Projet d'informatique

UMONS – Année académique 2019 – 2020

Le contenu de ce document est présenté lors de la première séance du projet, le mardi 11 février 2020. Il vous est conseillé de relire attentivement ce document et d'y revenir au besoin.

1 Objectifs du projet

Les cinq principaux objectifs de ce projet sont les suivants :

1. Participer au développement d'un logiciel de taille conséquente (représentant environ 120 heures de travail, à répartir entre 2 étudiants).
2. Appliquer les notions d'*algorithmique* vues au cours « Programmation et Algorithmique 1 ».
3. Appliquer les notions de programmation *orientée objet* vues au cours « Programmation et Algorithmique 2 ».
4. S'initier aux bonnes méthodes / habitudes utiles au développement d'un projet logiciel (design modulaire des classes, éviter la redondance du code, documentation correcte, création et utilisation de tests, création de packages, etc.).
5. Apprendre à gérer son temps, à travailler en groupe et à s'organiser.

Gardez ces objectifs à l'esprit : ils nous servent également de critères pour vous évaluer (voir ci-dessous).

Contenu

2 Déroulement général du projet

Le but final de ce projet est de réaliser une application graphique en Java permettant de jouer au jeu de Quoridor contre une des différentes intelligences artificielles (IA) que vous implémenterez. L'application devra en outre permettre de charger une partie précédemment enregistrée. Une fois que les fonctionnalités de base exigées seront correctement mises en place, vous aurez le loisir de personnaliser et d'étendre votre projet (ce qui est encouragé, notamment via l'évaluation comme expliqué ci-dessous).

Le déroulement du projet est particulier. Vous allez devoir le réaliser petit à petit, mais sans perdre de temps et en vous répartissant correctement les tâches. Vous devrez faire des choix consciencieux lors de la conception. Pour ce faire, vous utiliserez au mieux les notions de la programmation *orientée objet* vues au cours de « Programmation et Algorithmique 2 ». Ce cours étant donné en parallèle au projet, nous vous proposerons un calendrier adapté et nous organiserons des séances particulières pour vous aider sur un thème précis (voir ci-dessous).

2.1 Séances à l'horaire et travail en dehors de celles-ci

Parmi les heures prévues à l'horaire, il y a deux types de séances :

— **Séances spéciales** : séances (obligatoires) durant lesquelles un assistant présente un sujet directement utile pour la réalisation du projet (par ex., une séance sur la conception d'interfaces graphiques). La première séance de ce type aura lieu le mercredi 19 février 2020.

— **Séances questions** : séances (facultatives) où un assistant est présent pour répondre aux questions. Vos questions doivent être préparées à l'avance car l'assistant passera une et une seule fois parmi chaque groupe pour y répondre !

Un calendrier reprenant le type des séances est disponible sur moodle.

Le nombre d'heures nécessaires pour réaliser le projet est important (60h par personne, ce qui fait environ 120 heures de travail) et les heures indiquées dans l'horaire ne sont donc pas suffisantes : ce sont des heures destinées à vous donner des consignes ou des conseils, et permettant de répondre à vos questions. Pour mener à bien votre projet, vous devrez les compléter par des heures de travail en dehors des séances ! Essayez également de vous répartir les tâches de manière équilibrée et de vous organiser au mieux au sein de votre groupe. La gestion de votre temps est une des difficultés de ce projet : ne laissez pas le retard s'accumuler.

2.2 Aide et réponses aux questions : séances

Nous ne répondons pas aux questions par email. Par contre, outre les séances « questions » décrites ci-dessus, un **forum** est disponible sur moodle pour y poser vos questions et pour vous entraider entre étudiants. L'utilisation du forum doit se faire de manière constructive et courtoise :

— Soignez la manière de rédiger vos questions et vos réponses pour qu'elles soient claires et compréhensibles.

— Vous êtes autorisés à répondre aux autres étudiants mais dans le but de les faire réfléchir ou en donnant des pointeurs vers des références utiles. Ne donnez donc pas de solution toute faite. Ne postez pas non plus du code qui sera utilisé dans votre projet.

— L'équipe enseignante ne donnera a priori pas de réponse, sauf si la question (et sa

réponse) s'avère constituer un éclaircissement utile à l'ensemble des étudiants. Même dans ce cas, n'attendez pas de réponse immédiate : il peut être nécessaire de discuter des éclaircissements à donner avant de les publier.

L'équipe enseignante peut modifier (corriger) ou supprimer vos messages si cela s'avère nécessaire. Une utilisation abusive du forum entraînera la fermeture de celui-ci.

Pour éviter que les choses importantes soient réalisées en dernière minute, nous ne répondrons plus à vos questions lors des derniers jours avant la remise du projet. En pratique, la dernière séance pour poser vos questions aura lieu le lundi 11 mai 2020. Vous êtes prévenus plusieurs mois à l'avance : prenez donc vos dispositions !

2.3 Soumission et défense du projet final

En fin d'année, vous rendrez un rapport écrit, votre code source et défendrez oralement votre projet (la défense orale a lieu pendant la session d'examen de juin). L'évaluation se base sur tous ces points (voir ci-dessous).

En ce qui concerne la rédaction du rapport, sachez que la présentation et l'orthographe seront considérées lors de l'évaluation. Pour vous aider, le document « Eléments de rédaction scientifique en informatique » est mis à votre disposition sur e-learning et rassemble quelques conseils [1].

La défense orale se fera par groupe lors de la session de juin. Elle commencera par une démonstration du programme (environ 10 minutes) et se terminera pas une série de questions posées de manière individuelle ou collective. Même si une partie du programme a été réalisée par une personne du groupe, les deux membres doivent être capables de répondre aux questions. La date et le lieu vous seront communiqués ultérieurement (via les horaires d'examens de la première session). Notez que les deux étudiants d'un groupe seront interrogés de manière équilibrée lors de la défense et n'obtiendront pas forcément la même note.

3 Aspects administratifs et consignes

3.1 Composition des groupes

Le projet est réalisé par **groupes de 2 étudiants**. Vous nous communiquerez la composition de votre groupe pour le jeudi 27 février 2020 au plus tard via Moodle. Nous nous attendons à ce que le projet soit réalisé à deux et à vous voir tous les deux présents lors de la défense. **Vous devez gérer vous même les soucis éventuels (abandon, manque d'implication de l'un, etc.).** La gestion correcte du groupe fait partie des objectifs à atteindre et vous serez amenés à faire plusieurs travaux de groupes tout au long de vos études.

3.2 Rapport

Le rapport final doit contenir :

— la répartition des tâches au sein du groupe ;

1. Tout au long de vos études vous serez amenés à rédiger différents rapports, jusqu'à la rédaction de votre mémoire.

— une description argumentée des choix personnels effectués ;
— les points forts de votre projet (fonctionnalités supplémentaires, optimisation, complexité,...) ;
— les points faibles de votre projet (vitesse d'exécution lente, faiblesses de certains algorithmes,...) ;
— les différentes erreurs connues du programme. Les erreurs rencontrées lors de nos tests et qui ne sont pas répertoriées dans le rapport seront considérées plus sévèrement ;
— les apports positifs et/ou négatifs de ce projet ; et
— un mini guide utilisateur précisant la manière dont l'application peut être utilisée (par exemple pour sauver ou charger une partie).

En plus des éléments repris ci-dessus, vous pouvez ajouter tout élément qui vous semble utile comme par exemple un *diagramme de classes UML* (cf. cours de « Programmation et Algorithmique 2 »).

Le rapport devra être remis en PDF. **Attention à votre orthographe, elle sera considérée lors de l'évaluation.** Pour rappel, une note contenant des conseils de rédaction est disponible sur la page moodle du projet.

Pour vous aider à réaliser votre rapport, nous vous conseillons de maintenir un « journal de bord » tout au long du projet (concrètement, cela peut être un petit carnet ou un simple fichier de texte). Celui-ci vous permettra d'y noter progressivement vos idées, d'y indiquer la répartition des tâches (qui réalise quelle tâche au sein de votre groupe). Egalement, si quelque chose dans l'énoncé vous paraît ambigu ou trop peu défini, vous devrez faire un choix (en le justifiant) : notez ce choix et vos arguments dans votre journal puis dans votre rapport.

3.3 Code source

Voici les consignes à respecter concernant le code source.

— Les noms des interfaces, classes et méthodes que vous créerez seront en **anglais**.
— Le code doit obligatoirement être **documenté** en utilisant la javadoc de base (`@return`, `@param` et `@throws`).
— Les commentaires et la documentation peuvent être en français ou en anglais (mais restez cohérents une fois la langue choisie).
— Vous devrez inclure des tests unités pour **au moins une partie précise de votre projet** (voir section 9.4).
— Pour l'évaluation et la défense orale, votre code sera exécuté sur une machine de l'université et seule la version soumise sera prise en compte. La description technique de la machine utilisée correspond à une machine des salles de TPs (Salles Russell ou Vaughan, De Vinci, sous Linux).
— Vous devez utiliser *Gradle* (voir Section 4) pour simplifier et automatiser le processus de compilation et d'exécution de votre code source : en ouvrant votre archive les 4 commandes suivantes doivent impérativement avoir le comportement attendu :

1. `gradle clean` : supprime tous les fichiers `.class`, s'il y en a ;
2. `gradle build` : compile votre programme ;
3. `gradle run` : exécute votre programme ;
4. `gradle test` : lance les tests unités.

3.4 Défense du projet

Lors de la défense orale du projet (pendant la session de juin), votre application sera exécutée sur une machine des salles Russell ou Vaughan et non pas sur un ordinateur personnel.

3.5 Plagiat et triche

Le *plagiat* consiste à s'approprier comme personnel du texte, une image ou du code réalisé par une autre personne (en ce compris un texte traduit), sans le préciser de manière explicite. Vous pouvez baser vos arguments sur des éléments que vous avez lu, mais vous devez citer vos sources. S'appuyer sur des résultats connus pour en développer de nouveaux est d'ailleurs à la base de la démarche scientifique. Il faut simplement être honnête et très clair sur ces points, mais le plagiat est donc simple à éviter.

D'autre part, le plagiat est **strictement interdit** à l'université et peut entraîner des sanctions graves allant beaucoup plus loin que le simple échec au projet (cf. règlement des études et page dédiée à ce sujet sur le site de l'université [2]). Vous devez soumettre vos fichiers via moodle (et donc pas par email) où ils subiront une détection automatique du plagiat.

Le projet étant individuel, le fait que deux groupes différents ont manifestement échangé du code sera donc considéré comme de la triche et sera sanctionné en conséquence (pour les deux groupes).

3.6 Notes de présence et absences

En ce qui concerne le projet en première session, vous obtiendrez
- une **note entre 0 et 20** si le projet est soumis, recevable *et* défendu oralement pendant la session ;
- une **note de 0 sur 20** si le projet n'est pas recevable pour au moins un des critères énoncés en Section 5 ;
- une **note de présence** si le projet n'est pas déposé mais que vous signalez votre intention de ne pas le soumettre en envoyant un email à `hadrien.melot@umons.ac.be` avant le vendredi 15 mai 2020 à midi (vous recevrez un accusé de réception).

Dans *tous les autres cas* (soumis mais pas défendu, pas soumis sans nous prévenir, etc.), vous serez noté **absent** pour le projet d'informatique.

4 Gradle

Il vous est demandé de fournir un fichier build.gradle afin de permettre l'utilisation de *Gradle* [3] pour compiler, exécuter et tester votre projet.

En ouvrant votre archive dans une console, les 4 commandes suivantes doivent impérativement avoir le comportement attendu :

1. `gradle clean` : supprime tous les fichiers `.class`, s'il y en a ;

2. `https://sharepoint1.umons.ac.be/EN2/universite/admin/aff_academiques/Pedagogie_Qualite/Plagiat/plagiat.html`
3. `https://gradle.org/`

2. `gradle build` : compile votre programme ;
3. `gradle run` : exécute votre programme ;
4. `gradle test` : lance les tests unités.

Pour cela, vous devrez respecter la structure imposée par *Gradle* :

```
<votre projet> ..... dossier de votre projet
├── build.gradle
└── src
    ├── main
    │   ├── java ..... contient les fichiers .java
    │   └── resources ..... contient les resources (images, ...)
    └── test
        └── java ..... contient les fichiers .java pour les tests
```

5 Checklist des éléments indispensables pour que votre projet soit corrigé !

Vérifiez scrupuleusement les éléments repris ci-dessous qui décrivent comment nous déterminerons si votre projet est recevable. **Un projet non recevable n'est pas corrigé et entraîne une note de 0 / 20.** Notez que ce sont des choses toutes simples à vérifier et qu'aucune exception ne sera faite, quelles que soient les circonstances. Ce serait vraiment dommage de ne pas pouvoir présenter votre projet à cause d'un des points ci-dessous !

1. *Deadline.* La date limite de remise est le **vendredi 15 mai 2020 à 12h**. **La plateforme n'acceptera pas de retard.** Aucun ajout ni aucune modification du projet ne sera acceptée au-delà de cette date.
2. *Compilation.* Pour tester si votre projet est recevable nous procéderons à l'exécution des 3 commandes suivantes, dans cet ordre : `gradle clean`, `gradle build`, `gradle run`. Votre code doit pouvoir être compilé et exécuté sur une machine de l'université et seule la version soumise sera prise en compte. La description technique de la machine correspond à une machine des salles Russell et Vaughan (sous **Linux (Ubuntu)**). La compilation **ne peut pas produire d'erreur** empêchant la création des fichiers `.class` (warnings autorisés). En d'autres mots, nous devons être capable d'exécuter le programme sans modifier le code.
3. *Archive.* Le travail doit être livré sous la forme d'une archive (`.zip` ou un format libre) portant, en majuscules uniquement, votre nom. L'archive **doit** contenir un répertoire portant ce même nom. Ce répertoire comporte :
   a) le rapport au format **PDF** ;
   b) le code de votre application (en respectant la structure imposée par *Gradle* comme expliqué dans la Section 4) ;
   c) un fichier `build.gradle` (*Gradle*) pour permettre d'utiliser les quatre commandes `gradle` décrites ci-dessus.

   Ne sont requis que le code source (fichiers `.java`), les tests unités, ainsi que les fichiers nécessaires à la compilation et l'exécution (par ex. images). Votre ne devez **pas** inclure les fichiers compilés `.class`. Si vous souhaitez fournir d'autres éléments, placez-les dans un sous-répertoire nommé `misc` et indiquez dans un fichier `README.txt` les détails de ces différents éléments.

6 Conseils

Comment éviter le stress et m'assurer à temps que mon projet sera recevable ?

1. *Mettez la priorité là où il le faut.* Assurez-vous d'avoir en priorité une version simple mais qui répond à nos exigences et qui implémente les fonctionnalités demandées. Ensuite, s'il vous reste du temps, vous pourrez peaufiner le côté esthétique ou améliorer d'autres points de détails.
2. *Rédaction.* Rédiger prend du temps ! Le rapport ne doit pas être plus long que nécessaire mais doit contenir ce qui est attendu (voir section 3.2), et le présenter de manière adéquate. Commencez votre rapport au plus vite (pourquoi pas dès maintenant ?). Soignez la forme autant que le contenu et surtout : citez vos sources et évitez le plagiat (voir ci-dessus).
3. *Test réel.* Faites un test dans la salle Escher en utilisant exactement ce que nous recevrons :
   — ouvrez sur une machine du Escher l'archive telle que vous comptez la soumettre ;
   — testez la compilation et l'exécution.
4. *Soyez prévoyants.* Prévoyez de soumettre une version préliminaire au moins 24 heures à l'avance pour éviter toute mauvaise surprise (problèmes de connexion, etc.). C'est toujours mieux d'être évalué sur cette version sans doute imparfaite [4], plutôt que de ne pas être jugé du tout !
5. *Réfléchissez bien à votre design.* Ce projet comporte des éléments (comme les différentes pièces du jeu de Stratego) qui permettent une exploitation intense de certains concepts de la programmation orientée objet (comme la notion d'héritage ou les interfaces). Etant donné que ceci est votre premier projet de la sorte, trouver la bonne utilisation de ces notions ne sera pas facile du premier coup. N'hésitez pas à penser à plusieurs solutions et à les comparer avant d'aller trop loin dans votre développement. N'hésitez pas non plus à recommencer votre design si vous vous rendez compte qu'il n'est pas bien adapté. Par exemple, si vous constatez que vous avez des méthodes qui contiennent des instructions conditionnelles avec un grand nombre de branches ("if-else if-else if-...-else"), vous n'avez sans doute pas exploité suffisamment le polymorphisme. Nous sommes là pour vous aider et discuter de vos idées.
6. *Petit à petit, l'oiseau fait son nid.* Au début du projet, vous n'aurez pas encore à votre disposition toutes les connaissances en Java pour implémenter complètement votre projet. Ce n'est pas grave : faites une première version simplifiée avec des choses que vous savez faire. Par exemple, vous pourriez commencer par une première version d'un Stratego très simplifié en mode "texte" (dans la console) où il n'y a qu'une seule pièce qui peut se déplacer sur le plateau...

7 Evaluation de votre projet

Sur quels critères vais-je être évalué ?

Nous nous basons sur le rapport, le code et la défense orale. Pour chacun de ces éléments, une série de critères et de questions sont utilisés pour évaluer la qualité d'un projet.

4. Si vous êtes trop perfectionniste, quelle version sera vraiment parfaite ?

1. *Rapport.*
   (a) Forme : orthographe, style adapté, structure, clarté.
   (b) Contenu : le contenu est-il complet et informatif ? Pour rappel, votre rapport doit contenir :
   — Une description argumentée des choix personnels effectués ; en particulier vous expliquerez les idées exploitées pour décrire vos IA ;
   — Les points forts de votre projet (fonctionnalités supplémentaires, optimisation, complexité,...) ;
   — Les points faibles de votre projet (vitesse d'exécution lente, faiblesses de certains algorithmes,...) ;
   — Les différentes erreurs connues du programme. Les erreurs rencontrées lors de nos tests et qui ne sont pas répertoriées dans le rapport seront considérées plus sévèrement ; et
   — Les apports positifs et/ou négatifs du projet.
2. *Code.*
   (a) Les fonctionnalités de base sont-elles respectées ?
   (b) Des éléments supplémentaires aux fonctionnalités de base ont-ils été ajoutés ?
   (c) Les tests unités (pour au moins une partie du projet) sont-ils complets et pertinents ?
   (d) Les algorithmes ont-ils été écrits avec un souci d'efficacité ?
   (e) Les notions de la programmation Orientée Objet (héritage, interface, exceptions, etc.) sont-elles utilisées de manière correcte, élégante et pertinente ?
   (f) La documentation (javadoc de base : `@return`, `@param` et `@throws`) est-elle présente ?
   (g) La documentation (javadoc) est-elle pertinente et claire ?
   (h) Le code est-il de bonne qualité et bien organisé ?
   — lisibilité ;
   — non redondance ;
   — design modulaire ;
   — organisation en packages ;
   — respect des conventions, etc.
3. *Défense orale.*
   (a) Réponses aux questions.
   (b) Intérêt de la démonstration.
4. *Respect des consignes.* Présence aux séances obligatoires, etc.

Rappel : les étudiants au sein d'un groupe n'obtiendront pas systématiquement la même note. Cela dépendra des réponses aux questions ou d'autres critères objectifs.

8 Prototype en Python

Avant de se lancer dans un gros projet, il est utile de réaliser un prototype. Un prototype est une version rapide d'un projet, souvent implémentée sans interface graphique et dans un langage syntaxiquement simple (comme le Python). Il sert à dégrossir les choses pour mieux comprendre le sujet, commencer à réfléchir au design des classes, se rendre compte de ce qui

pourrait marcher ou pas, etc. Il n'est pas nécessaire que le prototype soit écrit dans le langage final, puisque son but est de mieux appréhender le projet et de concrétiser les choses. De plus, étant donné que le cours de « Programmation et Algorithmique 2 » qui vous apprendra le Java est donné en parallèle au projet lors de ce quadrimestre, cela vous permettra de commencer rapidement puisque vous connaissez déjà le Python.

Pour ces raisons, **nous vous conseillons fortement d'écrire un prototype en Python avant de commencer votre projet en Java**. Il serait intéressant que vous indiquiez dans votre rapport si celui a été utile et s'il vous a permis par exemple de mieux concevoir vos classes.

9 Description de votre application

Il est attendu que chaque groupe d'étudiants conçoive une application logicielle (écrite en Java) en essayant au maximum d'appliquer les concepts et principes vus au cours de l'année, principalement les concepts orienté objet (héritage, interfaces, exceptions, ...). A ce sujet, la section 7 donne la liste des critères qui nous serviront à évaluer votre projet.

Votre application doit permettre, entre autres, de jouer à Quoridor Classic (Gigamic) contre un joueur humain ou contre l'ordinateur via une interface graphique et en utilisant la souris. La page `http://www.gigamic.com/jeu/quoridor` présente ce jeu notamment via une vidéo et permet de le jouer en ligne. Votre jeu utilisera les règles officielles qui sont disponibles sur la page `http://www.gigamic.com/files/catalog/products/rules/quoridor-classic-fr.pdf`.

FIGURE 1 – Le plateau du jeu Quoridor (Source : www.gigamic.com)

Les différentes options de jeu (voir ci-dessous) pourront être sélectionnées via des menus.

9.1 Intelligences Artificielles et modes de jeux

Votre application doit permettre à deux joueurs humains de jouer l'un contre l'autre.

De plus, vous déterminerez au moins 2 intelligences artificielles (IA) pour permettre de jouer à Quoridor contre l'ordinateur. La première peut être très simple et naïve. Votre deuxième IA sera plus subtile et essayera par exemple d'exploiter une certaine stratégie (essayez de décrire les méthodes que vous utiliseriez vous-même en jouant). N'oubliez pas de décrire les algorithmes utilisés par vos IA dans le rapport. Vos deux IA de base devront jouer

dans un délai raisonnable (de l'ordre de moins de 5 secondes pour un coup).

Vous pouvez envisager autant d'IA supplémentaires (facultatives) que vous le voulez en plus de vos deux IA de base. Les IA supplémentaires n'ont pas de contraintes.

L'utilisateur doit pouvoir, via l'interface graphique, choisir le mode de jeu : humain contre humain, humain contre IA, IA contre IA, ainsi que de choisir parmi les IA disponibles (par exemple « `Aleatoire` » ou « `Difficile` »).

9.2 Modes d'exécution : graphique ou statistique

Le projet doit présenter deux modes d'exécution distincts : un premier mode permettant de jouer via l'interface graphique comme expliqué ci-dessus, et un second mode visant à fournir des statistiques basiques sur un ensemble de parties aléatoires jouées par deux IA. Quand le projet est exécuté via *gradle run*, c'est le mode graphique qui doit être lancé. Dans votre rapport (section "mini-guide" utilisateur), vous expliquerez comment lancer le mode statistique. En mode statistique, aucune interface graphique ne doit apparaître et les résultats sont affichés dans la console. Par exemple, en supposant que le point de lancement du mode statistique se situe dans un fichier nommé `MainStat.java` et que vous ayez deux IA nommées `Aleatoire` et `Difficile`, l'exécution de la commande « `java MainStat 30 Aleatoire Difficle` » effectuera ceci :

- faire jouer 30 fois l'IA `Aleatoire` contre l'IA `Difficile` en prenant soin d'alterner l'IA qui joue en premier ;
- afficher à l'écran des statistiques montrant quel est le pourcentage de victoires des deux IA (et éventuellement des statistiques plus fines, comme la « durée » d'une partie, c'est à dire le nombre de coups joués ; où le pourcentage de victoires en tant que premier joueur, etc.).

9.3 Archivage

En mode graphique, il doit être possible à tout moment de sauvegarder une partie en enregistrant l'état du jeu courant. Egalement, l'application doit permettre de charger une partie précédemment sauvegardée pour pouvoir la continuer.

9.4 Tests unités

Vous devez **obligatoirement** fournir des tests unités pour tester (cf. règles du jeu) :

a) la validité d'un déplacement d'un pion : un pion ne peut se déplacer que d'une case horizontalement ou verticalement, en avant ou en arrière. Il est interdit pour un pion de « traverser » une barrière ou de sortir du plateau. Vous pouvez ici écrire une série de tests dont chacun vérifie qu'un déplacement valide est accepté et qu'un déplacement invalide est bien refusé par votre code.

b) la validité de la pose d'une barrière : une barrière doit être placée entre 2 blocs de deux cases (voir figures dans les règles du jeu). Il est interdit de placer une barrière qui dépasserait des limites du plateau ou qui empêcherait l'adversaire d'atteindre son but. Soyez particulièrement attentifs à cette dernière condition et expliquez dans votre rapport l'algorithme utilisé pour vérifier qu'un joueur peut atteindre son but.

c) la validité de l'application de la règle du « face à face » (voir règles du jeu).

d) les résultats de victoire : la partie s'arrête-t-elle bien quand un joueur atteint son but ?

Vous pouvez, de manière facultative, tester tout autre comportement ou fonctionnalité de votre programme.

9.5 Éléments supplémentaires

Les éléments décrits dans les sections précédentes constituent la version de base de votre projet. Une version de base qui serait *parfaitement* conçue et qui attesterait du souci de bien faire (bonne utilisation du paradigme orienté objet, clarté du code, jouabilité, pas de bug détecté, documentation complète, respect des consignes, rapport clair et complet, etc. : voir Section 7) vous permettrait d'avoir une note globale de maximum 16/20.

Pour augmenter la note, diverses choses supplémentaires peuvent être ajoutées au projet. Par exemple :

- Permettre à plus de 2 joueurs de participer à Quoridor sur un même plateau de jeu.
- Proposer des variantes du plateau de jeu. Ces variantes doivent concerner aussi bien la taille que la forme du plateau de jeu, ainsi que la pose de murs initiaux. Les variantes doivent être **intéressantes** à jouer et au moins une IA doit être fonctionnelle sur ces variantes.
- Une variante *mondes parallèles* dans laquelle les joueurs jouent sur (au moins) deux plateaux en parallèle. Lors d'un même tour de jeu, le joueur peut déplacer chacun de ses pions et également poser exactement un mur. Ce mur apparaît alors au même endroit sur les différents plateaux.
- Le plateau de jeu contient des *bonus*. Une fois ramassé, ces bonus peuvent être utilisés une seule et unique fois par le joueur. Un bonus est automatiquement ramassé lorsqu'un pion est sur sa case ou une case adjacente, non séparée par un mur. Exemples de bonus : ajouter un mur supplémentaire ; déplacer ou supprimer un mur ; déplacer un pion d'une case, même à travers un mur ; faire passer le tour d'un autre joueur ; ...
- Chaque jouer a un nombre limité de *pioches*, lui permettant de creuser un trou dans un mur adjacent.
- Proposer des types de murs particuliers qui ont un comportement différent des murs de base. Exemples : murs pouvant être traversé une seule fois et par un seul joueur ; murs pivotant lorsqu'un joueur passe sur une case adjacente ; murs clignotants qui ne sont présents qu'un tour sur deux ; murs ayant une durée de vie limitée, en nombre de tours ; faux murs, à travers desquels il est possible de passer, et seul le joueur l'ayant placé le sait ; ... Ces murs doivent être pris en compte lorsqu'on vérifie que les joueurs peuvent atteindre leur objectif.
- Un mode de jeu avec visibilité limitée aux cases et murs déjà rencontrés par chaque joueur.
- Un mode de jeu dans lequel les joueurs ne peuvent pas reculer. Ils doivent néanmoins toujours pouvoir atteindre leur objectif.
- Dans l'interface graphique, mettre en évidence les cases éligibles pour le déplacement d'un pion. De la même manière, si une prévisualisation d'insertion de mur est disponible, afficher différemment les insertions non autorisées.
- Dans l'interface graphique en mode joueur contre joueur ou joueur contre IA, ajouter un composant suggérant un coup au joueur courant.
- Changer la topologie du plateau en grille hexagonale. Ainsi, chaque case est adjacente

à six autres cases, les murs sont toujours de longueur 2 (et ont une forme de « V »). La Figure 2 illustre un tel plateau dans lequel un mur a été inséré, avec les déplacements d'un pion.

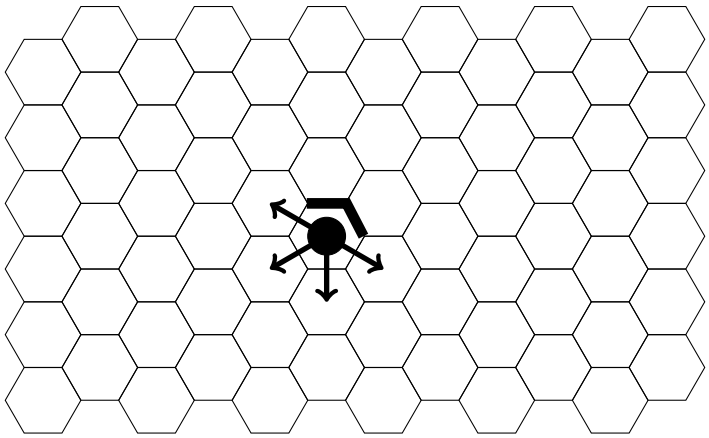

FIGURE 2 – Variante hexagonale de Quoridor

— Un mode plateau cylindrique. Lorsqu'un pion atteint l'extrémité droite du plateau, il est « téléporté » à l'extrémité gauche, et inversement. Un mur peut alors être « à cheval » sur la jonction.
— Ajouter un jeton « fantôme » utilisable une unique fois au cours du jeu permettant à un joueur de traverser un mur.
— Mettre en place une pendule. À chaque tour de jeu, le joueur dont c'est le tour voir le temps qu'il lui est imparti globalement pour le reste de la partie diminuer. Ce compte à rebours s'arrête une fois qu'il a joué son coup. Si un joueur ne dispose plus de temps, il a automatiquement perdu.
— Permettre de « revenir en arrière » dans les coups joués par un joueur humain.

Cette liste n'est pas exhaustive et les initiatives personnelles de qualité sont encouragées (il est cependant conseillé d'avoir d'abord une version de base solide avant de vous lancer dans les éléments optionnels).

Bon travail !