

AKINATOR

le mathémagicien



Projet tutoré sous la direction de
Maeva Paradis
et Clément Jacq

26/03/2023

Conçu par
Florent Fontaine
Killian Mourgand
Maxime Fournier
& Laurian Jamin



Nous tenons à remercier nos tuteurs Docteure Maeva Paradis et Docteur Clément Jacq pour leur soutien et leurs conseils,

Docteur Paul-Marie Grollemund pour avoir assisté à notre soutenance et ses questions,

Les auteurs de pokéAPI pour nous avoir fait gagner un temps précieux,

Satoshi Tajori et Nintendo pour avoir conçu l'un des plus fascinants univers de la culture vidéoludique et populaire qui a façonné notre enfance comme notre travail,

Tous ceux que nous oublions ici et qui nous auront aidés et soutenus au cours de ce projet.

Il s'agira dans ce projet d'étudier, comprendre et concevoir un jeu de devinettes automatisé inspiré du fameux jeu "Akinator" conçu en 2007 par Arnaud Megret.

Ce rapport trace les grandes étapes de la création de ce jeu et les problématiques théoriques et pratiques qui en découlent.

REMERCIEMENTS	1.
RÉSUMÉ	2.
INTRODUCTION	3.
I - LA BASE DE DONNÉES	4.
II - LES MODÈLES MATHÉMATIQUES	8.
III - LA PROGRAMMATION	11.
CONCLUSION	14.
BIBLIOGRAPHIE	15.
ICONOGRAPHIE	16.

INTRODUCTION

Nous connaissons tous le jeu Akinator qui a fait le plaisir de nos jeunes âgées et continue encore à nous surprendre par sa capacité à deviner les personnages les plus obscurs. Plus qu'un simple jeu, ce "magicien" opère sur nous une fascination justement liée à son caractère magique. Dans le cadre de ce projet nous souhaitions percer le mystère de cette magie. En bons étudiants de l'Institut Universitaire Technique d'Aurillac, la voie de la pratique concrète était donc la meilleure manière de comprendre et de théoriser ce mystère. C'est pourquoi nous avons décidé de nous atteler à produire notre propre réplique du jeu.

C'était en effet le meilleur moyen de mettre en lumière les mystères d'Akinator et ainsi d'en observer les rouages puis, ensuite, de les appliquer à notre propre jeu-projet.

Pour ce projet, nous avons donc dû mener un travail en groupe qui a mené à de nombreuses problématiques et défis pratiques.

Dans un premier temps, nous avons ainsi dû mettre en place une base de données qui soit assez cohérente, complète et connue pour permettre au jeu de fonctionner dessus et surtout de produire l'amusement attendu.

En parallèle, nous avons également dû réfléchir à la manière dont les mathématiques rentraient en compte dans la magie du jeu. Quels étaient les concepts appliqués au jeu ? Et surtout comment les comprendre et les exploiter dans notre version ?

Enfin, nous avons dû nous intéresser à la manière dont nous devons coder le jeu. Avec quel(s) langage(s) ? Sous quelle forme ? Ou encore comment y injecter la base de données retenue et les théories mathématiques apprises.

Tout cela n'a donc pas été de tout repos et, malgré le peu de temps imparti, nous sommes aujourd'hui fiers de vous présenter notre propre projet d'Akinator et le processus détaillé de sa conception.

Nous vous souhaitons une bonne lecture de ce rapport et une bonne partie !

I - LA BASE DE DONNÉES

A - Le choix du corpus

Les prémisses de ce projet ont été de définir quel serait le corpus le plus adapté pour mettre en place un jeu Akinator. Tout comme le jeu d'origine, nous avons rapidement décidé qu'il s'agirait de personnages. Ils étaient en effet plus amusants à identifier que des concepts ou des objets et, a priori, plus faciles à distinguer que des animaux ou des végétaux.

A partir de là, nous devons restreindre le choix de ces personnages à un corpus à la fois assez riche pour rendre le jeu intéressant, assez connu pour que les joueurs puissent penser à des personnages (les personnages de *League of Legend* n'auraient pas été assez connus pour certains) et assez varié dans ses personnages et leurs caractéristiques pour que le jeu distingue le bon personnage (des joueurs de football n'auraient pas donné assez de caractéristiques différentes et originales pour qu'on puisse les deviner aisément).

Après des hésitations entre les dieux des différentes mythologies, les personnages d'Harry Potter ou ceux de l'univers Marvel, nous avons décidé de retenir les personnages d'anime japonais qui étaient assez nombreux (mais moins indénombrables que ceux des mangas), assez variés dans leur apparence et largement connu du plus grand nombre.

Nous sommes donc arrivés à la conclusion que "personnages d'animés" pourrait être un thème intéressant et nous avons décidé d'en faire notre base de données.

B - Les étapes du recueil de la base de données

Afin de mettre en place cette base de données, nous avons tout d'abord webscrappé à l'aide de python et des bibliothèques pandas et beautiful soup des données depuis le site MyAnimeList pour récupérer des informations sur les personnages d'animés.

Nous avons cependant rencontré des problèmes avec de nombreuses valeurs manquantes et un manque de précision dans les données qui rendaient l'utilisation de la base données impossible pour devenir efficacement les personnages.

Suite à cela nous avons décidé d'opter pour une base de données préexistante spécialement conçue pour les personnages d'animés. Nous avons trouvé cette base de données sur [kaggle.com](https://www.kaggle.com/datasets/robertodaniel/anime-character-traits-dataset) au titre révélateur de "Anime Character Traits Dataset".

Cette base de données nous paraissaient intéressante car elle contenait non seulement un grand nombre de personnages mais également des caractéristiques physiques et morales particulières qui revenaient régulièrement d'un personnage à l'autre (par exemple, les cheveux anti-gravités ou la voracité qui sont très marquants mais renvoient aussi bien à San Goku de *Dragon Ball*, à Naruto ou à Luffy de *One Piece*). Pour prendre une terminologie de cybersécurité, les personnages étaient donc riches de caractéristiques quasi-identifiantes mais n'étaient pas immédiatement identifiables.

L'autre intérêt de la base de données était que le niveau de détail des caractéristiques des personnages allaient de la généralité (homme ou femme), à la semi-généralité (couleur de cheveux) pour finir sur du détail très précis avec ses tags sur les traits moraux (analytique, immortel), physiques (bandeaux dans les cheveux, yeux rares) et sociaux (militaire, ninja...).

En traitant ces données, nous avons d'abord essayé de mettre tout les tags en binaire, mais nous avons rapidement réalisé que cette approche n'était pas optimale pour notre projet. En effet, la base de données devenais bien trop lourde à gérer.

Une solution à ce problème aurait été de pouvoir réunir ces tags. Nous avons alors décidé d'essayer une approche de classification en testant les algorithmes de K-means et ceux de CAH. Cependant, les résultats n'étaient que partiellement convaincants et nous avons donc décidé d'y renoncer.

En testant des premières versions de notre code, nous avons par ailleurs compris que les tags n'étaient pas assez forts pour déterminer des personnages. En effet, l'inégalité de traitement entre les différents personnages (les plus connus avaient plus de tags que les secondaires) rendait le travail plus difficile pour notre Akinator qui finissait par trouver des personnages qui avaient certes les caractéristiques demandés mais n'étaient pas ceux à qui nous pensions.

Face à ce second constat d'échec, et alors qu'il ne nous restait plus assez de temps, nous avons décidé de nous replier vers le choix des Pokémon. Ce choix correspondait au corpus retenu par le précédent groupe de PTUT ayant planché sur la conception d'un Akinator.

Afin d'obtenir ce nouveau corpus de personnages, nous avons commencé par chercher des sites référençant les Pokemon, nous avons ainsi commencé à faire du webscrapping sur plusieurs sites de catalogage de Pokémons. L'entreprise s'est également avérée hasardeuse avec des résultats en anglais que nous avons du mal à traduire en français.

Finalement, le salut est venu de PokeApi, une API recensant tous les Pokémons de toutes les générations.

C - Traitement des données de la base

Cette base de données était la plus simple à manipuler. Pour autant, elle nécessitait encore d'être travaillée pour être appliquée à notre code.

Dans un premier temps, nous avons traduit les noms des Pokémons à l'aide d'un dictionnaire capable de récupérer les noms français à partir des noms anglais.

Ensuite, nous avons écarté les versions alternatives des Pokémons (notamment ceux de la génération Lune et Soleil qui sont les mêmes Pokémons mais avec des mutations morphologiques). Pour cela, nous n'avons retenu que le premier Pokémon de chaque ID.

Nous avons également créé la variable Evolutions qui permettait de demander si le Pokémon était lié à telle ou telle évolution.

Enfin, nous avons regroupé les tailles et poids numériques de Pokémons par catégories en regroupant ces quantités en terciles (petit, moyen et grand pour la taille, léger, moyen et lourd pour le poids).

A la fin de cette première étape, nous avons donc une base de données de 840 Pokémons (soit les premières générations) décrits par les variables "Nom", "Types", "Taille", "Poids", "Evolutions", "Génération" "Couleur" et "Description".

II - LES MODÈLES MATHÉMATIQUES

A - Comment rationaliser le choix des questions

Le deuxième aspect central du jeu Akinator est bien entendu de savoir comment il choisit ses questions et comment il reporte les réponses du joueur. En dialoguant avec nos tuteurs et en étudiant quelques articles de vulgarisation en ligne, la notion d'entropie est rapidement apparue comme la réponse à cette problématique. En effet, le principe de l'entropie est le plus souvent résumé à la notion de calcul de l'incertitude ou de chaos.

Or, nous étions face à un jeu dont tout le principe est de quantifier à quel point il ne sait pas ce que recherche le joueur. Autrement dit, en cherchant à deviner ce que pense le joueur, le jeu va tenter d'identifier les meilleures probabilités de poser les questions qui lui permettront de se rapprocher des attentes du joueur et enfin de ce qu'il pense.

Pour reprendre le concept d'arbre de décision, il s'agit pour le jeu de déterminer dans un arbre de décision comment trouver le plus facilement possible le Pokémon qu'imagine le joueur. Ainsi, les étages de l'arbre représentent chacun une nouvelle session de question et les différentes branches de chacun de ces étages une question particulière (question sur le type, le poids...).

Etant donné que le jeu connaît toutes les réponses possibles et toutes les probabilités associées, il peut alors calculer quel enchaînement de questions va le rapprocher le plus facilement du personnage mystère.

Pour cela, le jeu détermine la probabilité d'élimination de réponses que la question qu'il pose va produire. Chaque nouvelle réponse du joueur invalide alors un ensemble de Pokémon qui ne collent pas à sa réponse. A la nouvelle question, le jeu peut alors rechercher la meilleure probabilité d'éliminer une partie du corpus et ainsi de suite jusqu'à ce que le jeu n'ait plus qu'un individu.

D'un point de vue de mathématique, ce calcul est donc celui de l'entropie et se formule ainsi :

$$p = (p_1, \dots, p_n) \mapsto - \sum_{i=1}^n p_i \log p_i$$

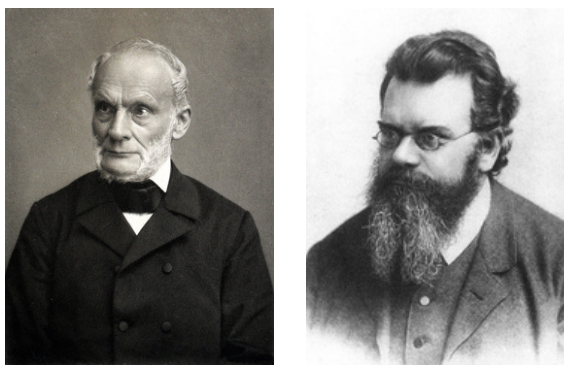
B - L'Histoire de l'entropie

Avant d'aller plus loin dans l'explication de l'entropie, il faut déjà en poser les bases historiques et conceptuelles.

Comme on l'a déjà énoncé, l'entropie est le concept scientifique derrière l'idée d'incertitude, de désordre ou encore de chaos.

Il a, au départ, été énoncé au 19^{ème} siècle pour étudier les modèles thermodynamiques. En particulier, il a été forgé par Rudolf Clausius en 1865 pour étudier le rapport entre la température reçue par un modèle et sa température finale.

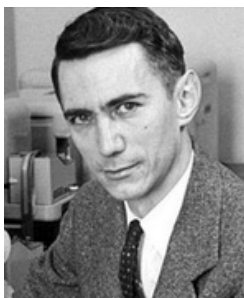
L'entropie a ensuite été développée par le physicien Ludwig Boltzmann en 1872 en une idée plus proche de ce qui nous intéresse : le rapport entre le désordre microscopique d'un modèle et sa forme macroscopique.



Rudolf Clausius et Ludwig Boltzmann

Néanmoins, l'interprétation de l'entropie telle que nous l'utilisons ici est dûe à Claude Shannon (on parle d'ailleurs d'"Entropie de Shannon"). Cet ingénieur de Bell avait pour mission d'étudier la perte d'information dans un système d'information, notamment au moment de convertir de l'information en bites puis de retrouver le sens de ces longues lignes de 0 et de 1.

En 1948, il a donc formulé l'expression que nous utilisons de l'entropie comme le moyen de mesurer le niveau d'incertitude d'un système et donc la difficulté (ou non) à avoir la bonne réponse dans notre cas et à retrouver le sens d'un code binaire.



Claude Shannon

C - Explication de l'entropie

C'est ainsi que Claude Shannon en est arrivé à la formule qui sera plus tard appelée "entropie" en raison de son usage des notions de désordre et d'incertitude communs au concept originel de thermodynamique :

$$p = (p_1, \dots, p_n) \mapsto - \sum_{i=1}^n p_i \log p_i$$

Cette formule est la somme de chaque probabilité multipliée par son logarithme de base 2, le tout en négatif.

Grâce à la propriété $\log(1/a) = -\log(a)$, on peut également exprimer cette formule comme la somme de chaque probabilité multipliée par son inverse passé au logarithme de base 2 soit :

$$H(X) = \sum (p_i \cdot \log(1/p_i))$$

Quelque soit la forme retenue, il s'agit à chaque fois d'étudier à quel moment l'entropie est la plus faible. En effet, plus l'entropie est faible et plus cela signifie que l'on va facilement trouver la bonne réponse (qu'il y a donc peu d'incertitude à répondre). Par exemple, si Akinator n'a plus qu'un personnage dans ses possibilités, alors son entropie est de 0 puisque la probabilité de trouver le personnage est de 1 (nombre de fois que le personnage revient) sur 1 (nombre de personnages possibles) et donc que l'entropie se calcule $-1 \cdot \log(1)$, c'est-à-dire 0 ($\log(1)$ étant égal à 0).

A l'inverse, plus, l'incertitude est grande, plus cela signifie que cela sera difficile de déterminer le personnage. Par exemple, s'il n'y a plus un personnage à deviner mais 3 qui ont la même probabilité d'apparaître, on a :

$$\begin{aligned} & -1/3 \cdot \log(1/3) - 1/3 \cdot \log(1/3) - 1/3 \cdot \log(1/3) \\ &= -3 \cdot (1/3 \cdot \log(1/3)) \\ &= 3 \cdot (1/3 \cdot \log(3)) \\ &= \log(3) \end{aligned}$$

De façon plus générale, on observe que cette formule donne toujours un résultat supérieur ou égal à 0 étant donné qu'une probabilité est forcément positive et comprise en 0 et 1. Le $\log(p)$ sera donc soit nul, soit négatif rendant $-p \cdot \log(p)$ nul ou positif.

De la même manière, le nombre de probabilités réduisant l'importance de chacune des probabilités et multipliant le nombre d'éléments de la somme de la formule d'entropie, plus il y a de probabilités et plus chacune est infime, plus l'entropie augmente.

III - LA PROGRAMMATION

A - Un code de test

Pour commencer la partie code de notre projet, nous avons tout d'abord cherché à obtenir quelque chose de fonctionnel et de visuellement parlant.

nous avons donc essayé de programmer un jeu avec une interface graphique. Ce code devait pouvoir fonctionner avec une base de données de test comprenant seulement quelques personnages et quelques questions.

Pour ce faire, nous avons utilisé deux langages de programmation : Python pour la gestion des questions et la manipulation des bases de données avec le module Pandas, et Java pour la création de l'interface graphique.

Ce choix était celui de l'évidence et de l'expérience puisque nous avons déjà beaucoup utilisé Pandas avec Python. De même, nous avons également étudié l'interface graphique en Java.

En cela, nous avons développé trois scripts Python pour réaliser différentes tâches :

- Un premier script était dédié à la création de fichiers temporaires pour stocker les données de jeu en cours ;
- Un deuxième script était responsable de la gestion des questions à poser au joueur. Pour cette première version, nous n'avions pas intégré l'entropie comme facteur déterminant des questions à poser car nous n'avions simplement pas pris connaissance de ce concept ;
- Enfin, un troisième script avait pour objectif de récupérer les réponses de l'utilisateur et de modifier les bases de données en conséquence.

Parallèlement à cela, nous avons créé l'interface graphique en Java, qui communiquait avec les scripts Python à l'aide d'un ProcessBuilder.

Cela nous a permis d'exécuter les scripts Python directement depuis l'interface Java et de récupérer les éléments imprimés par ces scripts pour les afficher à l'écran.

Pour rendre l'interface plus attrayante et engageante, nous avons ajouté des images et des éléments de pixel art qui s'harmonisent avec le thème du jeu.

Nous avons pris le soin de trouver des images libres de droit (voir l'exemple en fin de rapport).

III - LA PROGRAMMATION

B - Le code final

Nous n'avons finalement pas retenu ce premier code qui ne prenait pas en compte la notion d'entropie. Si nous avons gardé l'idée de séparer le code entre sa partie fonctionnelle en Python et son interface graphique en Java, nous avons dû recommencer le code fonctionnel afin de le rendre opérationnel sur une véritable base de données et en appliquant la notion d'entropie. Pour cela, nous avons utilisé les bibliothèques suivantes :

- **csv** : pour lire et écrire des fichiers CSV;
- **math** : pour effectuer des calculs mathématiques, en particulier le calcul de l'entropie ;
- **collections** : pour compter les occurrences d'éléments dans une liste;
- **os** : pour accéder aux fichiers et répertoires du système d'exploitation.

Les fonctions utilisées étaient (et sont toujours) :

load_data_from_csv pour charger les données Pokémon à partir d'un fichier CSV;

calculate_entropy pour calculer l'entropie d'un attribut donné dans la base de données dans la fonction à l'aide de la méthode `Counter()` de la bibliothèque `collections`. La fonction crée une liste de probabilités en divisant le nombre d'occurrences de chaque valeur d'attribut par la taille de la base de données. Enfin, la fonction calcule l'entropie en utilisant la formule de Shannon pour l'entropie : $H = -\sum(p * \log_2(p) \text{ for } p \text{ in probabilities})$;

get_answer pour demander à l'utilisateur une réponse valide (oui/non/je ne suis pas sûr) à une question donnée. Elle permet donc l'implémentation des questions. La fonction utilise la fonction `input()` pour demander à l'utilisateur une réponse. La méthode `strip()` est utilisée pour supprimer les espaces avant et après la réponse de l'utilisateur, et la méthode `lower()` est utilisée pour convertir la réponse en minuscules, afin de faciliter la comparaison.

La fonction utilise une boucle `while True` pour continuer à demander une réponse jusqu'à ce qu'une réponse valide soit donnée. Le programme continue de poser des questions jusqu'à ce qu'il devine le Pokémon ou qu'il n'y ait plus de questions à poser. Si le programme devine correctement le Pokémon, il propose de jouer à nouveau. Si le programme ne peut pas deviner le Pokémon, il signale une erreur et se termine ;

III - LA PROGRAMMATION

`ask_question` pour poser une question à l'utilisateur sur un attribut spécifique et retourne une nouvelle sous-base de données basée sur la réponse de l'utilisateur.

Le premier paramètre `attribute` est l'attribut pour lequel la question doit être posée.

Le deuxième paramètre `data` est la base de données (liste de dictionnaires) qui contient l'attribut.

Le troisième paramètre `uncertain_attributes` est un ensemble qui contient des attributs incertains, c'est-à-dire ceux pour lesquels l'utilisateur a répondu "je ne suis pas sûr" dans une question précédente. La fonction commence par créer un ensemble de toutes les valeurs possibles pour l'attribut donné dans la base de données.

Ensuite, la fonction itère sur chaque valeur possible et pose une question à l'utilisateur en utilisant un dictionnaire de questions prédéfinies (`question_map`) pour différents attributs. La réponse de l'utilisateur est obtenue en appelant la fonction `get_answer`.

Si la réponse de l'utilisateur est "oui", la fonction retourne une nouvelle sous-base de données qui contient uniquement les éléments avec la valeur d'attribut correspondante. Si la réponse de l'utilisateur est "non", la fonction crée une nouvelle sous-base de données qui contient tous les éléments sauf ceux avec la valeur d'attribut correspondante. Si la réponse de l'utilisateur est "je ne suis pas sûr", la fonction retourne la base de données originale et ajoute l'attribut à l'ensemble `uncertain_attributes`.

Si toutes les réponses sont "je ne suis pas sûr", la fonction retourne la base de données originale et ajoute l'attribut à l'ensemble `uncertain_attributes`.

La fonction retourne la nouvelle sous-base de données ainsi que l'ensemble `uncertain_attributes`.

Toutes ces fonctions sont construites séparément pour être ensuite appelées au besoin (dans le même esprit que Java) dans la fonction finale. De plus, cela permet l'ajout de nouvelle variable catégorielle dans le code il suffit juste de rajouter la questions dans `question_map()`.

Après toutes nos tentatives, le code fonctionne donc bien. Cependant, alors que vous lisez ces lignes, nous cherchons à le reconnecter à l'interface Java.

CONCLUSION

Dans le cadre de ce projet, nous avons eu à réaliser un jeu de devinettes inspiré du célèbre Akinator. Au cours des deux mois que nous avons eu à disposition pour y arriver, nous avons donc dû répondre à un ensemble de problématiques théoriques pour imaginer le jeu et ses éléments et surtout, des problématiques concrètes pour mettre en oeuvre ce "désir" de jeu.

Nous avons ainsi travaillé sur trois grands axes de réflexion. Dans un premier moment, nous avons cherché à définir l'idée d'une base de données pertinente que nous pensions être les personnages d'animes. Nous avons ensuite mis en place un ensemble de stratégie pour rechercher, récolter et mettre en place cette base de données. Si certaines pistes ont donné des résultats convaincants jusqu'à des étapes avancées, nous avons finalement préféré retenir un corpus qui avait déjà fait ses preuves et une source de données fiable. Nous avons retenu les Pokémons.

Dans un deuxième moment, nous avons étudié la notion d'entropie à l'oeuvre derrière le jeu d'Akinator et plus généralement associé au choix dans les systèmes par arbres de décision. Ce point théorique nous a permis d'assurer la base rationnelle du jeu tout en nous permettant d'assimiler ce concept austère en lui donnant une application claire et ludique. A cette étape, le concept reste néanmoins encore très complexe. Ce projet n'est donc que la première étape vers la maîtrise de cette notion pourtant au centre d'une des avancées les plus spectaculaires de notre temps : l'intelligence artificielle.

Enfin, dans un dernier moment, nous avons tout simplement cherché à mettre en oeuvre ce jeu en implémentant via python les concepts et formules étudiées, à interagir avec la base de données créée et enfin à produire une interface via Java qui puisse offrir au joueur un jeu ergonomique en plus d'être efficace. En cela que le jeu fonctionne et que l'interface existe, cette dernière phase a été globalement concluante... A cela prêt que nous n'avons pas réussi à "reconnecter" le jeu et son interface. Le temps nous manquant vous n'aurez donc qu'un jeu prototype mais fonctionnel avec la promesse de ce à quoi il aurait ressemblé en version définitive.

En conclusion, nous achevons ce projet avec la satisfaction de proposer un résultat fonctionnel. Néanmoins, cette conclusion s'opère avec le bémol de n'avoir finalement qu'amorcer une réflexion et un développement qui laisse de côté des fonctionnalités supplémentaires (la possibilité au jeu d'ajouter des nouvelles réponses, de s'autocorriger, de prendre en compte les joueurs indécis ou menteurs...). Nous formons donc le voeu de trouver le temps et les ressources de mener au bout des ses possibilités ce projet.

BIBLIOGRAPHIE

Bases de données animes

<https://myanimelist.net/character.php>

<https://www.kaggle.com/datasets/mjrone/anime-character-traits-dataset>

Base de données Pokémon

<https://pokeapi.co/>

Entropie et arbres de décisions

<https://sapt.fr/wp-content/uploads/poly-1.pdf>

<https://scaron.info/doc/intro-arbres-decision/>

https://www.youtube.com/watch?v=nAA7UyiCIOE&ab_channel=Wandida%2CEPFL

https://www.youtube.com/watch?v=DSdTiFc3Aws&t=12s&ab_channel=Wandida%2CEPFL

ICONOGRAPHIE



Interface du jeu