Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

FILIÈRE
INFORMATIQUE ET SYSTÈMES DE COMMUNICATION

---

Bachelor of Science HES-SO
Réseaux et systèmes

# Programmation élégante en GO
# Rapport de mini-projet


Superviseur:
Jacques Supcik

Rédaction :
Martin Roch-Neirey
William Margueron


Version : 1.0

Année académique 2022-2023

# Table des matières

# Chapitre 1
# Introduction

Dans le cadre de ce mini-projet, nous avons comme objectif principal d'améliorer un jeu de Tic Tac Toe sur la base d'un programme déjà existant.

Le programme se base sur l'implémentation du dépôt *LempekPL/GoTicTacToe* disponible sur github. Il s'agit d'une version de base utilisant la bibliothèque 2D *Ebitengine*.

Les objectifs de base sont les suivants :

- Tester, mettre à jour et analyser le code.
- *Refactoring* du code.
- Générer des images via la librairie *fogleman/gg*.
- Mettre en place une IA.
- Mettre en place un CI/CD pour valider les tests unitaires.
- Créer une *release* en version exécutable.
- Documenter et créer ce petit rapport.

Légende :

- ⏫ Nouvelle fonctionnalité implémentée
- 💡 Idée qui n'a pas été implémentée

# Chapitre 2
# Analyse et Fonctionnalités

---

Ce chapitre commence dans un premier temps par décrire l'état du programme initial, puis il liste l'ensemble des fonctionnalités présentes dans la dernière version du jeu. Pour chacune, il est précisé s'il s'agit d'une fonctionnalité déjà existante ou d'une nouveauté (voir légende page précédente).

## 2.1 Analyse initial

Après une analyse du code proposé, nous pouvons émettre plusieurs remarques concernant ses forces et ses faiblesses, ainsi que l'élégance de ce projet.

### 2.1.1 Forces

Le code implémente de façon correcte les fonctions proposée par le moteur *Ebitengine* pour la gestion de la logique et de l'affichage.

### 2.1.2 Faiblesses

Un point qui attire rapidement notre regard, est le fait que le code ne présente aucun commentaire. Pour les parties de code ambigu, des commentaires seraient appréciables. Un second point est la présence de *magic numbers*, ce qui n'est pas très élégant. Une bonne pratique serait de mettre des constantes. Un troisième point est l'algorithme de détection de victoire, ce dernier ne fonctionne pas dans certains cas. Il est donc nécessaire de le retravailler. Globalement, certaines instructions sont mal utilisées et la syntaxe de Go n'est pas toujours respectée.

### 2.1.3 Sythèse

Avec ces quelques points relevés, nous avons décidé de repartir sur une nouvelle base pour le projet. Il nous faut réorganiser le répertoire du projet et découper celui-ci en plusieurs dossiers et fichiers.

## 2.2 Jeu de base

Le projet est restructuré en utilisant un dossier **pkg** qui contient deux sous dossier.
- **api** contient les fonctionnalités ajoutées telles que l'intégration du système de sauvegarde et le système de multi-langues (voir sections ci-après).
- **tictactoe** contient les fichiers qui gèrent le jeu:

- **gameLogic.go** contient les fonctions de gestion de la logique du jeu. La fonction *Update* d'*Ebitengine*, la détection des entrées du joueur et la détection d'une fin de partie sont quelques exemples.
- **gameData.go** contient les variables, les structures et les constantes utilisées par le jeu.
- **gameRender.go** contient les fonctions de rendu du jeu. Les méthodes pour dessiner les symboles, le plateau de jeu, du texte ou les barres gagnantes sont présentes dans ce fichier.
- **gameAI.go** contient les fonctions pour le calcul du placement par l'IA. Une IA aléatoire et une IA imbattable utilisant l'algorithme *Minimax* sont implémentées dans le jeu.

Le dossier **cmd/tictactoe** contient le fichier **main.go** qui permet de lancer le jeu.

### 2.2.1 Système multi-langues

Pour proposer une internationalisation du jeu, nous avons développé une API de traduction. Le joueur a la possibilité de choisir sa langue et l'affichage est modifié automatiquement en conséquence. Pour cela, un fichiers *translations.json* contient toutes les traductions, dans toutes les langues du jeu. Chaque traduction possède un identifiant et une valeur. Une langue a la possibilité d'avoir une langue de secours (*fallback language)*. Cela signifie par exemple que si une traduction n'est pas définie en anglais, alors le message est affiché en français. La langue française est celle de plus bas niveau, elle n'a donc pas de langue de secours. Si une traduction française est manquante, alors un message d'erreur est affiché au joueur : *Missing Translation ([id de la traduction manquante)]*. Cette API est disponible dans le fichier *pkg/api/translationsAPI.go*.

### 2.2.2 Système de sauvegarde

Nous avons eu l'idée d'implémenter un système de sauvegarde des parties effectuées partagé via une base de données. Pour cela, nous avons créé une API basée sur la technologie de base de données relationnelle MySQL (disponible dans le fichier *pkg/api/mysqlAPI.go*. Pour le développement du mini-projet, nous nous sommes limités à déployer la base de données sur l'ordinateur hôte. Pour la démonstration et lors de la lecture de ce rapport, une base de données a été installée sur un serveur pour être accessible de partout. Le jeu est conçu pour gérer de lui-même la connexion à la base de données et agir en conséquence si cette dernière n'est pas disponible.

Pour gérer la connexion à la base de données, un test de connexion est effectué au démarrage du programme. Si la connexion est possible, alors les fonctionnalités liées à la base de données sont activées. Si la connexion n'est pas possible, alors les fonctionnalités sont bloquées et invisibles pour le joueur. Vous pouvez essayer en modifiant l'addresse IP de la base de données dans le fichier *pkg/api/mysqlAPI.go* et en mettant n'importe quelle autre valeur. Lorsque

le joueur se situe sur un menu qui requiert l'action d'une base de données, les requêtes sont effectuées autant de fois que le menu est rafraîchit par le moteur *ebitengine* ! Pour éviter du trafic réseau inutile et que la base de données soit surchargée (car les menus sont en moyenne mis à jour 60 fois par seconde), nous avons implémenté un système de cache. Une "vraie" requête SQL est effectuée à la base de données toutes les 10 secondes, et les valeurs récupérées sont mises en cache et utilisées sur les 10 secondes suivantes, jusqu'au prochain appel SQL.

» Les joueurs peuvent désormais voir le nombre total de parties jouées (lorsque ces dernières ont pu être sauvegardées en base de données).

» La totalité des données de chaque partie est sauvegardée en base de données. On peut ainsi retrouver les valeurs du plateau de jeu, les placements, le vainqueur de la partie, le mode de jeu (multiplayer, IA, IARandom), ...

» Les 5 dernières parties jouées sont disponibles dans un menu dédié. Les informations suivantes sont indiquées :

- Mode de jeu
- Vainqueur
- Plateau de jeu

» Avec ce système de sauvegarde, nous avons installé un dashboard Grafana pour visualiser certaines informations :

- URL : `https://graphs.leichap.dev` (serveur de Martin)
- User : demo
- Password : la ville dans laquelle l'école se trouve, en minuscules.
- Dashboard : TicTacToe

# Chapitre 3
# Idées d'amélioration

---

Voici certaines améliorations que nous avons pensé pour rendre le jeu encore plus amusant !

- Implémenter une version du Tic-Tac-Toe quantique, qui est une version modifiée qui invite les joueurs à comprendre certains principes fondamentaux de la mécanique quantique (comme par exemple la superposition d'états d'un objet quantique tant qu'il n'est pas mesuré) : `https://fr.wikipedia.org/wiki/Tic-tac-toe_quantique`

- Implémenter un réel mode multijoueur avec 2 ordinateurs différents. Cette fonctionnalité nécessite une architecture radicalement différente de celle actuelle. Il faudrait probablement créer un serveur de jeu qui synchroniserait les actions des 2 joueurs. Les clients n'auraient que la logique de clic et d'affichage, tandis que le serveur calculerait les données du jeu à afficher et les renverrais aux clients.

- Observer sur le dashboard Grafana différentes statistiques, comme le nombre de victoires de l'IA Random, ou le mode le plus joué...

- Remonter les erreurs logicielles dans Sentry. Il s'agit d'un outil qui permet d'enregistrer et de traiter les erreurs (`https://sentry.io`).

- Instaurer un système de profile de joueur, avec un pseudonyme et un mot de passe, les statistiques du joueur, ses dernières parties, ses derniers ennemis...

- Pour augmenter l'accessibilité du jeu, proposer en plus des différentes langues, un système de changement d'affichage. Par exemple, le joueur pourrait sélectionner un mode de couleurs qui afficherait le jeu de manière à ce que les daltoniens ne soient pas impactés.

- Utiliser l'API de ChatGPT pour jouer contre l'IA d'OpenAI. Nous avons essayé d'implémenter celle solution mais... ChatGPT ne sait pas jouer au Tic Tac Toe... 😐

# Chapitre 4
# Conclusion

---

Ce mini-projet a été très amusant à réaliser. Dès le début, nous nous sommes complètement pris au jeu et avons décidé d'ajouter plusieurs fonctionnalités importantes (multi-langues, lien avec une base de données...). Nous avions déjà conscience de la puissance du langage Go, et ce projet a permis d'appuyer ce point de vue. Pour avoir comparé de nombreux avis sur Internet, il ressort souvent que Go est un langage simple à lire et à comprendre tout en restant extrêmement puissant, et avec lequel les programmeurs ont souvent plaisir à travailler. Nous partageons tout à fait ce point de vue.

Nous pensons avoir produit un jeu avec une qualité satisfaisante, tant au niveau du code que de l'expérience utilisateur. Le jeu en lui-même est basique, mais les quelques fonctionnalités d'accessibilité et de sauvegarde permettent d'ajouter du contenu intéressant !