université de BORDEAUX

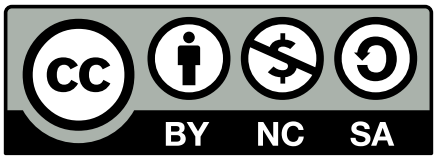
Projet de Programmation

(Comment ça marche ?)

Emmanuel Fleury
<emmanuel.fleury@u-bordeaux.fr>

LaBRI, Université de Bordeaux, France

27 mars 2026



Plan

Plan

Objectifs du cours

université de BORDEAUX

Les objectifs de cette UE sont :

- Pratiquer le **travail en équipe** ;
- Mise en œuvre des **concepts de développement** vus en cours ;
- Se familiariser avec les **outils de développement** ;
- S'attaquer à un **projet informatique de taille conséquente** ;
- Apprendre à **tenir les délais et respecter les contraintes** ;
- Apprendre à **faire des présentations techniques** ;
- Apprendre à **rédiger un rapport technique**.

Organisation des projets

université de BORDEAUX

- **20 groupes** constitués de **5 étudiants** (6 si nécessaire) ;
- **10 sujets** qui sont réalisés en **C**, **Java** ou **Python** ;
- **Chaque sujet** est choisi par **2 groupes** avec des **langages de programmation différents** ;
- **Un cours** de 2h chaque semaine **pour présenter les concepts nécessaires** ;
- **Le premier mois** est consacré au **rapport préliminaire** et à l'écriture de **l'architecture du projet et de ses besoins étendues** ;
- Puis, **chaque semaine** le groupe aura **30mn pour présenter les besoins achevés** à leurs chargés de TD ;
- Chaque besoin devra être **présenté avec son code, ses tests et sa documentation** pour validation auprès des chargés de TD ;
- Les chargés de TD pourront **valider les besoins** présentés ou **demander de retravailler** certaines parties.
- À la fin du projet, pensez à réserver du temps pour écrire le **rapport final** et préparer la **présentation finale** (1h).

Agenda

Le projet se déroule sur 13 (+1) semaines :

- **Semaine 0** (aujourd'hui) : Présentation de l'UE, finalisation des **groupes** et **choix des sujets** ;
- **Semaine 1** : **Présentation du projet aux chargés de TD**, **agenda** et **répartition des besoins dans le groupe** ;
- **Semaine 2** : Présentation des **slides de l'oral** et première version du **Rapport préliminaire** ;
- **Semaine 3** : **Oral** : 20mn, **Rapport préliminaire** : remis avant 21h.
- **Semaines 4–12** (8 semaines, Livraison besoins) :
  Faire le développement, livrer les **besoins** et démontrer qu'ils sont corrects ;
- **Fin Semaine 12** : **Rapport final** + **Code** ;
- **Semaine 13** : **Oral** : 1h.
  Présentation du projet complet, algorithmes, structures de données, architecture finale, besoins achevés, performances, ...

Plan

La présentation du projet (20mn) :

- **Explication** détaillée du sujet ;
  (historique du jeu, règles, complexité du jeu, existant)
- **Algorithmes** et **structures de données** ;
  (structures de données et algorithmes qui seront employés)
- **Architecture** visée pour le projet ;
  (découpage en modules, dépendances, principales interfaces, *etc.*)

Rapport préliminaire (final, semaine 3)

université de BORDEAUX

Le rapport (en LaTeX, 10 pages max) :

- **Explication détaillée** du sujet ;
  (historique du jeu, règles, complexité du jeu, existant)
- **Algorithmes** et **structures de données** ;
  (structures de données et algorithmes qui seront employés)
- **Architecture** visée pour le projet ;
  (découpage en modules, dépendances, principales interfaces, *etc.*)
- **Spécifications étendues**.
  (**une fonctionnalité** que vous comptez réaliser (**détaillez la jusqu'au bout**))
- **Références bibliographiques**.
  (au moins **5 citations** de sources diverses : articles, livres, site web, *etc.*)

Spécifications étendues

Les **spécifications étendues** sont un raffinement des spécifications initiales, elles listent les sous-besoins et les actions à mettre en place pour réaliser le besoin visé.

En partant d'une fonctionnalité de la spécification :

- Identifiez les **prérequis** avec d'autres besoins ainsi que des **sous-besoins** (besoins nécessaires pour réaliser le besoin visé) qui raffinent le besoin initial ;
- Décrivez les **actions** à mettre en place pour réaliser chaque sous-besoin ;
- Nommez quelques **fonctions** qui permettront de réaliser chaque sous-besoin, **donnez leur signature** et **quelques tests unitaires** (des exemples d'utilisation, quoi en entrée ? quoi en sortie ?) sur ces fonctions ;
- Expliquez comment ces fonctions s'intègrent dans l'architecture du project ;
- Enfin, expliquez comment **valider le besoin** une fois tous les sous-besoins réalisés avec des tests ou bien des scénarios précis.

Spécifications étendues (Exemples 1/2)

université de BORDEAUX

F4. Système cible

Le programme doit fonctionner sur des systèmes d'exploitation GNU/Linux.

Exemples de points à intégrer dans la réponse :

- Utiliser une Ubuntu 20.04 LTS pour le développement ;
- Tester sur une Debian 10.0 et une Fedora (compilation et tests automatiques sur une image Docker lancée sur le Gitlab).

F17. Version

L'option '`-V`', '`--version`' affiche la version puis quitte.

Exemples de points à intégrer dans la réponse :

- Fonctionnalité gérée par `getopt_long()` ;
- Tester avec '`-V`' et '`--version`' et vérifier que la version est bien affichée et que le programme se termine avec un code `EXIT_SUCCESS`.

Spécifications étendues (Exemples 2/2)

université de BORDEAUX

F21. Conseil aux joueurs

Sur demande du joueur courant, le programme pourra suggérer un coup à jouer.

Exemples de points à intégrer dans la réponse :

- Prérequis : F9. (Interface utilisateur), F12. (Joueur artificiel) et F37. (Heuristiques) ;
- Requiert une fonction `player_hint()` qui retourne un coup valide (basée sur la fonction `search_tree()` qui va rechercher le meilleur coup possible en fonction d'une heuristique donnée qui sera implémentée par F37.) ;
- Signature de la fonction :

```
move_t player_hint(board_t *board, player_t *player);
```

- Tester que la fonction retourne bien un coup valide (vérifier le coup avec un arbitre) pour un plateau de jeu et un joueur donné .

Plan

Erreurs sur l'Existant

- **Pas de description de l'existant** ;
- **Description incomplète** (manque de détails, manque de sources, pas d'historique du jeu, pas d'exploration de la littérature scientifique (ou grand public) à ce sujet, ...) ;
- **Description non pertinente** (description de l'existant non en rapport avec le projet) ;
- **Description non citée** (pas de référence à la bibliographie ou même pas de référence à des sources externes) ;
- **Description non mise en forme** (liste à points, pas de texte lisible, des phrases jetées ça et là, ...).

Erreurs Existant : Exemple 1

université de BORDEAUX

1. Contexte et existant du projet

1.1. Contexte du projet

Le projet porte sur la création d'une version numérique du jeu de stratégie abstrait **Agon**. Ce jeu, historiquement joué sur un plateau hexagonal, nécessite une réflexion stratégique approfondie et une parfaite maîtrise des règles. L'objectif principal est de concevoir une solution logicielle qui intègre les fonctionnalités suivantes :

- **Une interface graphique fluide et intuitive** : destinée à rendre l'expérience utilisateur immersive et accessible, avec une navigation claire et des animations fluides.
- **Simulation fidèle des règles officielles du jeu** : respect strict des mécanismes de déplacement, de capture, et des conditions de victoire.
- **Intelligence artificielle avancée** : l'intégration d'une IA performante capable de proposer des défis variés, adaptés aux niveaux de compétence des joueurs (débutant, intermédiaire, expert).
- **Mécanismes avancés de gestion du plateau** : utilisation des *Bitboards*, une structure de données efficace et compacte pour représenter le plateau de jeu et ses états.

Problèmes

- Aucune référence à la bibliographie ;
- Non-pertinent car reprend des éléments du cahier des charges et pas du contexte du jeu, ni de l'existant ;
- Plusieurs termes sont trop flous (« *fluide et intuitive* », « *IA avancée* » ? ? ?).

Erreurs Existant : Exemple 2

université de BORDEAUX

1 Contexte du projet

1.1 Contexte général

Le **Jeu des Amazones** est un jeu de stratégie combinatoire abstrait à deux joueurs, introduit en 1988 par l'argentin **Walter Zamkauskas**. Il se joue sur un plateau carré, avec des pièces appelées "amazones". Chaque joueur en contrôle un certain nombre, généralement 4, et a pour objectif de restreindre les mouvements de l'adversaire tout en maximisant ses propres possibilités de déplacement.

Le **Jeu des Amazones** est réputé pour sa richesse combinatoire et sa complexité algorithmique. Il s'agit d'un jeu qualifié de "stratégie combinatoire abstrait", désignant les jeux où deux joueurs jouant à tour de rôle s'opposent, avec une vision des pièces identique pour les deux adversaires et sans hasard impliqué. C'est un jeu semblable au go ou aux échecs.

La difficulté computationnelle provient de la croissance exponentielle du nombre de positions possibles au fur et à mesure que le jeu progresse, en raison des nombreux mouvements et blocages générés par les flèches. En particulier, déterminer qui va gagner une partie précise est NP-difficile, et déterminer qui va gagner une partie de façon générale est PSPACE-complet.[Wik24]

1.2 Contexte du projet

L'objectif du projet est de programmer le **Jeu des Amazones** en langage Python, du **8 janvier** au **18 avril** 2025, en groupe de 5 étudiants issus de spécialités différentes en Master en informatique.
Nous avons opté pour le langage Python plutôt que le C ou le Java, tous trois proposés, pour plusieurs raisons :

- Le langage est peu étudié lors de projets à l'Université de Bordeaux, ce qui constitue une bonne occasion de s'exercer
- Il offre de nombreuses bibliothèques simples et optimisées
- Il est plus flexible que le C, notamment au niveau gestion de mémoire, facilitant ainsi l'implémentation
- Il est mieux documenté que le Java.De nombreux articles scientifiques sont rédigés en Python, facilitant ainsi la compréhension et l'adaptation au projet

Correct

- Texte rédigé ;
- Historique du jeu ;
- Référence à la bibliographie ;
- Description pertinente de l'existant (complexité, stratégies, . . .).

Incorrect

- Le contexte du projet ne doit pas faire partie du rapport ;
- Les choix imposés par le cahier des charges original ne sont pas pertinents ici.
- Vos arguments ne sont pas justifiés et sont largement sujets à caution (langage peu étudié, plus flexible que le C (? ! ?), . . .).

Erreurs sur les Règles

université de BORDEAUX

- **Pas de description des règles du jeu** ;
- **Description incomplète** (manque de détails, pas assez d'explication, tous les cas ne sont pas traités, ...) ;
- **Manque d'exemples** (pas d'exemple de partie ou de coup, ...) ;
- **Début et fin de partie non décrits** (comment on commence, comment on gagne, ...) ;
- **Description non pertinente** (mélange avec l'algorithmique ou les structures de données, ...) ;

À la fin de la lecture des règles, on doit pouvoir jouer une partie sans ambiguïté !

Erreurs Règles : Exemple 1

université de BORDEAUX

2 Règles du Jeu

1. Le Gomoku se joue sur un plateau dont les dimensions peuvent être variables.
2. Le jeu est joué par deux joueurs : Noir et Blanc.
3. Noir commence en plaçant une pierre n'importe où sur le plateau.
4. Le gagnant est le premier joueur à aligner 5 pierres en ligne.
5. Les pierres doivent être placées de manière adjacentes à des pierres existantes.

Problèmes

- Trop imprécis (taille variable comment ?) ;
- Incomplet (quelles variantes existent ?) ;
- Aucun exemple de partie, ni schéma du plateau ;
- La notation d'un coup n'est pas abordée.
- Quelles pourraient être les stratégies gagnantes ?

Erreurs Règles : Exemple 2

2.1 Règles du jeu

Le jeu des Amazones est un jeu de plateau pour deux joueurs, où chaque joueur contrôle une série d'amazones qui se déplacent sur un damier et doivent bloquer les autres en lançant des flèches sur le plateau.

Situation initiale : un plateau de 10x10, sur lequel figure 8 amazones (4 noires et 4 blanches).

Objectif du jeu : L'objectif du jeu est de bloquer toutes les amazones de l'adversaire, en les empechant de faire un mouvement légal.

Règles de déplacement : Les amazones se déplacent sur le plateau de manière similaire aux reines dans les échecs :

1. Mouvement en ligne droite
2. Mouvement en diagonale

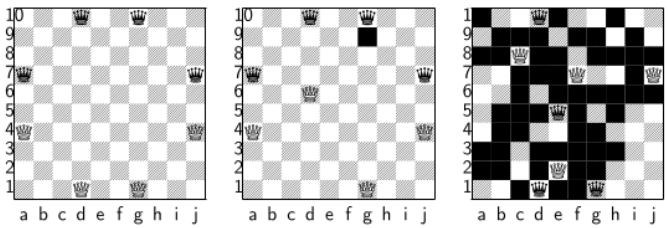
Elles ne peuvent pas passer par dessus une autre amazone ou une flèche.

Déroulement de la partie : Chaque action se divise en 2 parties :

1. Déplacer une amazone (♕) de même manière qu'une reine dans les échecs (verticalement, horizontalement et orthogonalement), il ne peut pas traverser une autre amazone ou une flèche.
2. Ensuite, le joueur peut tirer une flèche depuis l'amazone qu'il vient de déplacer, cette flèche a les mêmes propriétés que le déplacement d'une amazone.

Fin de partie : La partie prend fin lorsqu'un des joueurs n'a plus de coups possibles.

Voici un exemple de comment se déroule une partie d'amazones :



Correct

- Ce qui est présenté est précis et clair ;
- Les principaux éléments des règles sont donnés et permettent de jouer une partie ;
- Quelques exemples de parties sont donnés (mais probablement pas assez).

Incorrect

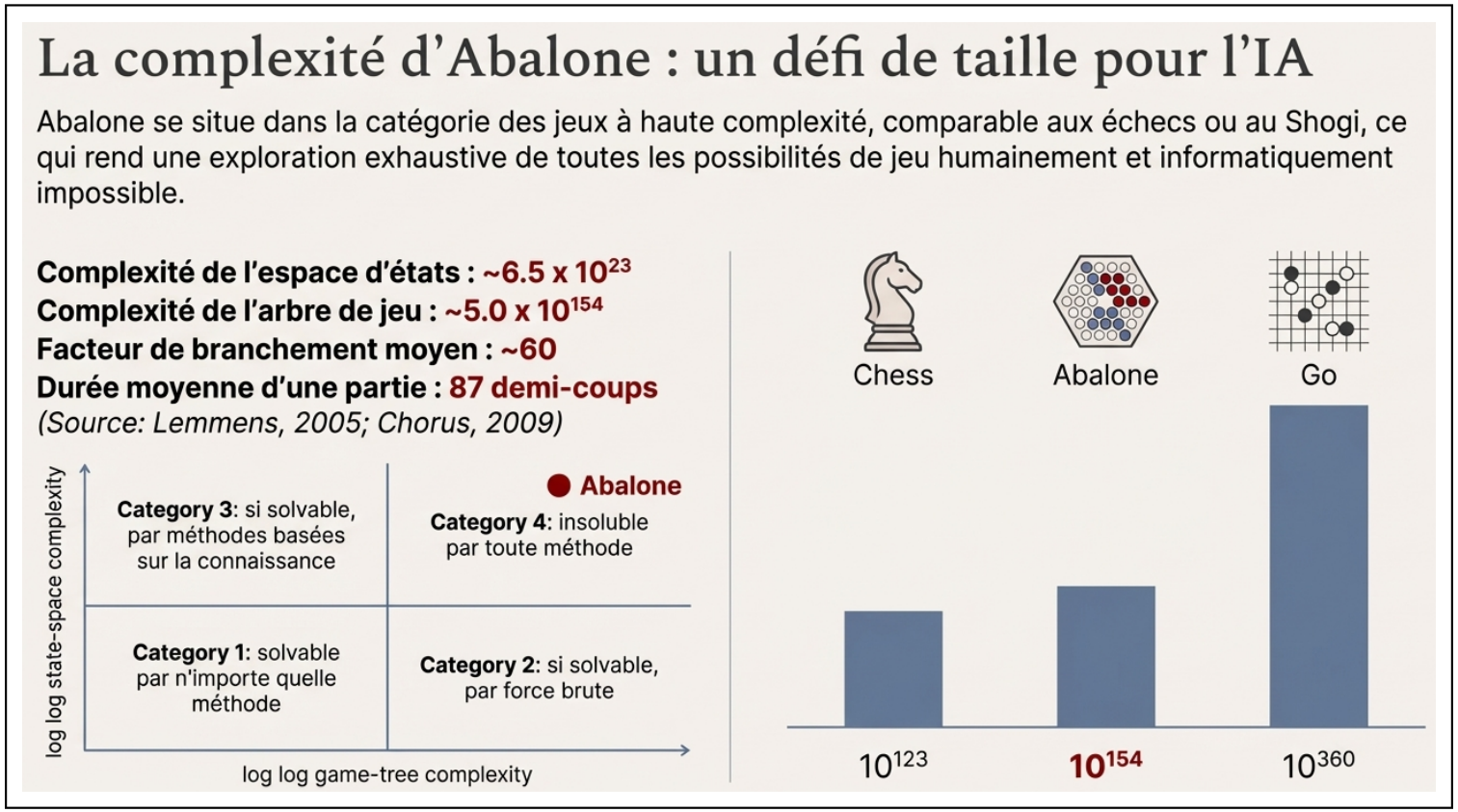
- Typographie approximative pour les indentations des différentes sections ;
- Ne pas utiliser le souligné pour marquer des sous-sections, utiliser `\paragraph{}` ou autre, à la place ;
- La position initiale du jeu est présente mais n'est pas décrite comme telle.
- Les règles sont incomplètes (est-ce qu'un match nul est possible ? Et si non, pourquoi ? Y-a-t-il des variantes du jeu ?) ;
- Quelques conseils stratégiques pourraient être donnés.

Complexité du jeu

Le but de cette section est de chiffrer l'espace d'état du jeu et la complexité inhérente au jeu. Il ne faut pas la confondre avec la complexité des algorithmes que vous allez employer pour jouer au jeu.

- **Pas de description de la complexité du jeu** ;
- **Description incomplète** (manque de détails, pas assez d'explication, . . .) ;
- **Manque de référence** (pas de référence à la bibliographie ou même pas de référence à des sources externes) ;
- **Manque d'exemple** (pas d'exemple chiffré, pas d'illustration, pas assez pédagogique, . . .) ;
- **Description non pertinente** (confusion avec les règles ou l'algorithmique).

Erreurs Complexité : Exemple 1



Correct

- Présentation claire de la complexité du jeu avec mise en perspective par rapport à d'autres jeux connus ;
- Quelques exemples chiffrés sont donnés ;
- Références à la bibliographie.

Incorrect

- Implicitement vous ne parlez que de méthodes de recherche exhaustives mais sans jamais l'expliciter. Qu'en est-il des autres méthodes ?
- Le tableau de gauche n'est pas très clair et n'apporte pas grand chose à l'explication. Le détail des calculs pour obtenir les chiffres donnés serait plus pertinent.

- **Pas d'explication des algorithmes & structures de données** ;
- **Description incomplète** (manque de détails, pas assez d'explication, ...) ;
- **Manque de référence** (pas de référence à la bibliographie ou même pas de référence à des sources externes) ;
- **Manque d'exemple** (pas d'exemple d'algorithme ou de structure de données, ...) ;
- **Description non pertinente** (mélange avec les règles ou l'architecture, ...) ;

Erreurs Algos et Données : Exemple 1

université de BORDEAUX

2.2 Structure de données

Pour ce projet nous utiliseron les structure de données suivante :

— des bitboards, qui nous permeteron de représenter l'état du plateau de jeux, tout au long d'une partie d'amazones. On implementera les bitboards en faisant une liste de liste ce qui nous permettra d'accéder rapidement à une case donné du plateau. Nous aurons 5 bitboards qui nous permeterons de représenter l'état des amazones, des pions blanc, des pions noir, des flèches et enfin de toutes les pièces.

— une liste, qui nous permettra d'enregistrer et de naviguer dans l'historique.

— la classe StateGame qui nous permetra de stocker si les différents mode de jeux sont activés ou non, sous formes de bool.

— des fichiers, pour pouvoir sauvegarder un plateau grace au fonctions comme

Problèmes

- Il manque des exemples concrets ;
- Orthographe et grammaire à revoir ;
- On voit tout de suite que les structures de données ne sont pas comprises ;
- Les algorithmes liés aux structures de données ne sont pas décrits ;
- Il n'est pas pertinent de faire référence à l'architecture ici (`StateGame`). Mais des exemples de code peuvent être donnés pour illustrer les explications.

Erreurs Algos et Données : Exemple 2

université de BORDEAUX

> Le groupe a ensuite réfléchi à d'autres structures de données. Rapidement, nous avons noté les listes chaînées, les tableaux, ainsi que les arbres. Les listes (doublement) chaînées peuvent être une solution dans certains cas. En effet, l'insertion et la suppression sont des opérations réalisables en O(1). L'historique est alors plus flexible et les modifications sont assez efficaces. Comme la pile, pour revenir à un certain coup de la partie, nous sommes en O(n). En revanche, il y a certains cas où cette approche n'est pas très optimale. Par exemple, pour les parties (très) longues. En effet, la surcharge en mémoire (due aux références) et le coût en temps pour parcourir la liste peuvent devenir problématiques. Aussi, s'il y a régulièrement un accès à différents coups (précédents comme futurs), cela est moins rapide que des structures indexées. Finalement, les listes chaînées sont très similaires aux piles sur beaucoup d'aspects (complexité notamment). Cependant, nous pensons qu'il s'agit d'une solution moins "élégante" et plus "coûteuse" en dette technique et maintenabilité.
> Nous avons également porté notre attention sur les tableaux, spécialement efficaces pour l'accès à des éléments (O(1)). C'est aussi plutôt simple et rapide à mettre en oeuvre. Toutefois, on doit avoir une taille fixe pour cela. Ou alors, il faudrait effectuer des réallocations, qui peuvent devenir coûteuses sur le long terme. De même, pour réécrire l'historique, cela peut prendre O(n) en temps, sans compter que l'espace mémoire a pu être alloué "pour rien" au sens où l'on aurait au bout du compte pas eu besoin d'autant de mémoire.
> Finalement, les arbres auraient aussi pu être intéressants dans le cas où il y aurait des variations (branchement), car ils offrent une complexité assez valable en temps. En effet, lorsque l'on déroule les branches ou variations de coups, l'arbre est en fait la structure de données naturelle dont on va se servir. C'est d'ailleurs ce que l'IA utilisera dans le cadre de la recherche du meilleur coup.
> Remarquons que sans les branchements (dans notre cas donc), utiliser un arbre revient à utiliser une liste chaînée. Regardons un exemple pour ensuite discuter les avantages et les inconvénients

Problèmes

- Utilisez la mise en forme du texte pour clarifier (paragraphes plus petits , . . .) ;
- Ne parlez pas de vos chemins de réflexion, parlez de vos conclusions et justifiez les ;
- Parler de complexité est pertinent mais vous n'avez pas identifié les opérations les plus fréquentes sur ces structures de données ;
- N'hésitez pas à donner votre implémentation sous forme de code et à justifier vos choix.

Erreurs Algos et Données : Exemple 3

(b) On applique la réduction en utilisant l'algorithme BPR 4. Le principe est de réduire les triplets d'hexagones en un seul qui contient la majorité des couleurs. L'algorithme est le suivant. Pour le jeu de Hex $D_y = 2 \times dim - 1$ et $C = dim + 1$. a, b et c représentent un triplet de pairs de bits et l'opération $a\&(b|c)|b\&c$ les réduit à une paire 3. L'algorithme renvoie 0 si personne a gagné, 1 si le joueur vertical a gagné et 2 si le joueur horizontal a gagné.

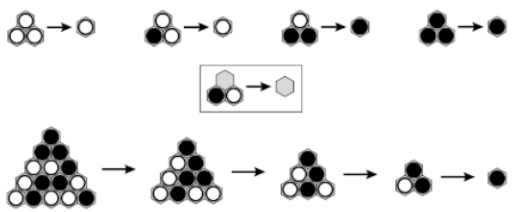


FIGURE 3 – Déroulement de BPR sur un jeu de Y. Figure de [BT12]

Listing 1 Bitwise-parallel reduction.

```
int BPR()
{
    for (int pass = 0; pass < D_y − 1; pass++)
        for (int col = 0; col < min(C, D_y-pass)-1; col++)
        {
            final int a = bits[col];
            final int b = (a >> 2);
            final int c = bits[col + 1];
            bits[col] = (a & (b | c)) | (b & c);
        }
    return bits[0] & 0x3;
}
```

FIGURE 4 – L'algorithme BPR. Figure de [BT12]

Correct

- L'algorithme est décrit en pseudo-code et référencé dans le texte ;
- Un exemple non-trivial est donné avec quelques pas d'exécution ;
- On cite l'article original où est présenté l'algorithme.

Incorrect

- Ne vous appuyez pas seulement sur des sources externes, donnez votre propre compréhension des algorithmes et des structures de données ;
- Le déroulement de l'algorithme sur l'exemple pourrait être plus détaillé ;
- Les structures de données ne sont pas données sous forme de code.

Erreurs sur l'Architecture

Le but de la section sur l'architecture est de permettre au lecteur de comprendre comment votre programme est organisé, comment les différentes parties interagissent entre elles et comment les algorithmes et structures de données sont utilisés dans ce contexte.

- **Pas de description de l'architecture** (partie absente ou hors sujet) ;
- **Description incomplète** (manque de détails, pas assez d'explication, ...) ;
- **Seulement des schémas** (pas de texte pour expliquer les schémas, ...) ;
- **Manque de justifications** (manque un texte pour justifier les choix d'architecture et donner les arguments positifs/négatifs liés à ces choix, ...) ;
- **Architecture pas implémentable** (trop complexe, trop naïf, pas de lien avec les algorithmes & structures de données, ...) ;
- **Description non pertinente** (confusion avec règles ou algorithmes, ...).

Erreurs Architecture : Exemple 1

5 Architecture du Projet

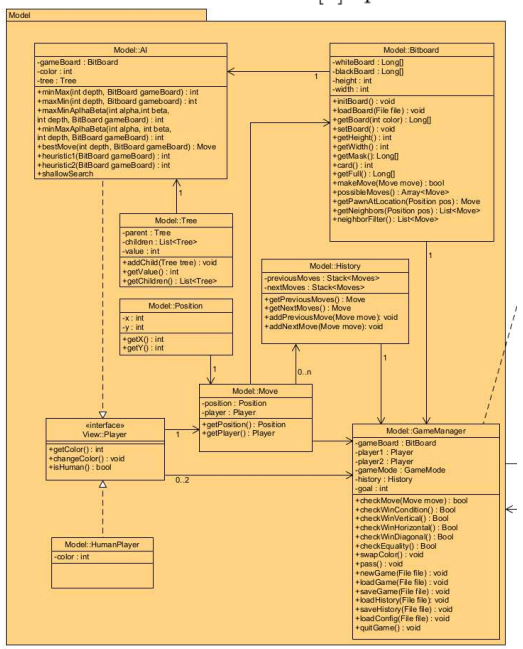
Le projet va suivre une architecture Model-View-Controller[3] qui est définie ci-dessous :

5.1 Package Model

Le package Model comprend :

- 'Position'
- 'Tree'
- 'GameManager'
- Une interface 'Player'
- 'AI'
- 'Move'
- 'History'
- 'Bitboard'

5.2 Package Controller



Correct

- Utilisation d'un diagramme UML ;
- Les composants sont clairement identifiés.

Incorrect

- Aucune justification des choix d'architecture (il existe plusieurs variantes du MVC, laquelle avez-vous choisi et pourquoi ?) ;
- Aucune explication des rôles des différents composants ;
- Les interfaces ne sont pas clairement détaillées en dehors de l'UML ;
- Le diagramme de classes contient aussi trop de méthodes sans réel intérêt (en particulier le getters et les setters). Utiliser le mode « *sketch* » pour être plus synthétique.
- Pas de lien avec les algorithmes et structures de données.

Erreurs Architecture : Exemple 2

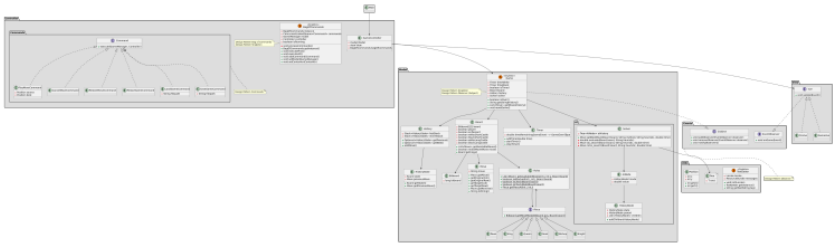
université de BORDEAUX

5 Architecture visée pour le projet

Nous avons décidé de présenter le projet sous une forme d'architecture MVC personnalisée. Cette architecture est particulièrement pratique pour nous, développeurs, aussi bien en termes de maintenance que pour l'amélioration future du projet.

Les différents modules de notre projet sont les suivants : **Model**, **Controller**, **Vue**, **Events** et **Utils**.

Voici l'architecture UML du projet :



5.0.1 Description des Modules

- **Model** : Gère la logique métier du jeu, incluant l'état de la partie, les règles d'échecs et les fonctionnalités comme l'historique et le timer.
- **Controller** : Sert d'intermédiaire entre la Vue et le Model. Il gère les événements utilisateurs et orchestre les interactions entre les différentes couches.
- **Vue** : Responsable de l'affichage et des interactions avec l'utilisateur, qu'il s'agisse d'une interface CLI ou graphique.
- **Events** : Implémente un système d'observateurs pour gérer les notifications et la communication asynchrone entre les modules.
- **Utils** : Contient des outils génériques comme le gestionnaire de textes internationalisés (`TextGetter`) et les algorithmes d'intelligence artificielle pour le jeu.

5.0.2 Design Patterns Utilisés

Les différents design patterns utilisés dans ce projet sont les suivants :

- **Singleton** : Utilisé pour le module `Utils/TextGetter` afin de centraliser l'accès aux ressources de texte. Il sera également utilisé pour la classe `Game` pour s'assurer qu'une seule instance de game est présente. La classe `BagOfCommands` utilise ce design pattern afin que la gestion des commandes ne soit gérée que par un seul bag.
- **MVC** : L'architecture principale du projet, séparant clairement la logique métier, l'affichage et la gestion des événements. C'est un MVC modifié, grace au design pattern observer qui va permettre d'envoyer des messages du Model vers la Vue. Cela rend le code plus simple et digeste.
- **Observer** : Permet de notifier la Vue des modifications dans le Model.
- **State** : Utilisé pour gérer l'état de la partie, comme le statut du jeu (en cours, échec et mat, pat, etc.).

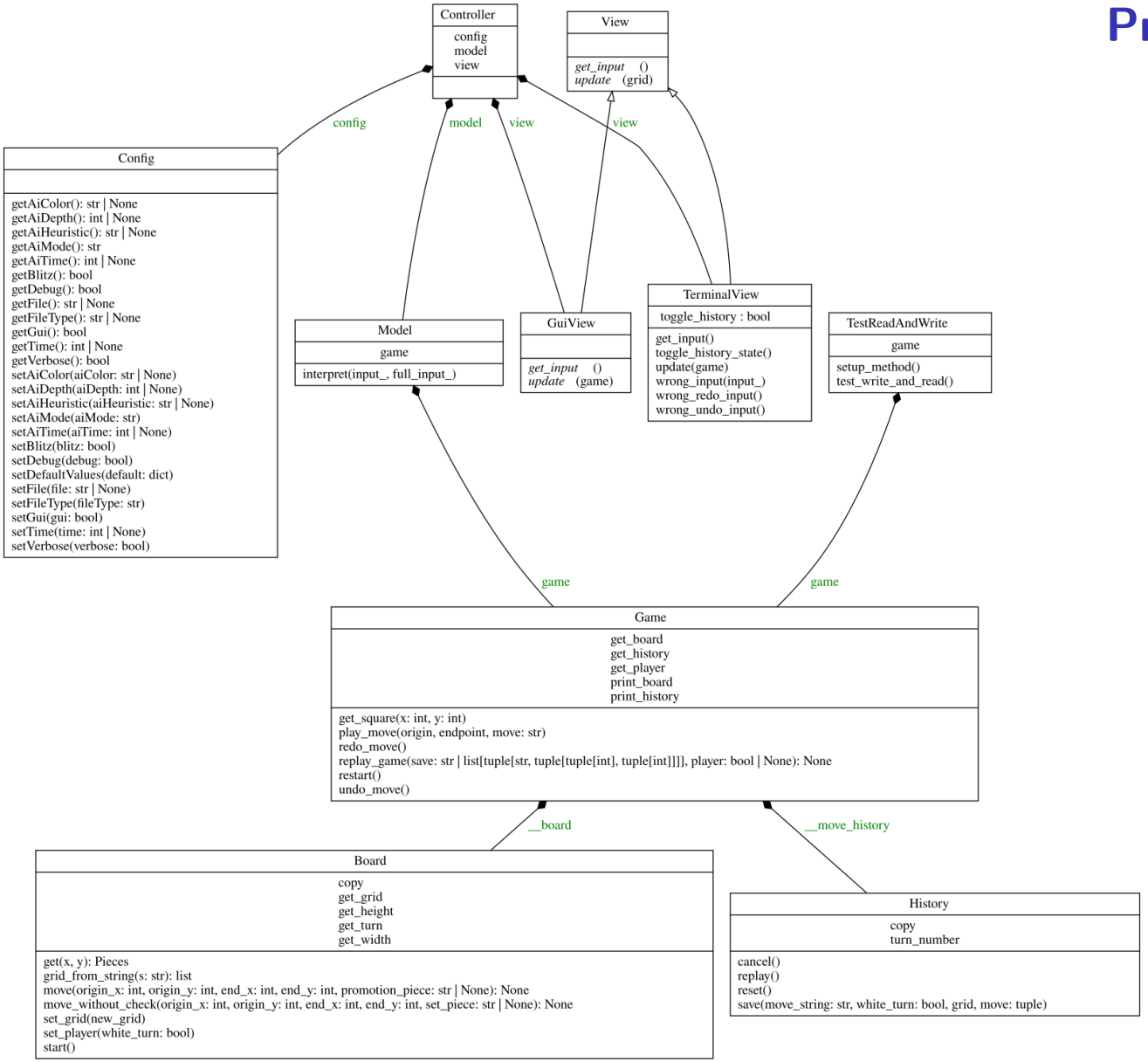
Correct

- Utilisation d'un diagramme UML ;
- Les composants sont clairement identifiés ;
- Une courte description des composants est donnée.

Incorrect

- Schéma trop petit pour être lisible ;
- Aucune justification des choix d'architecture (il existe plusieurs variantes du MVC, laquelle avez-vous choisi et pourquoi ?) ;
- Les interfaces ne sont pas clairement détaillées en dehors de l'UML ;
- Pas de lien avec les algorithmes et structures de données ;
- Le Pattern MVC est architectural et ne devrait pas être mélangé avec les autres car il n'est pas de même nature.

Erreurs Architecture : MVC (Python)



Problèmes

- Inutile d'avoir des classes `Model`, `View` et `Controller`. Ce sont des rôles du Design Pattern, pas des classes.
- Ici, le modèle contient presque tout le code et le contrôleur n'est qu'un simple intermédiaire.
- On peut suggérer que la classe `Game` est le contrôleur, la classe `Board` est le modèle et on peut renommer la classe `View` en `UserInterface`.
- On préférera souvent utiliser **une architecture N-tiers** (en couches) qui est souvent plus simple à implementer.
- La classe `Config` peut simplement être remplacé par un dictionnaire.
- La classe `TextReadAndWrite` indique que le code des tests n'a pas été suffisamment séparé du code de l'application. Elle ne devrait pas apparaître ici.

Erreurs sur les Besoins Étendus

université de BORDEAUX

- **Pas de description des besoins étendus** ;
- **Description incomplète** (manque certaines descriptions pour le besoin : date de rendu, développeur responsable, dépendances, explication du besoin, découpage en sous-besoins, stratégie de validation du besoin, dépendance inversée, ...) ;
- **Description non pertinente ou fausse** (besoin non en rapport avec le projet, mauvaise classification (fonctionnel/non-fonctionnel), ...) ;
- **Description non mise en forme** (liste à points, pas de texte lisible, des phrases jetées ça et là, ...) ;
- **Manque de référence à la spécification initiale** (ne renommez pas les besoins, ne les changez pas sans justification, ...).

Erreurs Besoins Étendus : Exemple 1

université de BORDEAUX

3 La liste des besoins visés

	Besoin	Dépendance
B01	Mettre en place une AI	B07, B13, B10
B02	Lancer le programme grâce à la commande scrabble	B12
B03	Pouvoir jouer en plusieurs langues	B07
B04	Avoir une interface graphique GUI	B02, B13
B05	Avoir une interface de jeux en ligne de commande CLI	B02, B13
B06	Pouvoir accéder à un historique des coups joués commentés	B13
B07	Déterminer si un mot est dans le dictionnaire	
B08	L'application ne doit jamais crasher de façon non gérée	B11
B09	Pouvoir jouer en multijoueur local	B04, B13
B10	Avoir un système de score	B13
B11	Mettre en place des tests et les automatisés	
B12	Respecter un fichier de configuration au lancement de l'appli	
B13	Gérer le déroulement d'une partie	B07, B08
B14	Pouvoir sauvegarder et charger une partie	B13
B15	Support de différents modes de jeux	B13

— **B01 : Mettre en place une AI**
Il faut que le joueur puisse jouer contre l'IA et que l'IA puissent trouver les

Problème

- Reprenez les besoins tels qu'ils sont dans le cahier des charges. Ils ont été réfléchis et ne sont pas là pour être modifiés à la légère.

Erreurs Besoins Étendus : Exemple 2

université de BORDEAUX

F11. Gestion des options

Les options en ligne de commande seront gérées par le module *argparse*.

— Besoin non fonctionnel.

Étapes à produire :

— Définir les différentes options en ligne de commande nécessaires pour le programme.
— Configurer `argparse` pour parser et gérer ces options.
— Implémenter des validations et des messages d'aide pour chaque option.
— Intégrer les options dans le flux de contrôle principal du programme.
— Documenter l'utilisation des options en ligne de commande dans le fichier `README.md`.

Prototypes de fonction :

— `def parse_arguments():`
— `def validate_options(args):`
— `def handle_options(args):`

Problèmes potentiels et décisions :

— **Problème :** Options en ligne de commande ambiguës ou mal définies pouvant entraîner des comportements inattendus.
— **Décision :** Définir clairement les options avec des descriptions précises et des validations rigoureuses pour éviter les ambiguïtés.
— **Problème :** Difficulté à gérer un grand nombre d'options, rendant l'interface utilisateur en ligne de commande complexe.
— **Décision :** Organiser les options de manière logique, utiliser des sous-commandes si nécessaire, et fournir une aide détaillée pour chaque option.
— **Problème :** Manque de rétrocompatibilité lors de l'ajout de nouvelles options, pouvant casser des scripts existants.
— **Décision :** Introduire de nouvelles options de manière rétrocompatible et déprécier progressivement les anciennes options si nécessaire.
— **Problème :** Documentation insuffisante sur l'utilisation des options, rendant difficile pour les utilisateurs de comprendre les fonctionnalités disponibles.
— **Décision :** Rédiger une documentation exhaustive dans le fichier `README.md`, incluant des exemples d'utilisation et des descriptions détaillées de chaque option.
— **Problème :** Gestion des erreurs liée aux options en ligne de commande pouvant générer des messages d'erreur peu informatifs.
— **Décision :** Implémenter des messages d'erreur clairs et instructifs, indiquant comment corriger les erreurs de saisie des options.

Correct

- Présentation claire et synthétique du besoin ;
- Les différentes sections présentées sont toutes pertinentes (mais il en manque, notamment la stratégie de validation) ;

Incorrect

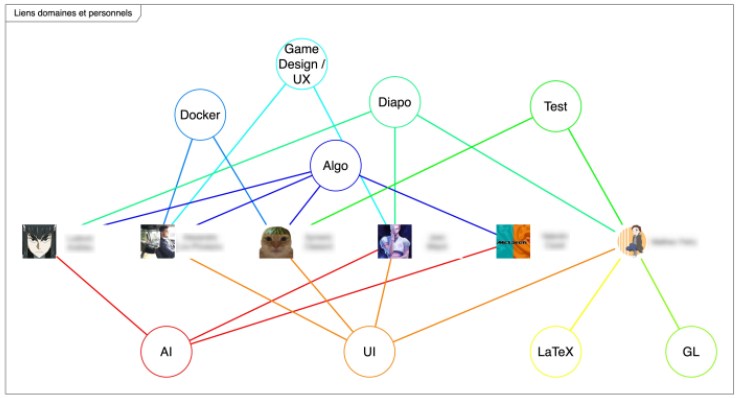
- Les besoins ne sont pas clairement identifiés (quelles sont les options en question, que doivent elles faire ?) ;
- À quels autres besoins celui-ci est-il relié ?
- Quelle est la stratégie de validation de ce besoin ?
- Excès de détails inutiles dans certaines sections et manque de précision dans d'autres ;
- Des prototypes de fonctions sont données mais les arguments sont trop vagues ;
- La moitié des Problèmes/Décisions pourraient être omis car ils ne sont pas pertinents ;
- La façon de noter si le besoin est fonctionnel ou non pourrait se signaler sur la couleur du bandeau de titre plutôt que dans le texte.

Erreurs Générales

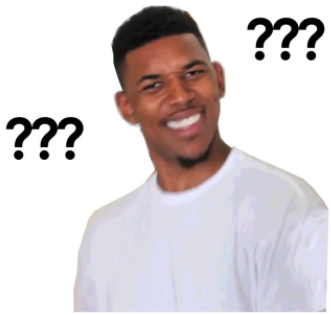
université de BORDEAUX

- **Manque de mise en forme** (pas de titre, pas de table des matières, maîtrise de LaTeX trop approximative, ...) ;
- **Manque de clarté** (phrases trop longues, style trop lapidaire ou trop chargé, pas assez pédagogique, ...) ;
- **Manque de rigueur** (fautes d'orthographe, de grammaire, de typographie, références croisées incorrectes, ...) ;
- **Manque de cohérence** (les parties n'ont pas d'ordre logique, la terminologie n'est pas homogène sur le document, ...) ;
- **Manque de recul** (pas de regard critique, pas de perspective, pas d'ouverture vers d'autres travaux, ...) ;
- **Manque de justifications** (pourquoi avez-vous fait ces choix, donnez arguments et références pour appuyer vos décisions, ...).

Erreurs Générales : Exemple 1



Pour établir notre agenda prévisionnel, nous devons donc combiner le graphe des dépendances avec les préférences de développement des membres de l'équipe. Mais comment faire ?



Nous pourrions réduire ce problème à un problème de coloration de graphe. Cependant, nous avons décidé de ne pas suivre cette approche, car cela nécessiterait trop de temps pour le codage, d'implémentation et les tests. Nous avons estimé que ce développement prendrait au moins une heure, alors que le faire manuellement ne prend que 5 minutes.

Problèmes

- Ce rapport est sensé être un document sérieux, si vous le prenez à la légère, les examinateurs aussi (évitez les traits d'humour déplacés) ;
- Ne faites pas perdre son temps au lecteur en exposant des décisions que vous n'avez finalement pas choisies.
- Justifiez mieux vos conclusions et choix ;
- Enfin, n'oubliez pas qu'il ne s'agit pas d'un journal intime, mais d'un document professionnel. Évitez les tournures de phrases trop personnelles ou trop familières.

Erreurs Générales : Exemple 2

université de BORDEAUX

0.12 Historique des coups - Système de sauvegarde

0.12.1 Structure de l'historique

```java
public class MoveHistory {
    private List<Move> moves;
    private int currentIndex;

    public class Move {
     private Position start;
     private Position end;
     private List<Position> capturedPieces ;
     private boolean promotion;
     private long timestamp;
     }
    }
```

0.12.2 Fonctionnalités de l'historique

Problèmes

- Maîtrise de LaTeX trop approximative (pas de première section ?) ;
- Lorsque vous mettez une image, une table ou du code, il faut ajouter un caption et un label pour pouvoir y faire référence dans le texte ;
- N'utilisez pas une image pour afficher du code, utilisez `minted` ou `lstlisting` ;
- Pas de code sans explication de ce qu'il fait et de comment il s'insère dans le projet ;

Erreurs sur la Bibliographie

université de BORDEAUX

- **Pas de bibliographie** (ou moins de 5 références) ;
- **Bibliographie incomplète** (manque d'informations sur les items : pas de date, d'auteur, de titre, ...) ;
- **Bibliographie non pertinente** (On ne cite pas Wikipedia mais les sources de Wikipedia ! ! !) ;
- **Bibliographie non citée dans le texte** (voir Existant) ;
- **Bibliographie non mise en forme** (Utilisez impérativement bibtex avec LaTeX).

Erreurs Bibliographie : Exemple 1

université de BORDEAUX

6 Bibliographie

- Quoridor AI
- Lloyd, M. (2022). Quoridor Story.
- Glendenning, Lisa. Mastering Quoridor. Bachelor's thesis, University of New Mexico, 2005.
- Mertens, P. J. C. A Quoridor-playing Agent. 2006.

Problèmes

- Ce n'est pas au format BibTeX.
- Il manque une référence sur les 5 attendues ;
- Elles ne peuvent pas être citées dans le texte.
- Les URL doivent être explicitement cités dans le texte ;
- Les références ne sont pas complètes (manque : date, auteurs, type de document, ...).

Erreurs Bibliographie : Exemple 2

université de BORDEAUX

Références

[1] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1) :1–43, 2012.

[2] S. A. Gordon. A faster move generation algorithm. *SOFTWARE—PRACTICE AND EXPERIENCE*, 24(2) :219–232, February 1994.

[3] A. W. A. . G. J. Jacobson. The world's fastest scrabble program. *Programming Techniques and Data Structures*, 31 :572–578, May 1988.

[4] S. J. Metsker. *Building Parsers with Java*. Addison-Wesley Professional, 2001.

[5] B. Sheppard. World-championship-caliber scrabble. *Artificial Intelligent*, 134, January 2002.

Correct

- Utilisation de BibTeX et 5 références ;
- Références complètes (auteurs, titre, date, ...) ;
- Références citées dans le texte (pas visible ici mais c'est le cas) ;
- Références pertinentes (pas de Wikipedia, pas de sources non vérifiables, ...).

Plan

Ce que vous trouverez dans le dépôt

université de BORDEAUX

```
project_name/
|-- README.md # README de base, à modifier...
`-- pdp/
    |-- project-specs.pdf # Specifications initiales
    |-- 01-preliminary/   # Rapport préliminaire (LaTeX)
    |-- 02-final/         # Rapport final (LaTeX)
    |-- references/       # Bibliographie en PDF (articles)
    `-- slides/           # Slides des présentations
```

Le répertoire `pdp/` contiendra tous les rapports, slides et quelques références du projet et ne devra pas dépendre du reste du projet (le code).

Le code source, les tests et la documentation du projet devra être placé à la racine du dépôt en étant bien séparé du répertoire `pdp/`.

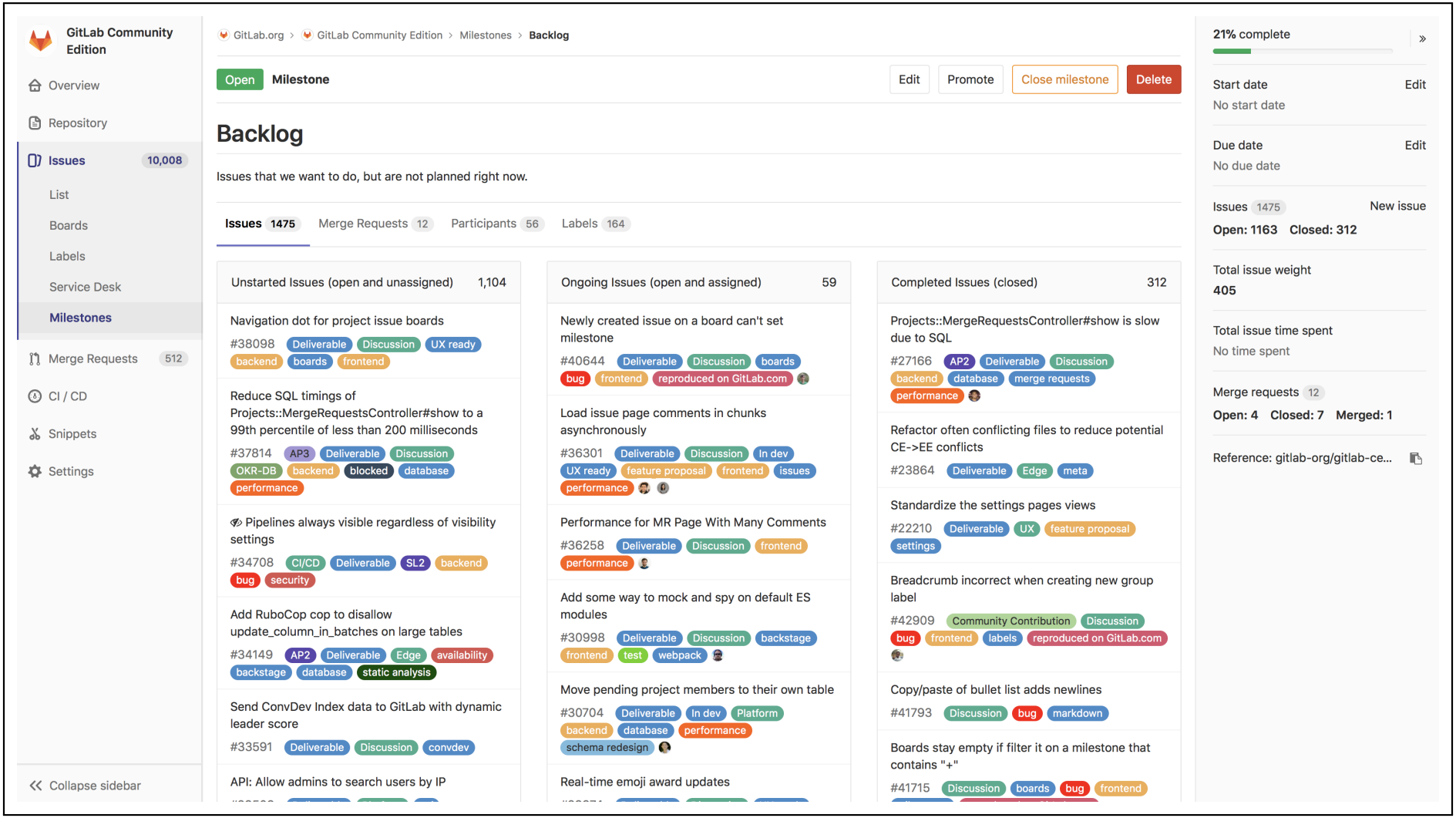
Projet Gitlab (semaine 4)

université de BORDEAUX

- **Mise en place du projet** : Mettez en place la structure initiale du projet (répertoires, fichiers de configuration, le build, les tests, *etc.*) en suivant la fiche qui correspond au langage du projet.
- **Issues et Milestones** : Créer un **issue** pour chaque fonctionnalité en lui donnant **un responsable**. Puis, Créez un **milestone** par TD de rendu de code et associez les issues à rendre pour chaque TD ;
- **Labels** : Utilisez les **labels** pour marquer les besoins fonctionnels, non-fonctionnels, chaque module de votre logiciel ainsi que les bugs, les améliorations, *etc.* ;
- **Continuous Integration** (Optionnel) : Mettez en place un système d'intégration continue pour compiler et tester automatiquement le code à chaque commit.

Gitlab Milestones et Issues

université de BORDEAUX



Plan

Contenu du projet Gitlab (CREMI)

université de BORDEAUX

Le Gitlab du CREMI servira de référence pour la livraison du code hebdomadaire et final. Les éléments suivants seront considérés comme faisant partie du rendu pour la note :

- **Le code source** du projet sur la branche `main` ;
- **Les tests** automatisés ;
- **La documentation** du code ;
- **L'historique** des commits sur `git` ;

Le projet devra aussi contenir un répertoire '`pdp/`' qui contiendra :

- Le code LaTeX des rapports **préliminaire** et **final** ;
- Les PDF des slides des présentations **préliminaire** et **finale** ;

Validation des fonctionnalités

université de BORDEAUX

Chaque semaine, le groupe présentera ses avancements.

- Les fonctionnalités seront **présentées via des commits ou du code**, avec leurs **tests**, leur **documentation** ainsi qu'une démo ;
- Les chargés de TD pourront **valider la fonctionnalité** présentée ou **demander de retravailler** certaines parties ;
- Chaque fonctionnalité sera **comptabilisée dans la note finale** du projet après avoir été revérifiée à la fin du projet.
- Les **fonctionnalités refusées lors du TD devront être retravaillés** et présentés à nouveau la semaine suivante.
- Les **fonctionnalités non validées** à la fin du projet ne seront **pas comptabilisés dans la note finale**.
- Il est aussi possible de discuter des problèmes rencontrés, des solutions possibles, *etc.* avec les chargés de TD.

Contraintes de développement

université de BORDEAUX

Quelques contraintes à respecter durant le projet :

- **Travail en équipe** : Tous les membres du groupe doivent participer raisonnablement au projet (ou bien vous serez sorti du projet) ;
- **Programmer est obligatoire** : Tout le monde doit programmer, ne vous spécialisez pas uniquement dans l'écriture du rapport, la documentation ou les tests. Cela se verra sur l'historique git ;
- **Utilisation de Gitlab** : Le projet doit être développé sur le Gitlab du CREMI. Tout autre dépôt ne sera pas pris en compte pour la notation ;
- **Approbation des chargés de TD** : Les besoins réalisés doivent être présentés chaque semaine en TD pour validation ;
- **Responsabilité sur le code** : Vous êtes responsable du code que vous commitez et vous devez toujours être capables de justifier et d'expliquer ce qui a été produit (sinon vous risquez une minoration de votre note).

Plan

Rapport final (Semaine 12)

université de BORDEAUX

Le rapport (en LaTeX, **25 pages max**) :

- **Explication** détaillée du sujet ;
  (historique du jeu, règles, existant, 10 citations (au moins), *etc.*)
- **Algorithmes** et **structures de données** ;
  (structures de données employées, algorithmes spécifiques, *etc.*)
- **Architecture** du projet ;
  (découpage en modules, dépendances, principales interfaces, *etc.*)
- **Explication approfondie** ;
  (deux fonctionnalités que vous avez réalisées (**détaillez jusqu'au bout**))
- **Revue critique du projet**.
  (ce qui a été fait, ce qui n'a pas été fait, les performances, les cas limites, les limitations, les bugs, améliorations possibles, *etc.*)
- **Liste des 40 besoins** (en annexe, non compté dans les 25 pages).
  (liste de tous les besoins avec une indication de ceux qui ont été réalisés ou pas : **Fait**, **Partiel**, **Non fait**, *etc.*)

Présentation finale (Semaine 13)

université de BORDEAUX

La présentation finale (**25mn**) :

- **Explication** détaillée du sujet ;
  (historique du jeu, règles, existant, *etc.*)
- **Algorithmes** et **structures de données** ;
  (structures de données employées, algorithmes spécifiques, *etc.*)
- **Architecture** du projet ;
  (découpage en modules, dépendances, principales interfaces, *etc.*)
- **Explication approfondie** ;
  (une fonctionnalité que vous avez réalisées (**détaillez jusqu'au bout**))
- **Revue critique de votre projet** ;
  (ce qui a été fait, ce qui n'a pas été fait, ses performances, ses forces, ses limitations, ses bugs encore ouverts, des améliorations possibles, *etc.*)

Plan

Notation

La note finale sera répartie de la manière suivante :

- **Présentation préliminaire** : **10%**
- **Rapport préliminaire** : **10%**
- **Rapport final** : **20%**
- **Présentation finale** : **20%**
- **Code** : **20%** + **Besoins validés** : **20%**

Plan

De quoi parle-t-on ?

L'**IA générative** désigne une catégorie de modèles d'intelligence artificielle capables de générer du contenu original, tel que du texte, des images, de la musique, ou même du code, en réponse à des prompts donnés par un utilisateur.

Chat Bots

Un Chat Bot est un programme capable de converser avec un utilisateur en langage naturel (texte ou voix) en simulant une conversation humaine.

Exemples : ChatGPT, Gemini, Claude, *etc.*

Agents

Un Agent est un programme qui utilise des techniques d'IA pour accomplir des tâches spécifiques avec une certaine autonomie, souvent en interagissant avec son environnement via des outils ou des API.

Exemples : GitHub Copilot, Amazon CodeWhisperer, Claude Code, *etc.*

Dans le cadre de ce projet, l'utilisation d'IA génératives est **autorisée sous certaines conditions** (voir la suite).

Conditions d'utilisation

L'utilisation d'IA génératives est autorisée **à condition que vous respectiez les règles suivantes** :

- **Transparence** : Vous devez ajouter dans vos rapports une section pour expliquer **quels modèles** vous avez utilisés et **comment vous les utilisez**.
- **Compréhension et justification** : Vous devez être capables de d'**expliquer et de justifier tout code ou texte généré par une IA**. Si vous n'arrivez pas à le faire lors d'une présentation ou d'une discussion avec les chargés de TD, vous serez sanctionné ('-1' sur la note finale par occurrence).
- **Restez en maîtrise du projet** : Vous devez toujours **garder le contrôle sur le projet**. N'utilisez pas l'IA pour faire le travail à votre place mais comme un outil pour vous aider. **Vérifiez** toujours ce qui a été généré, essayez de le **comprendre** et de l'**améliorer**.
- **Plus et mieux** : Comme nous autorisons l'utilisation d'IA génératives, le travail rendu doit être **plus complet et de meilleure qualité** pour espérer obtenir la moyenne. Notez que si vous refusez d'utiliser des IA, cela ne vous dispense pas de cette obligation. Tout le monde sera jugé au même niveau.

Comment utiliser les IA génératives ?

université de BORDEAUX

- **Allez-y étape par étape** : Forcez vous à faire de petites étapes que vous pourrez contrôler et comprendre plutôt que de demander de générer un gros bloc d'un coup ;
- **Toujours pouvoir revenir en arrière** : Utilisez '`git`' et les branches pour toujours pouvoir faire marche arrière lorsque vous travaillez avec une IA. Et, vérifiez les différences de code introduites avec '`git diff`' ;
- **Comprenez ce qui est généré** : Passez du temps à chercher à comprendre ce qui a été généré et à vous assurez que vous comprenez bien tout. N'hésitez pas à poser des questions de clarification à l'IA elle-même, elle saura bien souvent vous l'expliquer ;
- **Vérifiez tout ce qui est généré** : Relisez, testez, analysez le code ou le texte généré pour détecter les erreurs ou incohérences. Les IA peuvent générer des erreurs subtiles ou des informations incorrectes, faites donc preuve de vigilance car c'est vous qui en endosserez la responsabilité un fois le commit fait ;
- **Pensez à l'introspection** : Les IAs peuvent aussi être utilisées pour juger de la qualité de votre code ou de votre texte, demandez-leur des critiques constructives sur votre travail (rapport, code, tests, *etc.*) et si vous les trouvez pertinentes, intégrez les ;
- **Ajoutez votre patte personnelle** : Il est connu que les IAs laissent derrière elles des parties qui mériteraient d'être améliorées. N'hésitez pas à retravailler le code ou le texte généré pour l'améliorer, le rendre plus clair, plus efficace ou plus adapté à vos besoins. Considérez le travail de l'IA comme un point de départ et non comme un produit fini.

Sanctions en cas de problème

université de BORDEAUX

Si l'un des membres du groupe ou le groupe ne se comporte pas correctement, l'équipe pédagogique prendra des sanctions :

- **Absences** : La présence aux TDs hebdomadaires est obligatoire, chaque absence non justifiée sera sanctionnée par un '-1' cumulatif sur la note finale pour chaque manquement.

  Venez à tous les TDs hebdomadaires, ils sont obligatoires.

- **Imposture** : Chaque fois que vous n'arriverez pas à expliquer ou justifier vos productions, vous écoperez d'un '-1' cumulatif sur votre note finale.

  Vous êtes considérés comme responsables de ce que vous rendez, vous devez pouvoir l'expliquer et le justifier.

- **Passivité** : Si un membre du groupe ne participe pas suffisamment au projet (nombre de commits, qualité des commits, absence aux TDs, ...), il sera mis sur un fork du projet et devra continuer seul le reste du projet. Sa note finale sera calculée en fonction de son travail personnel uniquement ;

  Restez attentif à bien travailler de manière régulière avec votre groupe et à rendre compte de vos avancées aux chargés de TD.

Site Web de l'UE

https://pdp-projects-20f056.pages.emi.u-bordeaux.fr/

On y trouve :

- Ces slides ;
- Le planning du projet ;
- Des références vers des ressources utiles ;
- Des liens vers le cours et les fiches de mise en place du projet.

Questions ?

Projets 2025-2026

université de BORDEAUX

- **Agon** : Information parfaite, déterministe ;
- **Amazons** : Information parfaite, déterministe ;
- **Bridge** : Information **imparfaite**, **stochastique** ;
- **Carcassonne** : Information parfaite, **stochastique** ;
- **Checkers** (Dames) : Information parfaite, déterministe ;
- **Othello** : Information parfaite, déterministe ;
- **Quoridor** : Information parfaite, déterministe ;
- **Scrabble** : Information **imparfaite**, **stochastique** ;
- **Shatranj** (Échecs Indiens) : Information parfaite, déterministe ;
- **Shogi** (Échecs Japonais) : Information parfaite, déterministe.