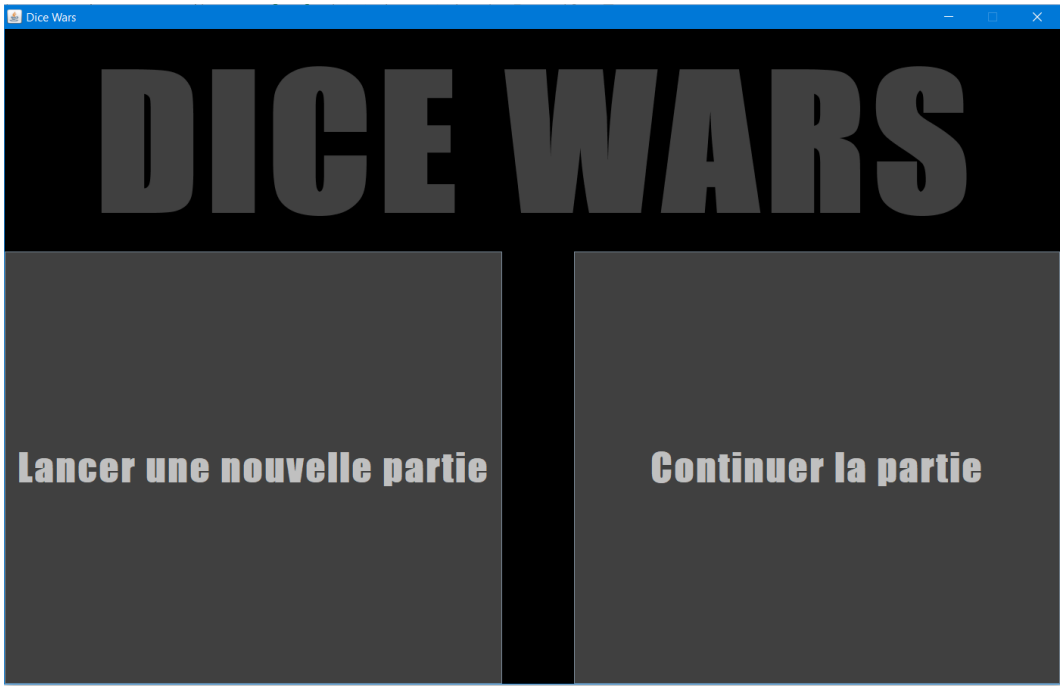
- Ensuite c'est au tour d'un nouveau joueur. Ici le joueur gagne son tour

```
tour du joueur 1
Pour commencer votre tour appuyer sur n'importe quelle touche.
Pour finir votre tour rentrer q
*
Veuillez rentrer l'id du territoire attaquant
15
Veuillez rentrer l'id du territoire à attaquer
14
Score Defense
8
Score Attaque
15
true
affichage jeu maj attaque
[Territoire [id=18, joueurID=2, nbDe=4, exist=true], Territoire [id=27, joueurID=2, nbDe=1, exist=true], Terr
Territoire [id=0, joueurID=3, nbDe=8, exist=true] || Territoire [id=1, joueurID=3, nbDe=8, exist=true] || Ter
Territoire [id=7, joueurID=1, nbDe=8, exist=true] || Territoire [id=8, joueurID=0, nbDe=0, exist=false] || Te
Territoire [id=14, joueurID=1, nbDe=7, exist=true] || Territoire [id=15, joueurID=1, nbDe=1, exist=true] || T
Territoire [id=21, joueurID=2, nbDe=8, exist=true] || Territoire [id=22, joueurID=1, nbDe=1, exist=true] || T
Territoire [id=28, joueurID=2, nbDe=1, exist=true] || Territoire [id=29, joueurID=2, nbDe=1, exist=true] || T
```

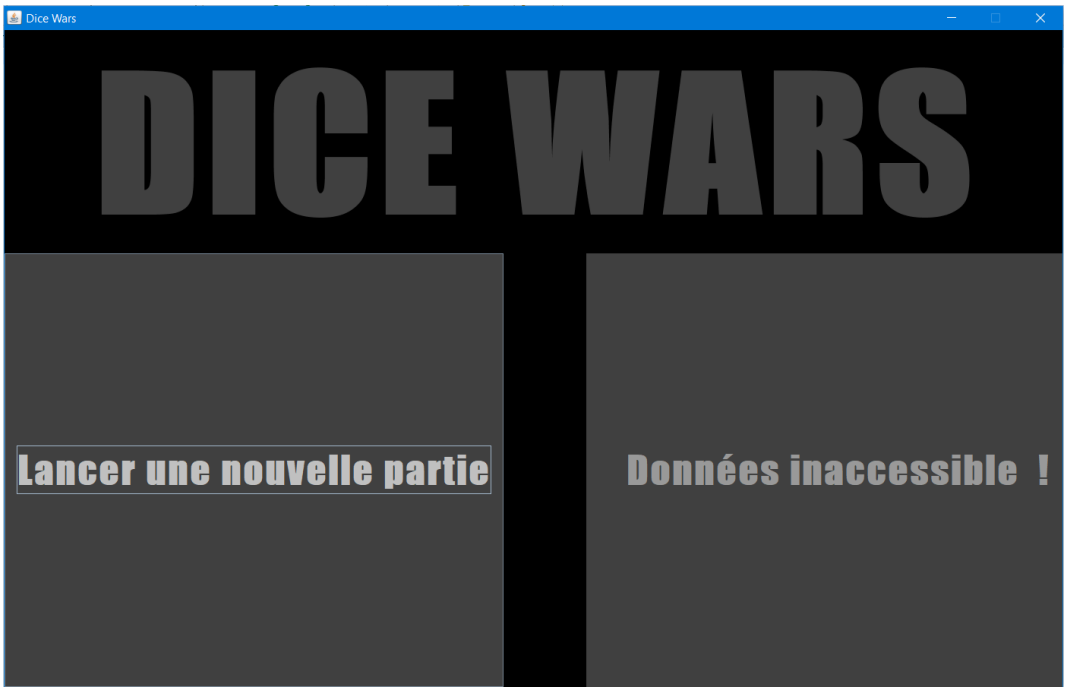
- Ainsi, chaque joueur joue les uns après les autres jusqu'à qu'il y a un vainqueur

❖ Version graphique

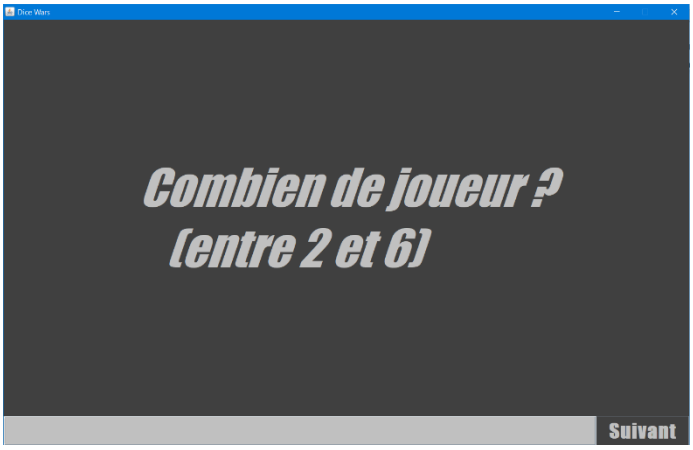
- Voici le menu de démarrage de l'application. On peut soit créé une partie soit chargé une partie sauvegarder

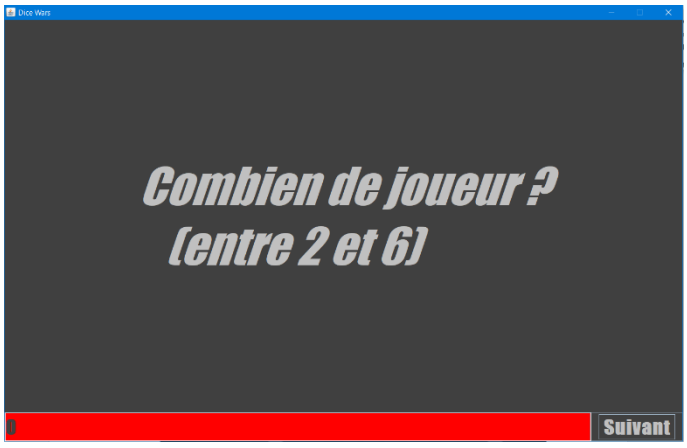



- Lorsque le joueur clique sur le bouton « continuer la partie » alors qu'il y a pas de sauvegarde, le message suivant apparait

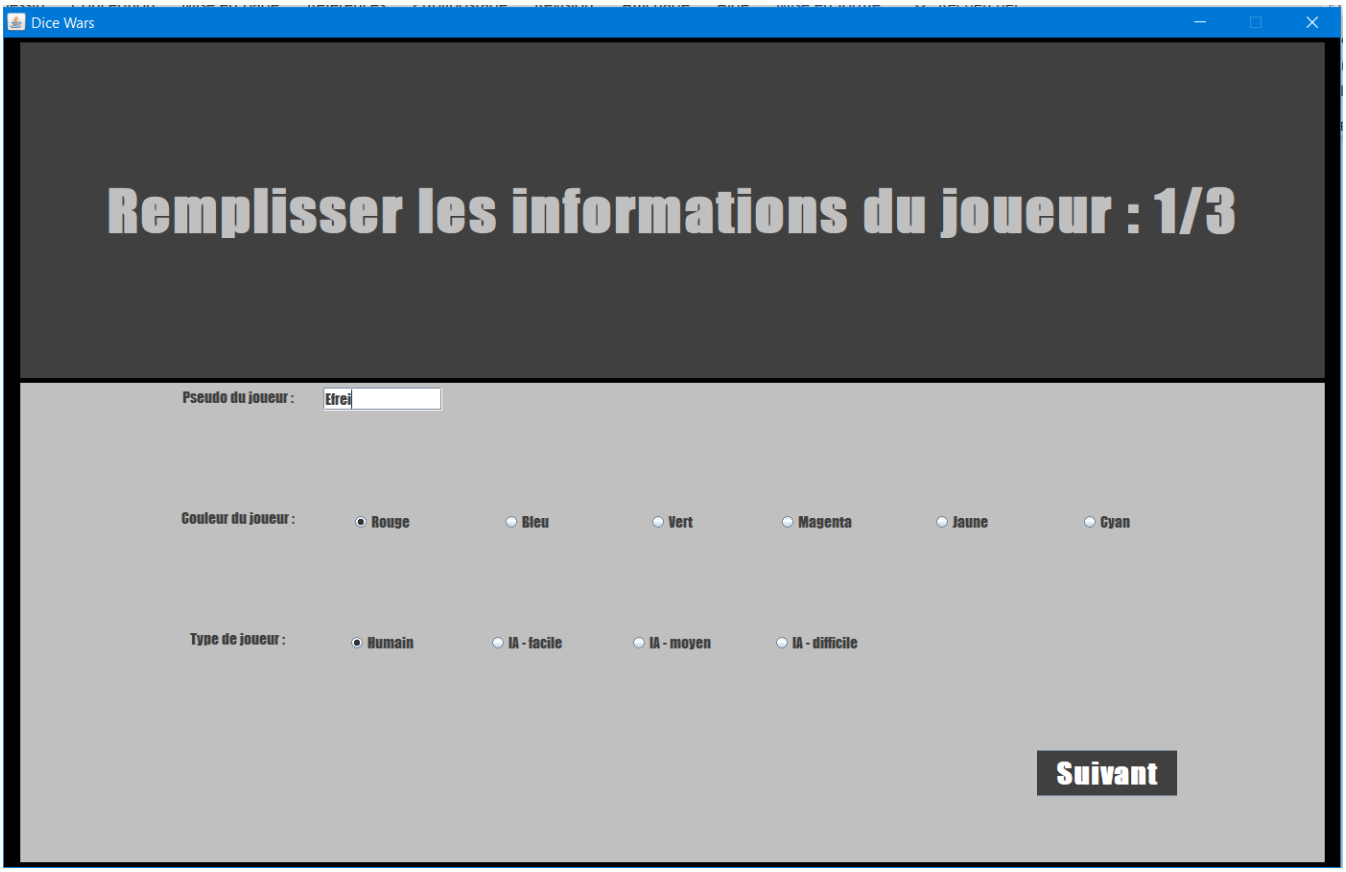


- S'il crée une nouvelle partie il doit d'abord renseigner un nombre de joueurs entre 2 et 6. Sinon la zone de texte devient rouge

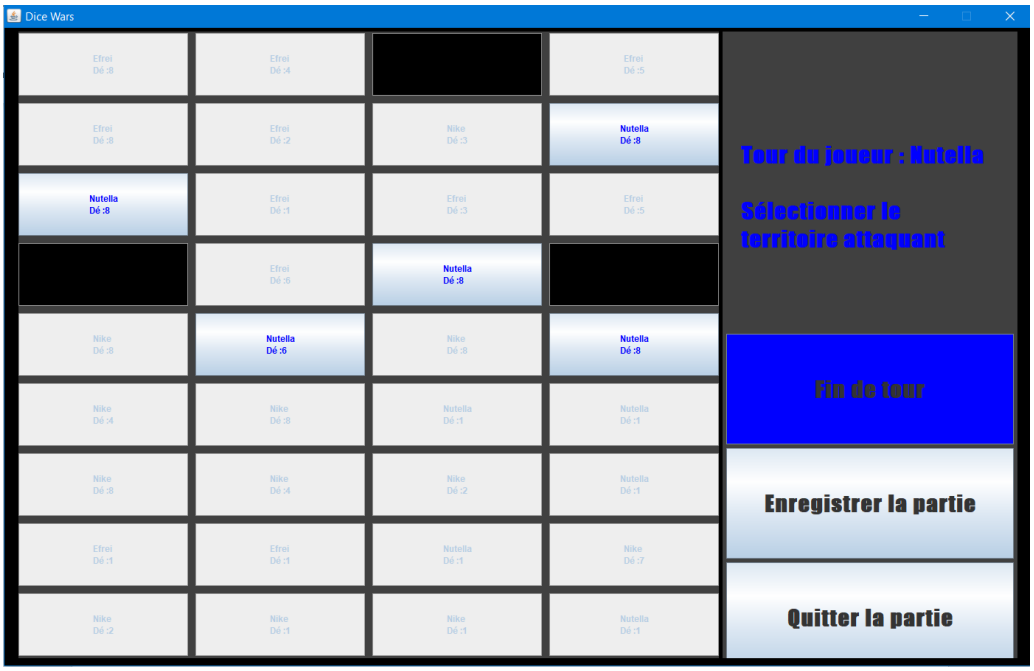


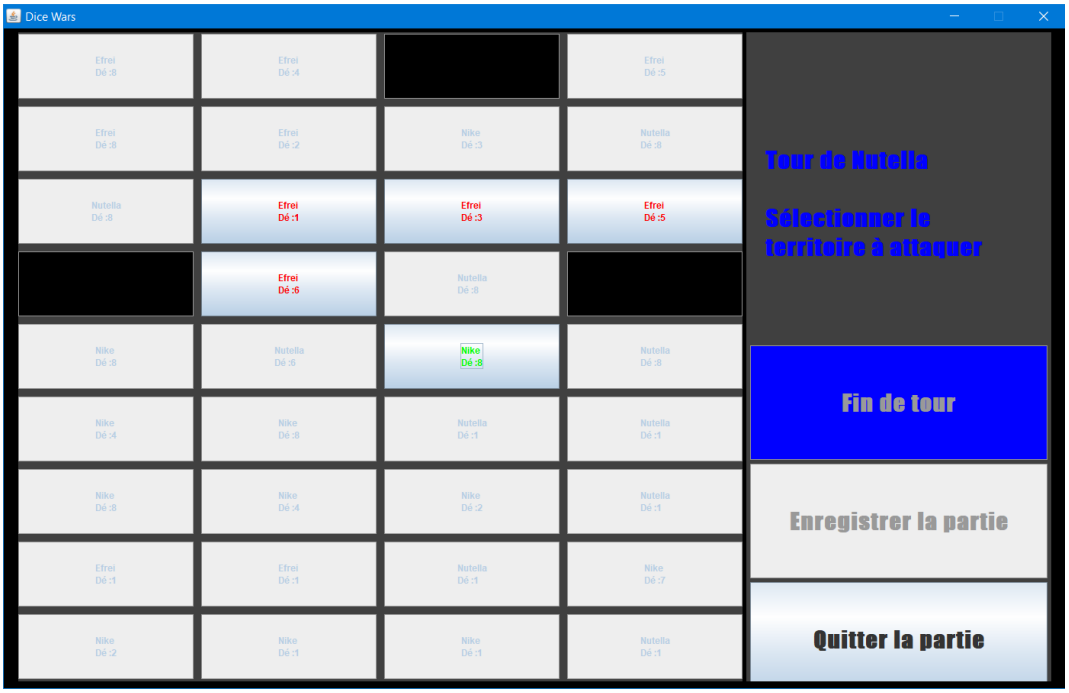

- Puis pour chaque joueur il demande un pseudo, une couleur et le type de joueur (humain ou IA)



- Ensuite, une carte connexe aléatoire est générées. Ici, c'est au tour du joueur « Nutella », il doit choisir un de ses territoires pour attaquer. Il ne peut cliquer que sur ses territoires qui ont plus d'un dé et qui ont au moins un voisin ennemi




- Une fois le territoire attaquant sélectionné, les territoires voisins, ennemis du territoire attaquant sont activé. Le joueur n'a plus qu'a sélectionner le territoire a attaquer



- Le joueur peut mettre fin à son tour, sauvegarder la partie ou la quitter via les boutons situés sur le menu à droite. Au-dessus des boutons, un message informe aux joueurs ce qu'ils doivent faire. Il informe également la victoire d'un joueur.