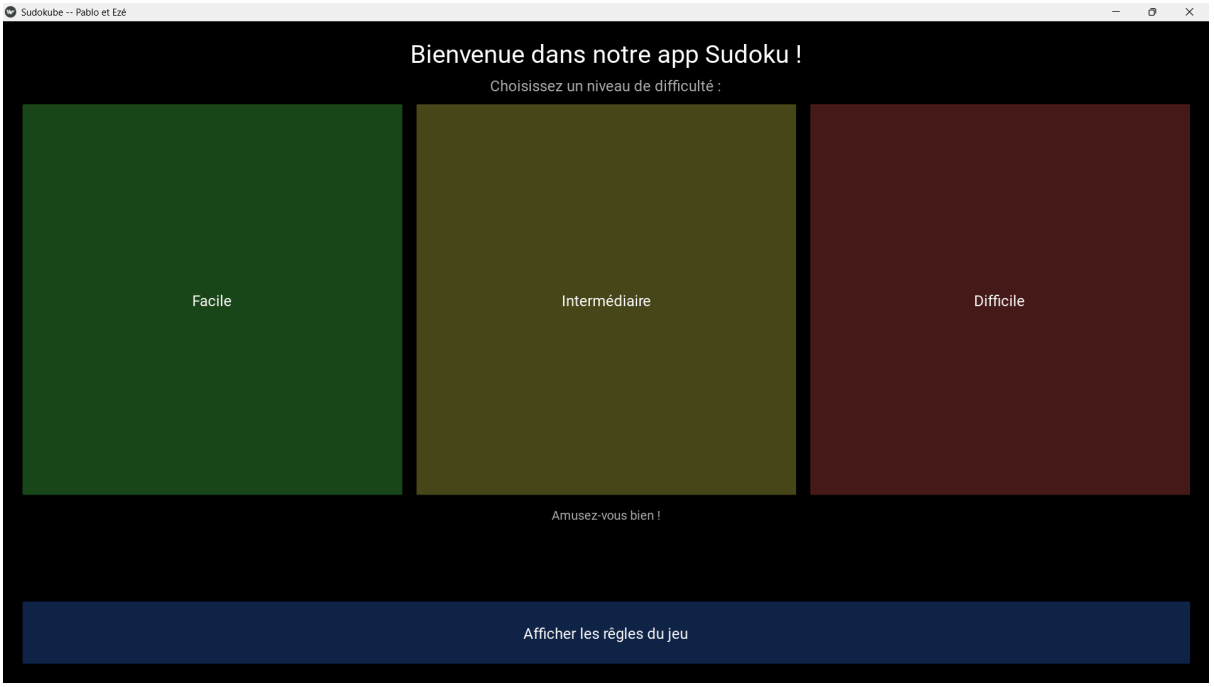
Sudokube



RENAULT Ezéchiel
FAIVRE Pablo
INFO - Année 2024-2025

I. Table des matières

II. Présentation du projet

La société multinationale **Epic-Steam** souhaite élargir son champ d'action en développant une application novatrice qui combine un **générateur** et une **interface de jeu de Sudoku**. Avec cette initiative, la firme ambitionne de s'imposer dans l'univers exigeant et compétitif du **développement de jeux vidéo**. Pour concrétiser ce projet, elle a fondé une nouvelle branche dédiée : **Sudokube**, qui deviendra le cœur de ses activités dans le domaine vidéoludique.

Pour ce premier projet, l'objectif est de concevoir une application permettant aux utilisateurs de résoudre des grilles de Sudoku générées automatiquement, tout en bénéficiant d'une **interface claire, moderne et intuitive**. L'ambition est de faire forte impression auprès des joueurs dès le lancement, afin de générer un engouement autour de Sudokube et de positionner la startup comme un acteur prometteur dans l'industrie du jeu vidéo.

Conscient de vos compétences et de votre expertise dans le domaine, le directeur de cette jeune entreprise vous a sélectionné comme deux des premiers développeurs pour piloter ce projet. Votre mission est de créer une application qui allie **simplicité, efficacité et attractivité**, en offrant une expérience utilisateur motivante et accessible, et ce, dans des délais courts.

Pour garantir une expérience de jeu optimale, le directeur marketing a rédigé un cahier des charges spécifiant les **fonctionnalités essentielles** nécessaires pour se démarquer sur le marché :

- **Un générateur de Sudokus aléatoires** fonctionnant localement sur le PC du client, en l'absence de serveurs en ligne à ce stade.
- **Plusieurs niveaux de difficulté**, permettant de s'adresser à un large public, des joueurs débutants aux experts en quête de défi.
- **Un menu de règles** succinct mais clair, expliquant les bases du jeu aux nouveaux joueurs.
- **Un gameplay fluide et intuitif**, rendant la résolution des grilles agréable et immersive.
- **Une compatibilité multiplateforme** (PC, Smartphone Android et Appareils IOS) permettant de toucher un large public, incluant des utilisateurs non initiés au gaming.

En parallèle, le designer en chef a défini des **exigences visuelles** destinées à rendre l'application attrayante et moderne :

- **Des palettes de couleurs** dynamiques, pour offrir une ambiance contemporaine et plaisante.
- **Des illustrations et images** soigneusement choisies, renforçant l'identité visuelle du jeu.
- **Des transitions fluides**, même sur des machines modestes, afin d'assurer une expérience utilisateur homogène.
- **Un redimensionnement dynamique de la fenêtre**, garantissant une adaptation parfaite à tous les types d'écrans et résolutions.
- **Un écran de victoire engageant**, conçu pour motiver les joueurs à rejouer et à relever de nouveaux défis.

Ensemble, ces éléments permettront à Sudokube de poser les bases solides d'un **jeu captivant et bien conçu**, offrant une expérience de qualité, même sans infrastructure serveur. Grâce à une exécution soignée et fidèle à ce cahier des charges, cette application pourra marquer un premier pas réussi pour Epic-Steam dans le monde des jeux vidéo.

III. Choix effectués

Pourquoi Kivy?

1. Kivy : Idéal pour les interfaces modernes et multiplateformes

- **Multiplateforme** : Kivy est conçu pour fonctionner sur plusieurs plateformes (Windows, macOS, Linux, Android, iOS) avec un code unique. C'est parfait si vous envisagez de rendre Sudokube disponible sur mobile.
- **Interface utilisateur moderne** : Kivy fournit des widgets riches et personnalisables, parfaits pour créer des interfaces élégantes et interactives, avec des animations intégrées.
- **Gestion tactile** : Kivy gère naturellement les interactions tactiles, ce qui est crucial pour les appareils mobiles et les tablettes.
- **Performances graphiques** : Basé sur OpenGL ES 2, Kivy offre des graphismes fluides et rapides, adaptés aux applications visuellement riches.

---

2. Pygame : Spécialisé dans les jeux, mais limité pour les interfaces

- **Pas d'interface utilisateur intégrée** : Pygame manque de widgets ou d'éléments d'interface utilisateur (boutons, menus, champs de texte), nécessitant leur implémentation manuelle.
- **Pas adapté pour les applications multi plateformes modernes** : Pygame est limité aux ordinateurs de bureau et ne prend pas en charge nativement les interactions tactiles.
- **Graphismes moins modernes** : Comparé à Kivy, il est plus difficile de produire des interfaces esthétiques et dynamiques.

---

3. Tkinter : Simple mais peu moderne

- **Apparence datée** : Les widgets de Tkinter ont un aspect classique qui peut ne pas correspondre à une interface moderne et immersive.
- **Limitations graphiques** : Pas de support pour les animations avancées ou des graphiques fluides.
- **Pas idéal pour le tactile ou le multiplateforme** : Contrairement à Kivy, Tkinter n'est pas conçu pour les écrans tactiles ou pour fonctionner de manière optimale sur des appareils mobiles.

Résumé du choix : Pourquoi Kivy ?

Notre objectif est de créer une **application moderne**, visuellement attrayante, fonctionnant sur **plusieurs plateformes**, et intégrant des fonctionnalités tactiles, **Kivy** est donc un choix naturel.

Pourquoi l’algorithme de Backtracking?

On a choisi l’algorithme de Backtracking, car il n’existe pas d’algorithme très efficace pour générer des Sudokus. Ce sont toujours des algorithmes se basant sur le brutforce, c’est-à-dire en testant toutes les combinaisons possibles jusqu’à en trouver une qui fonctionne.

Le Backtracking est particulièrement adapté pour ce type de problème, car il permet de construire la solution pas à pas en essayant différentes possibilités et en revenant en arrière dès qu’une contrainte est violée. Cela permet de réduire considérablement l'espace de recherche par rapport à une approche exhaustive. De plus, il est relativement simple à implémenter et offre une flexibilité pour ajuster les niveaux de difficulté en modifiant le nombre de cases initialement remplies.

Pourquoi le Backtracking ?

L'algorithme de Backtracking est une méthode récursive qui essaie de construire une solution à un problème en construisant progressivement une solution candidate, et en abandonnant toute solution partielle dès qu'il devient évident que cette solution ne peut pas être prolongée pour former une solution complète. Dans le cadre du Sudoku, cela signifie que nous plaçons un chiffre dans une case vide et vérifions si cela conduit à une solution valide. Si ce n'est pas le cas, nous essayerons un autre chiffre. Si aucun chiffre ne fonctionne, nous revenons en arrière (backtrack) et modifions les chiffres précédemment placés.

Avantages du Backtracking

1. **Simplicité d'implémentation** : L'algorithme de Backtracking est relativement simple à comprendre et à mettre en œuvre. Il ne nécessite pas de structures de données complexes ou de concepts mathématiques avancés. Cela permet de se concentrer sur la logique du jeu et les règles du Sudoku.

2. **Flexibilité** : Le Backtracking permet d'adapter facilement l'algorithme pour générer des grilles de différentes tailles et niveaux de difficulté. En ajustant simplement le nombre de cases initialement remplies, nous pouvons créer des puzzles allant de très faciles à extrêmement difficiles.

3. **Efficacité pour les grilles de Sudoku** : Bien que le Backtracking puisse être inefficace pour des problèmes plus généraux, il fonctionne étonnamment bien pour les grilles de Sudoku en raison de la nature des contraintes du Sudoku. Les contraintes strictes du Sudoku (chaque chiffre doit être unique dans chaque ligne, colonne et sous-grille 3x3) réduisent considérablement l'espace de recherche, ce qui permet à l'algorithme de trouver des solutions rapidement dans la plupart des cas.

Inconvénients et Limitations

Bien que le Backtracking soit efficace pour le Sudoku, il y a quelques inconvénients à prendre en compte :

1. **Complexité de temps** : Pour des grilles de Sudoku particulièrement difficiles ou des configurations spécifiques, le Backtracking peut devenir très coûteux en termes de temps de calcul. Dans le pire des cas, l'algorithme peut avoir à explorer toutes les combinaisons possibles avant de trouver une solution, ce qui peut être décourageant pour l'utilisateur.

Alternatives envisagées

Avant de choisir le Backtracking, nous avons envisagé plusieurs autres approches, notamment :

1. **Algorithmes probabilistes** : Ces algorithmes utilisent des techniques comme la recherche aléatoire et les heuristiques pour trouver des solutions. Bien qu'ils puissent être plus rapides dans certains cas, ils ne garantissent pas toujours une solution valide et peuvent nécessiter des ajustements fins des paramètres pour chaque grille.

2. **Algorithmes basés sur des contraintes** : Ces algorithmes utilisent des techniques avancées de programmation par contraintes pour résoudre le Sudoku. Bien qu'ils soient très puissants et puissent résoudre des grilles très difficiles, ils sont également beaucoup plus complexes à implémenter et à comprendre.

3. **Possibilité de passer à un autre langage de programmation plus rapide** comme le C ou le C++ par exemple, qui sont des langages de bas niveau et plus de 100x plus rapide que le Python, connu pour sa lenteur.

En conclusion, bien que le Backtracking puisse être chronophage pour des grilles de grande taille ou des niveaux de difficulté élevés, il reste une méthode fiable et efficace pour générer et résoudre des petites grilles de Sudoku (9x9 dans notre cas) de manière algorithmique. Sa simplicité d'implémentation et sa flexibilité en font un choix idéal pour notre projet, permettant de créer une application de Sudoku intuitive et agréable à utiliser.

IV. 11 fonctions essentielles

Voici une analyse de onze fonctions essentielles du programme, avec une variation dans leur complexité, ainsi qu'une explication des fonctions récursives. Toutes les fonctions viennent du programme backend.py

1. Classe Box et méthode __init__

Objectif :

- Initialiser une grille vide de Sudoku.

Paramètres:

- **self**: Box - Nécessaire pour utiliser des éléments du type “Class”.
- **value**: int - Valeur de la case lors de son initialisation (vaut 0 si la case est vide).
- **locked**: bool - Valeur pour savoir si la case est verrouillée (car initialisée au début) ou pas. Ce paramètre vaut False par défaut.

Contexte:

- Cette fonction est appelée par défaut lorsqu’on initialise un élément de la classe Box.

Complexite:

- Simple

2. Classe Grid et méthode __str__

Objectif :

- Afficher la grille dans la console lorsqu'un print(grille) est appelé, à la place d’un résultat similaire à <class '__main__.Grid'>. Elle affiche un point (.) si la case est vide, et sinon sa valeur. Cette fonction permet donc, si on le veut, de jouer avec le programme contenu dans backend.py dans sa console, sans utiliser le fichier main.py.

Parametres:

- null

Contexte:

- Cette fonction est appelée par défaut lorsqu’on essaie d’afficher un élément de la classe Grid dans la console avec print(element).

Complexité :

- Simple.

3. Méthode generate de la classe Grid

Objectif :

- Générer une grille de Sudoku partiellement remplie selon le niveau de difficulté.

Paramètres:

- **self**:Grid - Nécessaire pour utiliser des éléments du type “Class”.
- **level**:int - Nombre de cases à vider après la génération de la solution.

Contexte:

- Cette fonction est utilisée pour créer une nouvelle grille de Sudoku avec un certain nombre de cases vides, en fonction du niveau de difficulté spécifié. Elle initialise la grille, remplit quelques cases aléatoires avec des valeurs aléatoires valides, résout la grille, puis vide un nombre de cases correspondant au niveau de difficulté.

Complexité :

- Intermédiaire.

4. Méthode is_valid de la classe Grid

Objectif :

- Vérifier si un élément peut être placé à une position donnée sans violer les règles du Sudoku.

Paramètres :

- **self**:Grid - Nécessaire pour utiliser des éléments du type “Class”.
- **col**:int - Colonne où placer l'élément.
- **row**:int - Ligne où placer l'élément.
- **number**:int - Valeur à placer dans la case.
- **initial**:bool - Indique si la vérification se fait sur la grille initiale. Ce paramètre vaut False par défaut.

Contexte:

- Cette fonction est utilisée pour s'assurer qu'un nombre peut être placé dans une case spécifique sans enfreindre les règles du Sudoku (ligne, colonne, et bloc 3x3).

Complexité :

- Intermédiaire.

5. Méthode find_empty_cell de la classe Grid

Objectif :

- Trouver une cellule vide (valeur 0) dans la grille.

Paramètres:

- **self**:Grid - Nécessaire pour utiliser des éléments du type “Class”.

Contexte:

- Cette fonction est utilisée pour localiser la prochaine cellule vide dans la grille, ce qui est utile pour l'algorithme de résolution par backtracking.

Complexité :

- Simple.

6. Méthode solve de la classe Grid

Objectif :

- Résoudre la grille en utilisant la technique de backtracking.

Paramètres:
- **self**:Grid - Nécessaire pour utiliser des éléments du type “Class”.

Contexte:
- Cette fonction est utilisée pour trouver une solution complète à la grille de Sudoku en remplissant toutes les cases vides avec des valeurs valides. Elle utilise un algorithme de backtracking pour essayer différentes solutions jusqu'à ce qu'elle trouve une solution valide.

Complexité :
- Avancée.
- Récursive

7. Méthode is_allowed de la classe Grid

Objectif :
- Vérifier si une case donnée peut être modifiée par l'utilisateur.

Paramètres:
- **self**:Grid - Nécessaire pour utiliser des éléments du type “Class”.
- **col**:int - Colonne de la case.
- **row**:int - Ligne de la case.

Contexte:
- Cette fonction est utilisée pour déterminer si une case particulière de la grille est modifiable (non verrouillée) par l'utilisateur.

Complexité :
- Simple.

8. Méthode set_element de la classe Grid

Objectif :
- Définir un élément dans la grille si l'emplacement est valide et modifiable.

Paramètres:
- **self**:Grid - Nécessaire pour utiliser des éléments du type “Class”.
- **col**:int - Colonne de la case.
- **row**:int - Ligne de la case.
- **element**:int - Valeur à insérer dans la case.

Contexte:
- Cette fonction est utilisée pour insérer une valeur dans une case spécifique de la grille, à condition que l'emplacement soit valide (ne viole pas les règles du Sudoku) et que la case soit modifiable.

Complexité :
- Intermédiaire.

9. Méthode get_9x9 de la classe Grid

Objectif :

- Retourner la grille sous forme de liste 9x9 d'objets Box.

Paramètres:

- **self**:Grid - Nécessaire pour utiliser des éléments du type “Class”.

Contexte:

- Cette fonction est utilisée pour obtenir la grille de Sudoku sous forme d'une liste 9x9 contenant des objets Box.

Complexité :

- Simple.

10. Méthode get_81 de la classe Grid

Objectif :

- Retourner la grille sous forme de liste linéaire de 81 objets Box.

Paramètres:

- **self**:Grid - Nécessaire pour utiliser des éléments du type “Class”.
- **rb**:int - Bloc de ligne (par défaut 0).
- **cb**:int - Bloc de colonne (par défaut 0).
- **row**:int - Ligne dans le bloc (par défaut 0).
- **col**:int - Colonne dans le bloc (par défaut 0).
- **g**:list[Box] - Liste des cases (par défaut None).

Contexte:

- Cette fonction est utilisée pour obtenir la grille de Sudoku sous forme d'une liste linéaire de 81 objets Box.

Complexité :

- Intermédiaire.
- Récursive

11. Méthode is_solved de la classe Grid

Objectif :

- Vérifier si la grille est résolue.

Paramètres:

- **self**:Grid - Nécessaire pour utiliser des éléments du type “Class”.
- **row**:int - Ligne actuelle (par défaut 0).
- **col**:int - Colonne actuelle (par défaut 0).

Contexte:

- Cette fonction est utilisée pour vérifier si la grille de Sudoku est complètement et correctement résolue.

Complexité :

- Intermédiaire.
- Récursive

V. Focus sur les fonctions récursives

1. solve

- Cette fonction repose sur le backtracking pour explorer toutes les solutions possibles d'une grille. Chaque étape implique :
  - Trouver une case vide.
  - Tenter d'y insérer un chiffre valide.
  - Revenir en arrière si nécessaire (car aucun des chiffres de 1 à 9 ne fonctionne).
- **Atout :** Efficace pour les grilles classiques de Sudoku.
- **Limitation :** Sa complexité augmente rapidement avec la taille de la grille.

2. get_81

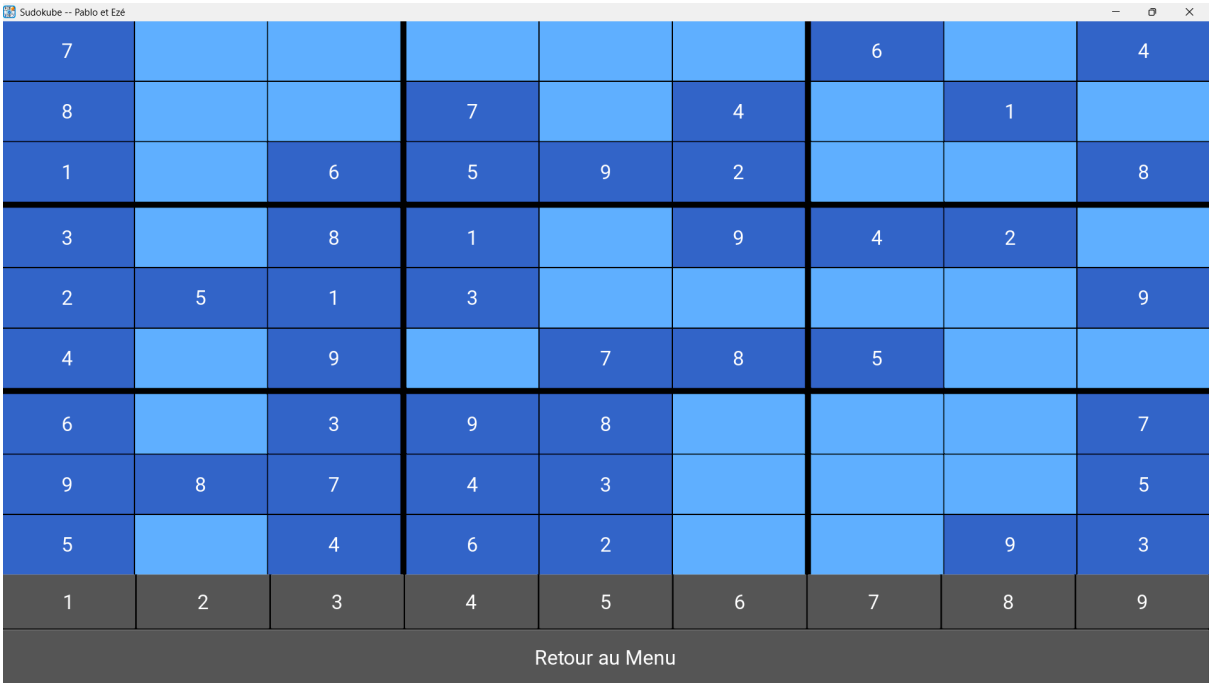
- Elle utilise la récursion pour parcourir la grille bloc par bloc et ligne par ligne. Cela permet de transformer une structure 2D en une liste linéaire, tout en respectant l'organisation des blocs.
- **Atout :** Organisation systématique des données.
- **Limitation :** La logique imbriquée peut être difficile à suivre pour les débutants.

3. is_solved

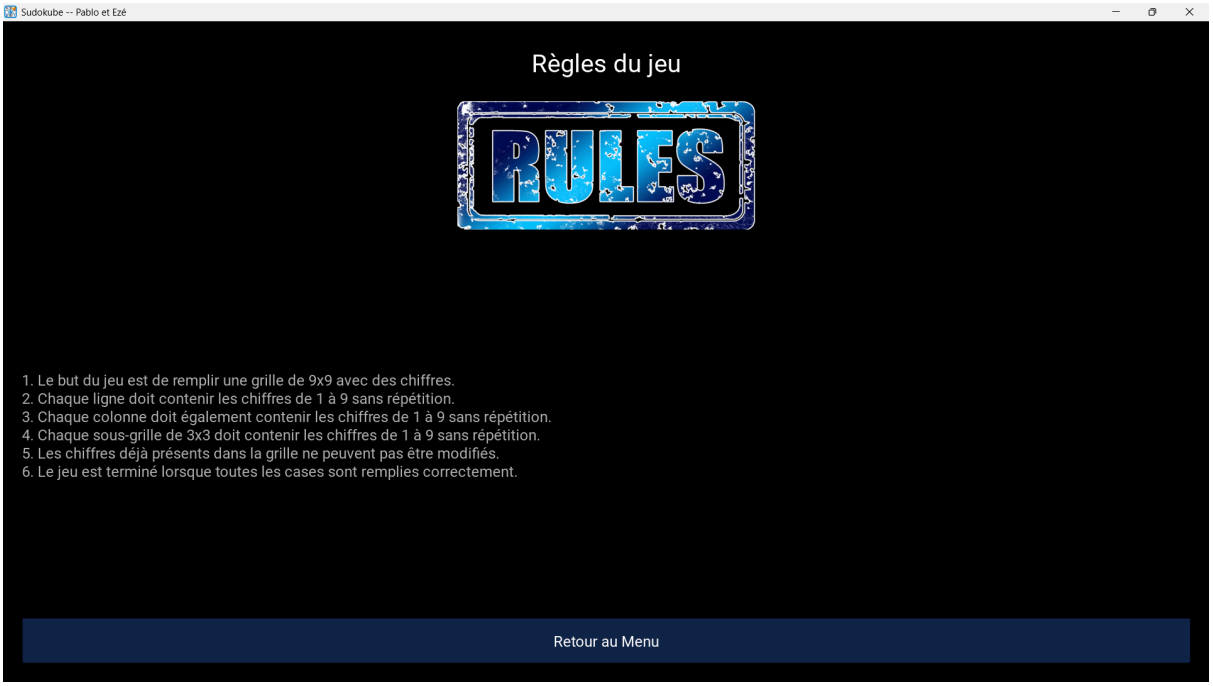
- Cette fonction vérifie si la grille de Sudoku est entièrement et correctement résolue. Elle repose sur une vérification systématique des lignes, des colonnes et des sous-grilles 3x3 pour s'assurer que chaque chiffre de 1 à 9 y figure sans répétition. Chaque étape implique de :
  - Parcourir la grille case par case.
  - Vérifier que chaque ligne contient les chiffres de 1 à 9 sans répétition.
  - Vérifier que chaque colonne contient les chiffres de 1 à 9 sans répétition.
  - Vérifier que chaque sous-grille 3x3 contient les chiffres de 1 à 9 sans répétition.
- **Atout :**
  - Permet de valider rapidement si une grille est correcte et complète, ce qui est essentiel pour déterminer la fin du jeu.
  - La vérification systématique assure que toutes les règles du Sudoku sont respectées.
- **Limitation :**
  - La complexité de la vérification augmente avec la taille de la grille, bien que ce soit généralement acceptable pour les grilles standard 9x9.
  - Peut être coûteux en termes de temps de calcul si utilisé fréquemment sur de grandes grilles ou dans des boucles de résolution très profondes.

VI. Captures d’écran

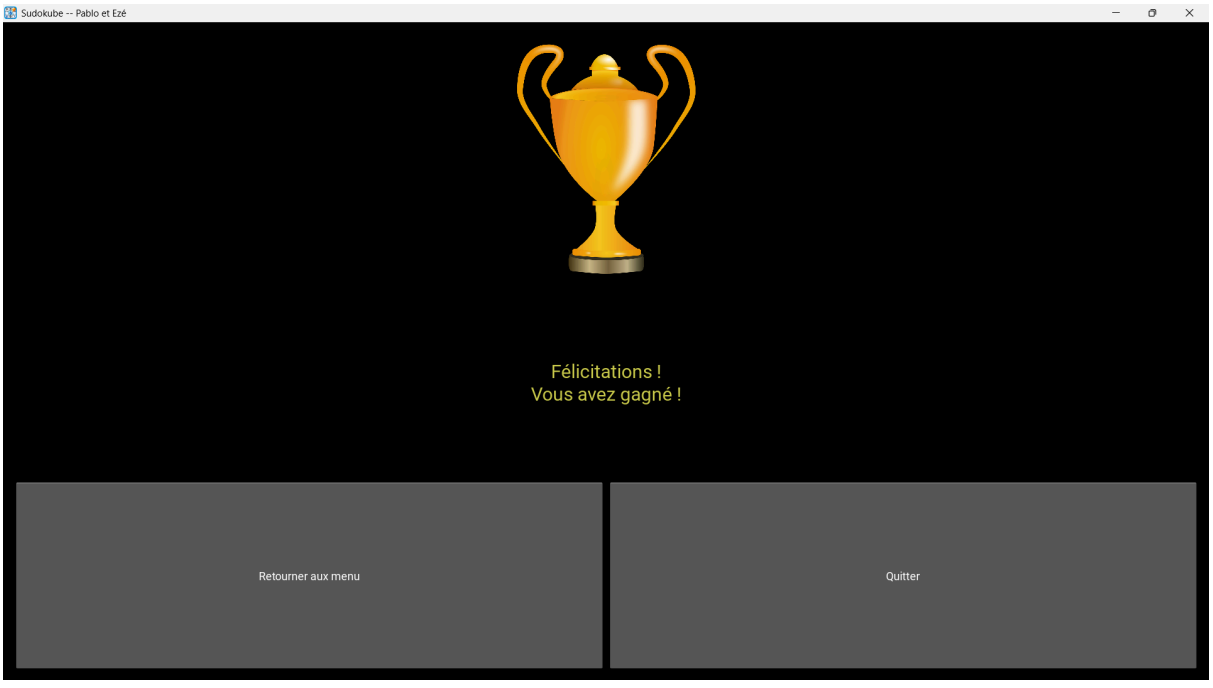
Écran d'accueil permettant de choisir la difficulté de jeu, ou d'accéder aux règles du jeu.



Écran représentant le plateau de jeu. Il représente ici un plateau “simple”. Les boutons permettent de poser un chiffre sur la case sélectionnée. il y a aussi un bouton pour retourner au menu.

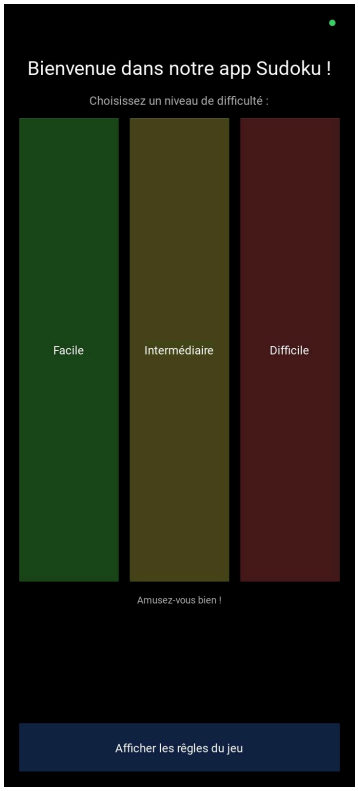


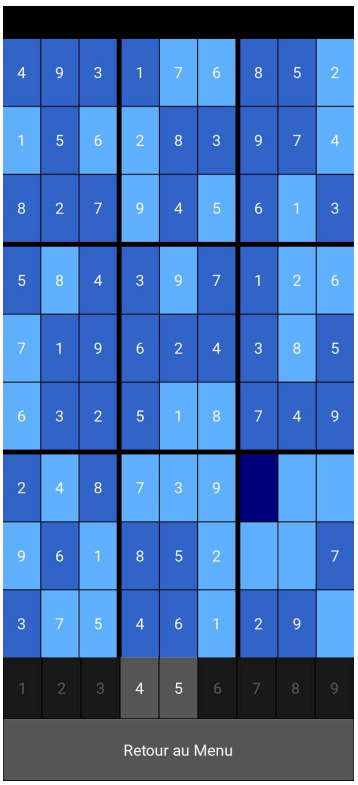
L'écran présent est l'écran règles du jeu. On peut y trouver les règles, et retourner au menu pour ensuite commencer le jeu.

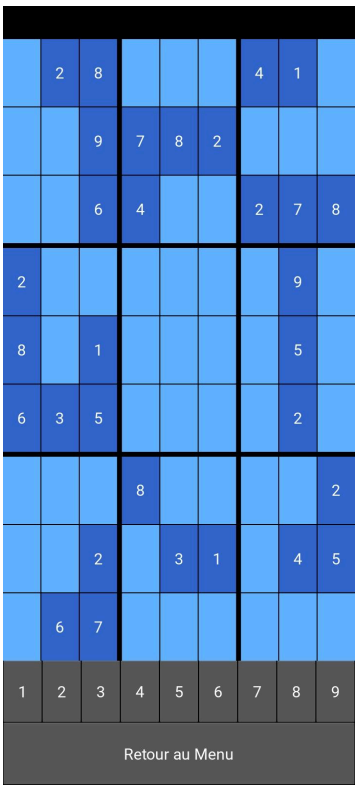


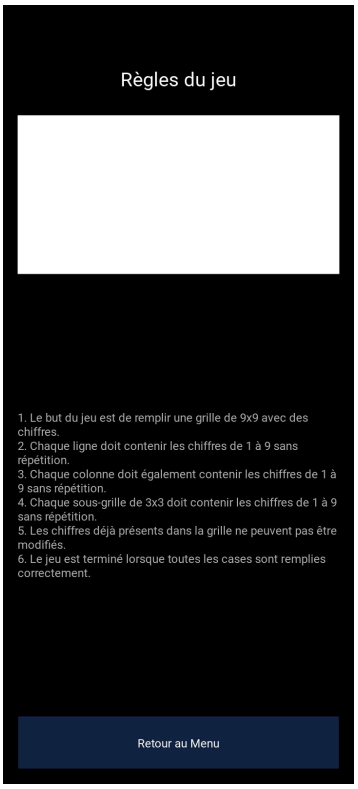
Ceci est l'écran de victoire, qui s'ouvre automatiquement lorsque le plateau est rempli correctement.

VII. Captures d'écran Android



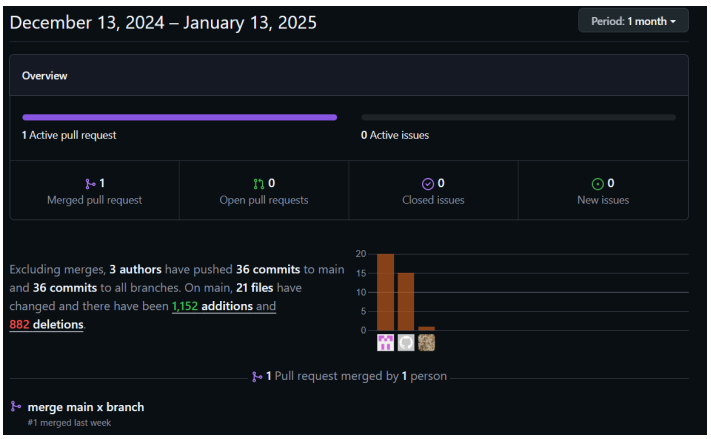






Les captures représentent la même chose, mais sur un écran de téléphone.

VIII. Activités Github



@Lasergamer-04 est le pseudonyme GitHub de Pablo Faivre, tandis que @eze-lab est celui d'Ezéchiel Renault. Les deux captures d'écran ci-dessous illustrent l'évolution des commits tout au long du projet.

Sur la capture de gauche, on observe que le projet a débuté début octobre, correspondant à la réception du sujet, et s'est poursuivi jusqu'à mi-janvier, date limite de rendu. Cependant, un détail attire l'attention : à partir de mi-novembre, @eze-lab semble ne plus avoir effectué de commits, ce qui n'est pas conforme à la réalité.

En examinant de plus près le graphique de droite, on remarque la présence des photos de profil de Lasergamer-04 (rose et blanche) et d'eze-lab (jaune clair et blanche), mais également d'une troisième (blanche avec un logo Github). Ce phénomène s'explique par une erreur de configuration de Git sur le nouveau PC d'Ezéchiel Renault (changé mi-novembre), où le pseudo "eze-lab" a été incorrectement remplacé par "Ezé".

Ainsi, les commits réalisés après mi-novembre apparaissent sous ce nom incorrect, ce qui empêche leur reconnaissance par GitHub et leur inclusion dans le premier graphique. Cela explique l'absence apparente d'activités de la part d'eze-lab, alors que les deux collaborateurs ont continué à travailler activement et à pousser du code régulièrement.

IX. Conclusion

Difficultés rencontrées

Lors du développement de l'application Sudokube, nous avons rencontré plusieurs défis techniques et conceptuels :

1. **Gestion des interactions entre l'interface utilisateur et la logique backend** : Assurer une communication fluide entre l'interface graphique créée avec Kivy et la logique de gestion du Sudoku dans le backend a nécessité une attention particulière. La synchronisation des états des cellules entre l'interface et la grille de données a été un point crucial.

2. **Optimisation des performances du solveur** : Le solveur de Sudoku utilise un algorithme de backtracking, dont la complexité peut rapidement augmenter. Nous avons dû optimiser certaines parties du code pour réduire le temps de résolution, surtout pour les grilles de difficulté élevée. Il arrive cependant que parfois, l'algorithme mette du temps à générer la grille, ce qui crée une latence dans l'application au moment du lancement d'une partie.

3. **Manipulation des widgets Kivy** : Travailler avec Kivy pour créer une interface utilisateur esthétique et réactive a été un apprentissage en soi. La gestion des événements, des transitions d'écran et des mises à jour dynamiques des widgets présente des défis intéressants.

4. **Gestion des états des cellules** : Assurer que les cellules verrouillées et modifiables soient bien différenciées et que les interactions utilisateur respectent les règles du Sudoku a nécessité des ajustements fréquents et des tests rigoureux.

5. **Gestion du Plateau de Jeu** : Le backend a généré le plateau en 9x9, mais le frontend en 3x3x3, ce qui a au début généré différents problèmes, mais nous avons résolu cela en ajoutant la fonction generate_81 dans le backend.

6. **Utilisation de Git** : la mauvaise connexion d'une des deux personnes sur Git a malencontreusement faussé les statistiques de commit et push fournies par Github sur les deux derniers mois.
7. **Réécriture complète du main.py**: Aux alentours de mi-novembre, donc au milieu du projet, il nous est clairement apparu que le fichier main.py (qui contient le frontend en kivy) n'avait pas été construit de façon optimale à cause de notre méconnaissance de Kivy. Nous avons donc décidé de réécrire ce fichier de 0 (en se basant tout de même sur la base écrite dans la 1ère version) afin d'optimiser le programme vers ce que nous voulions créer (en pouvant ajouter des transitions, en créant un lien plus solide et stable avec le backend.py etc…). Cette refonte nous a ainsi aussi permis de se rendre compte de quelques problèmes dans backend.py, qui a donc pu être optimisé et amélioré.

Extensions possibles

Pour améliorer et étendre les fonctionnalités de l'application Sudokube, plusieurs pistes peuvent être envisagées :

1. **Ajouter un mode chronométré** : Introduire un chronomètre pour mesurer le temps que met chaque utilisateur à résoudre une grille pourrait ajouter un élément compétitif et ludique.

2. **Sauvegarde et reprise des parties en cours** : Permettre aux utilisateurs de sauvegarder leur progression et de reprendre une partie ultérieurement améliorerait l'expérience utilisateur.

3. **Amélioration de l'interface utilisateur** : Intégrer des animations supplémentaires, des thèmes personnalisables et des sons peut rendre l'application plus attractive et immersive. Il pourrait aussi être possible, dans le futur, d'activer ou désactiver le blocage de certains boutons afin d'incrémenter la difficulté pour les joueurs avancés.

4. **Compatibilité multiplateforme** : Nous avons réussi à exécuter notre application sur Android (notamment à l'aide de Pydroid 3) et en tant qu'application Windows (notamment à l'aide de PyInstaller). Il pourrait donc être intéressant de publier notre application sous format IOS afin de toucher un public plus large. Ceci pourrait être rendu possible avec PyInstaller, mais il faut pour ce faire posséder un Macbook, ce que nous n'avons pas.

5. **Ajout de nouvelles fonctionnalités de jeu** : Par exemple, des modes de jeu innovants comme des défis quotidiens, des niveaux de difficulté progressifs et des tutoriels interactifs pour les débutants.

6. **Refaire le programme backend.py en C** : ceci permettrait d'améliorer la vitesse de génération du Sudoku, et donc diminuer la vitesse de chargement pour l'utilisateur

X. Bibliographie

1. **Documentation de Kivy**
   - Documentation Kivy

2. **Python**
   - Documentation officielle : Python.org

3. **Gestion de projets de jeux vidéo ou interfaces ludiques**
   - Schell, J. *The Art of Game Design: A Book of Lenses*. CRC Press, 2019.

4. **Vidéos pour apprendre le Kivy**
   - https://www.youtube.com/watch?v=_TMZMJN6Ozg&t=8s
   - https://www.youtube.com/watch?v=I8Imtec4ReQ&t=2s

5. **Articles Wikipédia pour connaître le sudoku**
   - https://fr.wikipedia.org/wiki/Sudoku
   - https://fr.wikipedia.org/wiki/Math%C3%A9matiques_du_sudoku

6. **Recours à l'intelligence artificielle:**
   - Exclusivement pour des explications complémentaires sur le fonctionnement de Kivy dans la phase initiale du projet.

XI. Répartition de Travail

Frontend	Pablo + Ezéchiel
Backend	Ezéchiel
Conversion en .exe	Ezéchiel
Testeur Mobile	Pablo
Présentation du projet	Ezéchiel
Choix effectués	Pablo + Ezechiel
11 fonctions essentielles	Pablo + Ezechiel
Focus sur les fonctions récursives	Pablo
Captures d'écran	Pablo + Ezechiel
Captures d'écran Android	Pablo
Activitées Github	Pablo + Ezechiel
Conclusion	Pablo + Ezechiel
Bibliographie	Pablo