UNIVERSITÉ SORBONNE PARIS NORD

SPECIF. DES SYSTEMES COMPLEXES

Rapport du Model Checker CTL

Prénom	*Nom*	*N° étudiant*
Mohand Lounis	**BENSEKHRI**	11710457
Salem	**BOUDEBABA**	12017165
Younes	**IBIZZI**	12212407
Bingqin	**ZHANG**	11707768

Formation : M2 – PLS

Année universitaire : 2022 / 2023

M. Kaïs **KLAI**

TABLE DES MATIERES

INTRODUCTION

Dans le cadre de l'UE (Unité d'Enseignement) Spécification des systèmes complexes, il nous a été demandé de réaliser un projet. L'objectif de ce travail a été de mettre à profit l'ensemble des connaissances que nous avons acquises sur les formules CTL au cours de ce semestre pour réaliser un Model Checker CTL .

Ce projet a été réalisé par Lounis, Salem, Younes et Bingqin. Pour ce faire, il nous a fallu développer un parseur afin de reconnaître ce que l'utilisateur saisit sur le terminal. Puis, nous avons dû traiter les informations saisies, tout en respectant les contraintes syntaxiques des formules CTL. Notons que cette première partie est essentielle au bon fonctionnement par la suite. En effet, avec la bonne saisie des formules CTL, il sera possible de les tester sur une structure de Kripke.

Pour l'écriture du code, on a décidé d'utiliser le javaCC pour le parseur, et le Java pour le code grâce à l'IDE Intellij IDEA sous Maven.

Afin de mieux comprendre la démarche entamée, nous commencerons par présenter les différentes parties du projet. Puis, nous présenterons la marche à suivre pour utiliser notre Model-Checker. Et enfin nous énumérerons les difficultés et problèmes rencontrés.

PRESENTATION DU PROJET

1. Parseur

Le parseur a été fait avec javaCC, au niveau de la syntaxe les labels sont exclusivement en minuscule (une lettre ou plus) suivie éventuellement de chiffres, tandis que les majuscules (seulement certaines) sont spécifiquement dédiées aux formules CTL, tout comme certains caractères spéciaux.

Ces dernières inclus E : Il existe, A : pour tout, X : Next, F : Futur, G : Globally, U : Until, /\ : Et, \/ : Ou, => : implique, ~ : Not, TRUE : true, FALSE : false et enfin les ( ) doivent être utilisée pour la lisibilité.

Notre parseur reconnaît la syntaxe de tous types de formule CTL, mais ne donne que les 6 types de formules de base en sortie. A cet effet, nous avons établi les équivalences pour les formules non reconnues (cf. Formules CTL).

Pour la structure syntaxique, nous avons une formule CTL qui est soit une proposition liée avec un quantificateur (A, E), soit une simple proposition.

Puis, une proposition contient soit un terme, soit plusieurs termes liés avec des symboles logiques (/\, \/, =>).

Et enfin, un terme est soit atomique, soit une valeur de vérité, soit une négation, ou alors contient des parenthèses.

2. Structure de Kripke

Pour gérer notre structure de Kripke, nous avons créé une classe KripkeStr qui est composée d'arcs et d'états.

Les arcs possèdent les attributs source et destination qui sont des entiers qui représentent les index des états souhaités.

Les états possèdent un nom, une liste de labels, un booléen (état initial), une liste de successeurs et de prédécesseurs, ainsi qu'un index qui s'incrémente automatiquement à chaque création d'un nouvel état.

3. Formules CTL

Pour les formules CTL, notre model-checker ne traite que 6 types de formules. En effet, cela est possible grâce aux équivalences faites dans le parseur.

Nos formules CTL ont deux types celles qui ont besoin de deux arguments (coté gauche et coté droit de la formule CTL) comment par exemple le ET, le OU et le IMPLIQUE, et celle qui ont besoin que d'un seul argument comme par exemple le NOT et le NEXT.

FONCTIONNEMENT DU MODEL CHECKER CTL

0. Création du JSON de la SK

L'écriture de la SK, sous format JSON, requiert un champ states (état). En effet ici, les états sont représentés sous une forme de liste, avec des champs pour chaque état. Les différents champs sont name pour le nom de l'état, labels une liste de labels et enfin isInitial, comme son nom l'indique, indique si l'état est initial ou non. Notez que les index des états sont incrémentés automatiquement et commencent à 0, et donc nous n'avons pas besoin de les renseigner dans les states.

Puis, la SK requiert aussi un champ arcs. Ce dernier est une liste de couple qui indique pour chaque arc sa source et sa destination. Il faut noter ici que les sources et destinations représentent les index des états. Exemple : pour relier votre premier état avec le second (en sachant que l'ordre des états est celui que vous avez déclaré dans le JSON : le premier est celui qui est tout en haut), il faut écrire [0, 1].

- Voici une image qui montre un exemple d'un fichier JSON qui est KS_example.json dans le projet :

```
{
  "states": [
    {
      "name": "S1",
      "labels": [
        "p",
        "q"
      ],
      "isInitial": true
    },
    {
      "name": "S2",
      "labels": [
        "q"
      ],
      "isInitial": false
    },
    {
      "name": "S3",
      "labels": [
        "p"
      ],
      "isInitial": false
    }
  ],
  "arcs": [
    [
      0,
      0
    ],
    [
      0,
      1
    ],
    [
      1,
      0
    ],
    [
      1,
      2
    ],
    [
      2,
      2
    ]
  ]
}
```

Figure 0 : Exemple d'une SK dans un fichier JSON

1. Structure de Kripke

Pour utiliser notre Model Checker CTL, il faut en premier lieu spécifier la SK que vous souhaitez vérifier sous format JSON, pour débloquer l'accès vers le menu Formules CTL.

Si vous voulez tester une autre structure de Kripke pendant l'exécution de l'application, vous pouvez le faire sans arrêter le programme, et cela, en chargeant directement votre fichier JSON contenant la nouvelle SK via le Menu SK. Sachez toutefois que le fichier JSON doit être bien syntaxé et doit être présent dans le dossier src/main/resources faute de quoi le programme s'arrête avec une erreur. A noter que la petite étoile jaune sur le côté dans le Menu SK signifie qu'une SK a déjà été chargée.

Une fois que la SK est chargée, la possibilité de l'afficher sera alors accessible. Il suffit alors de choisir l'option afficher. On aura ainsi un affichage en console de la SK qui montre l'ensemble des états et arc de celle-ci, mais le plus intéressant dans cette fonctionnalité reste bien sûr la création d'une image PNG représentant la structure de kripke dans le sous répertoire src/main/resources/images portant le nom du fichier JSON depuis lequel nous avons créé notre SK. A noter que pour visualiser le PNG, il vous faudra l'ouvrir manuellement.

- Voici quelques images qui montres l'exécution du projet (partie SK) :

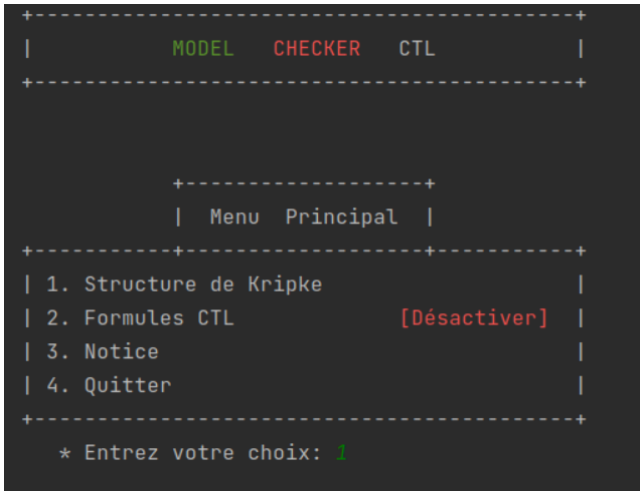

Figure 1 : Menu Principal du Model Checker CTL. Formules CTL [désactiver] car aucune SK n'est renseignée. On peut accéder à la notice du Menu. On peut quitter l'application.

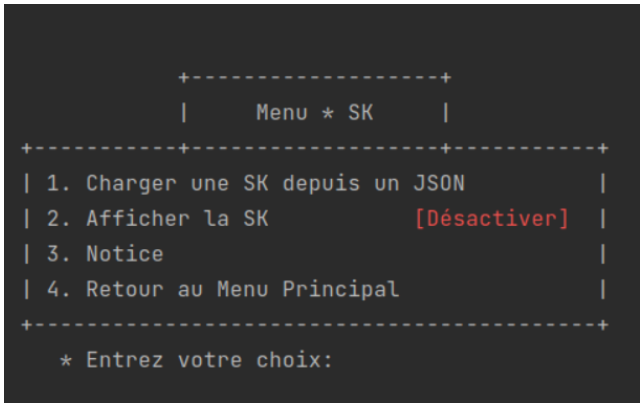

Figure 2 : Menu de la SK. On peut charger une SK depuis un JSON. la fonctionnalité Afficher la SK est [désactiver] car aucune SK n'est encore renseigné. On peut consulter la notice du Menu SK. On peut revenir au Menu Principal

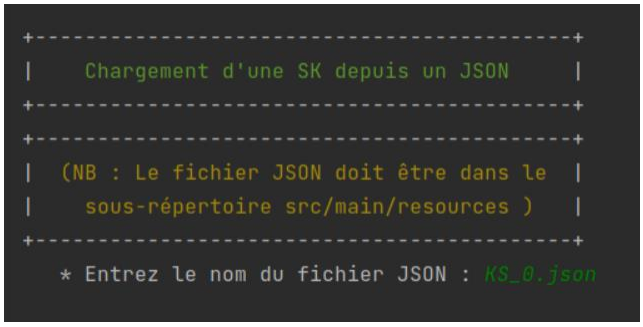

Figure 3 : On charge une SK depuis [KS_0.json].

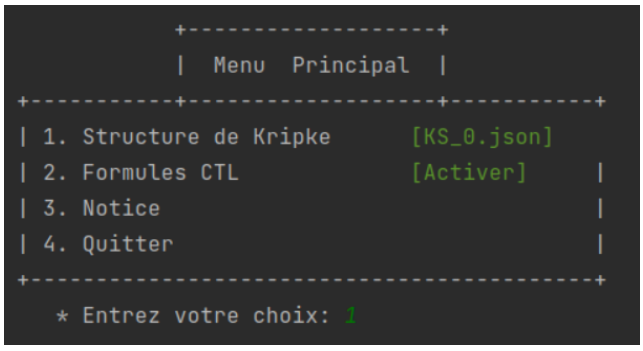

Figure 4 : Menu Principal. On remarque qu'on a chargé une SK depuis le JSON [KS_0.json]. le Menu CTL est maintenant accessible.

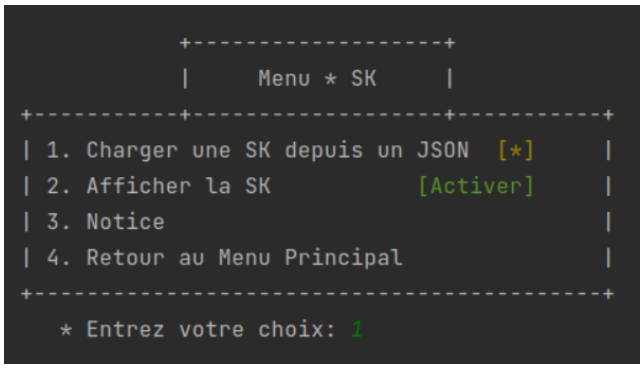

Figure 5 : Menu SK. l'étoile veut dire qu'on a déjà chargé une SK. Mais on peut toujours charger une nouvelle qui écrasera la première et ainsi on utilisera la nouvelle dans l'application.

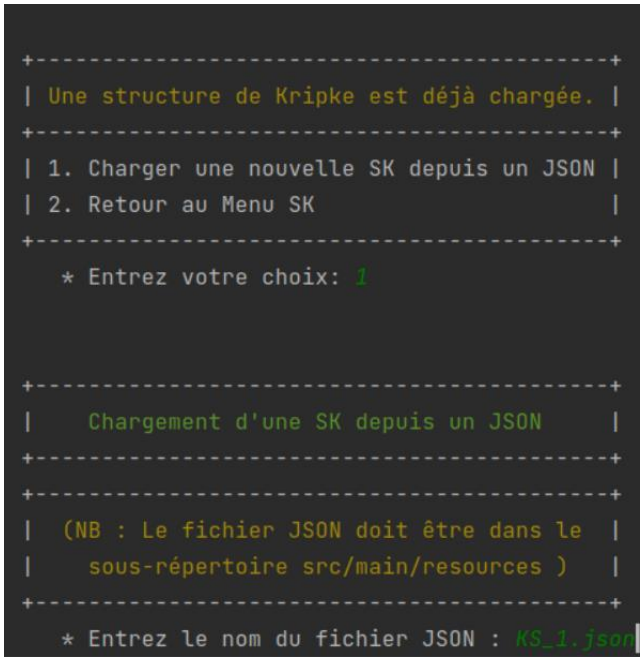

Figure 6 : Ça nous demande confirmation de chargement d'une nouvelle SK depuis un autre JSON dans ce cas [KS_1.json].

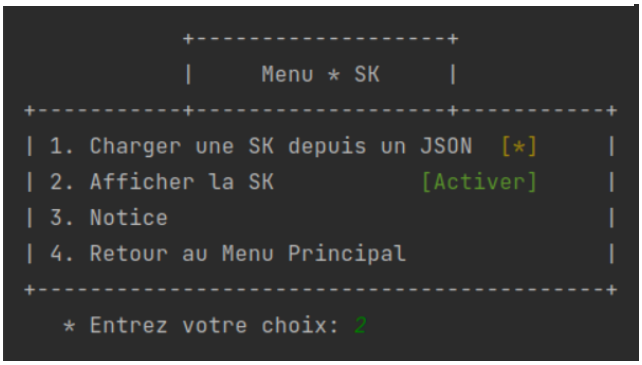

Figure 7 : Affichage d'une SK en tapant 2 depuis le Menu SK.

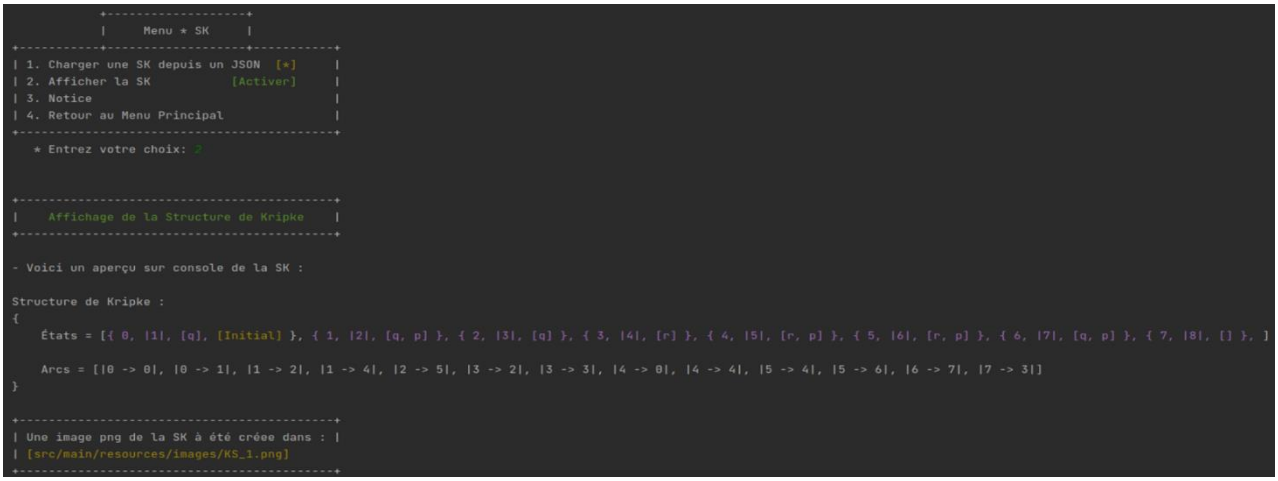

Figure 8 : Affichage de la SK du fichier [KS_1.json]. On voit l'affichage sur console. On remarque le message indiquant qu'il a créé une image PNG de la SK.

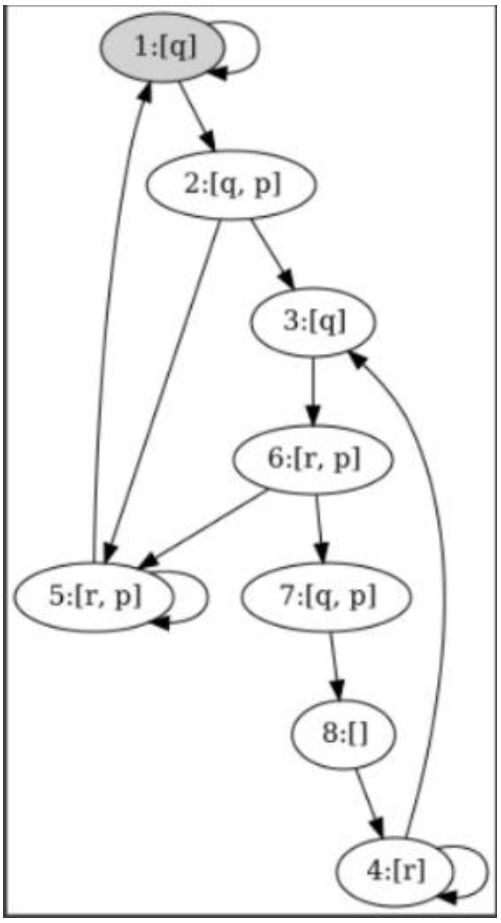

Figure 9 : Image PNG nommé [KS_1.png] représentant la SK [KS_1.json].

2. Formules CTL

Après avoir chargé la SK Vous pouvez ensuite spécifier les propriétés à vérifier en utilisant des formules CTL que vous taperez directement sur la console. Il faut avant cela retourner sur le menu principal et accéder au Menu - Formule CTL :

- Voici quelques images qui montres l'exécution du projet (partie CTL) :

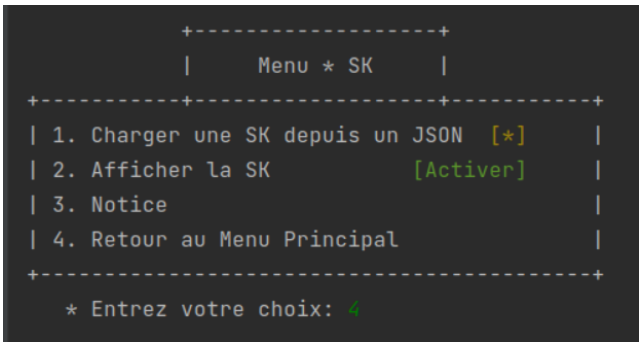

Figure 10 : Retour au Menu Principal

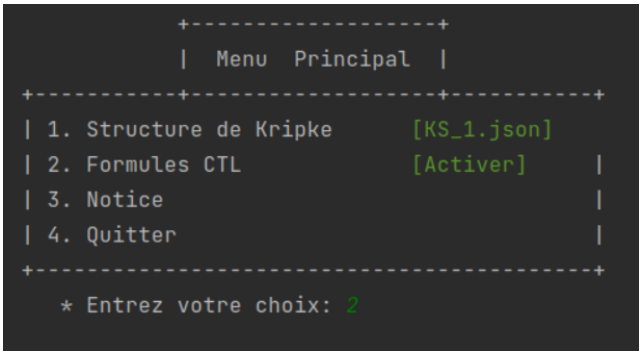

Figure 11 : On remarque bien que la SK [KS_1.json] est déjà chargé. On accède au Menu CTL en tapant 2.

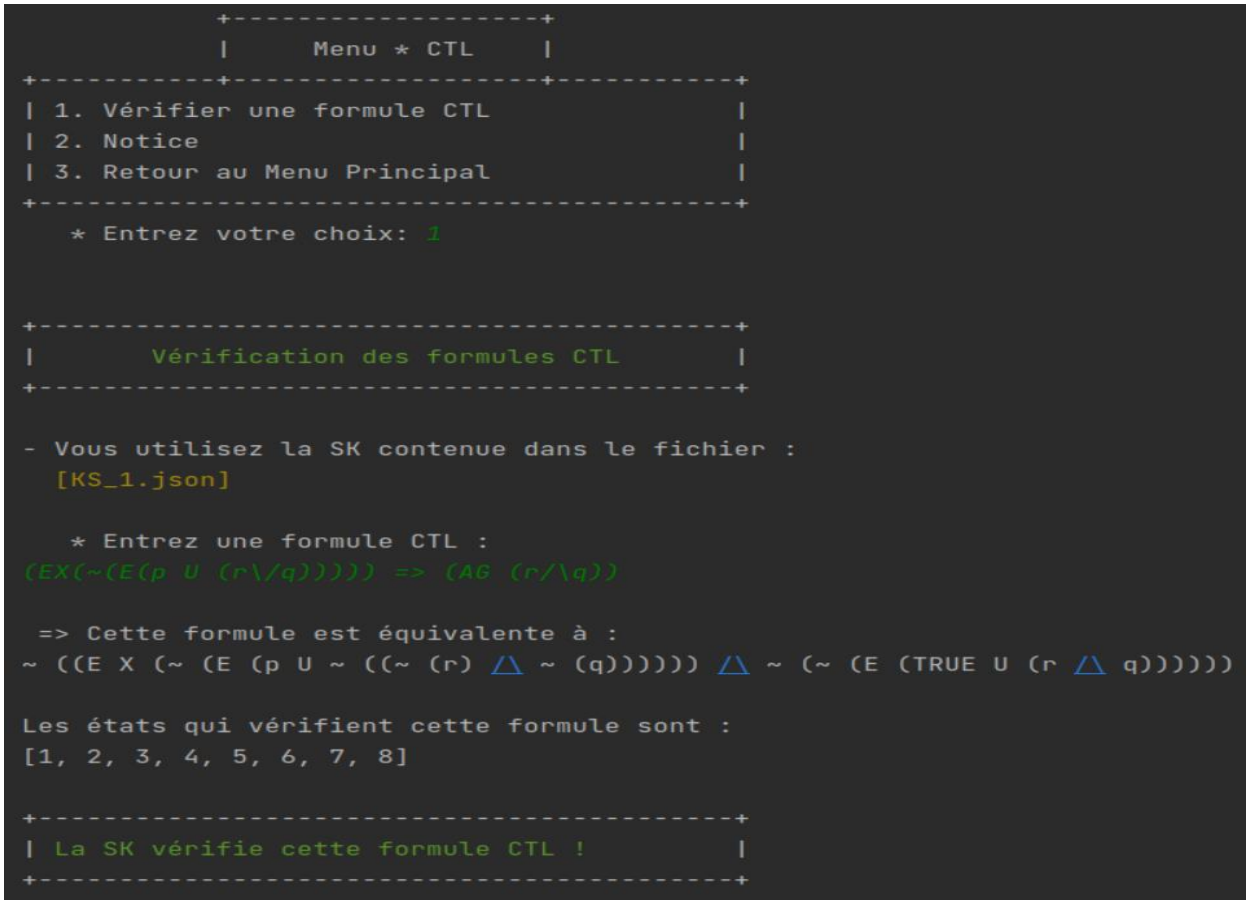

Figure 11 : On vérifie la formule CTL. Le programme nous donne une formule CTL équivalente à celle qu'on a tapé, ainsi que l'ensemble des états qui la vérifient. La phrase « La SK vérifie cette formule CTL ! » est affiché car tous les états de la SK vérifie la formule.

DIFFICULTES ET PROBLEMES RENCONTRES

Au cours de ce projet de vérification de modèles en CTL, notre équipe a rencontré plusieurs défis qui ont rendu le développement du logiciel plus difficile.

Tout d'abord, nous avons eu des problèmes avec le parseur JavaCC, qui a été très difficile à mettre en œuvre en raison de la complexité du langage. De plus, la transformation du fichier JSON en une structure de Kripke a été difficile en raison de la complexité du format de données.

En outre, le chargement de plusieurs fichiers SK pendant une même exécution a également été un défi, car nous devions gérer de manière efficace les dépendances pour éviter les erreurs. De plus, la transformation du graphe en image PNG s'est avérée plus difficile que prévu, car nous devions trouver un moyen de représenter graphiquement les résultats de la vérification des modèles.

Enfin, le temps de travail a également été un problème, car nous devions gérer nos emplois du temps pour être en mesure de travailler ensemble en équipe sur le projet. Malgré ces défis, nous avons continué à travailler ensemble pour développer un logiciel de vérification de modèles efficace et fiable, qui peut être utilisé pour valider les modèles de différentes tailles et de différents types.

CONCLUSION

En conclusion, ce projet de vérification de modèles en CTL a été un défi pour notre groupe, mais nous avons relevé ce défi avec succès en utilisant Java et Maven. Au début, nous avons rencontré des difficultés, mais en travaillant ensemble en équipe, nous avons surmonté ces obstacles et atteint notre objectif. Nous sommes fiers de ce que nous avons accompli et de la manière dont nous avons résolu les problèmes rencontrés en cours de route. Ce projet a renforcé nos compétences en programmation et en travail d'équipe, et nous sommes impatients de continuer à développer nos compétences dans ce domaine.