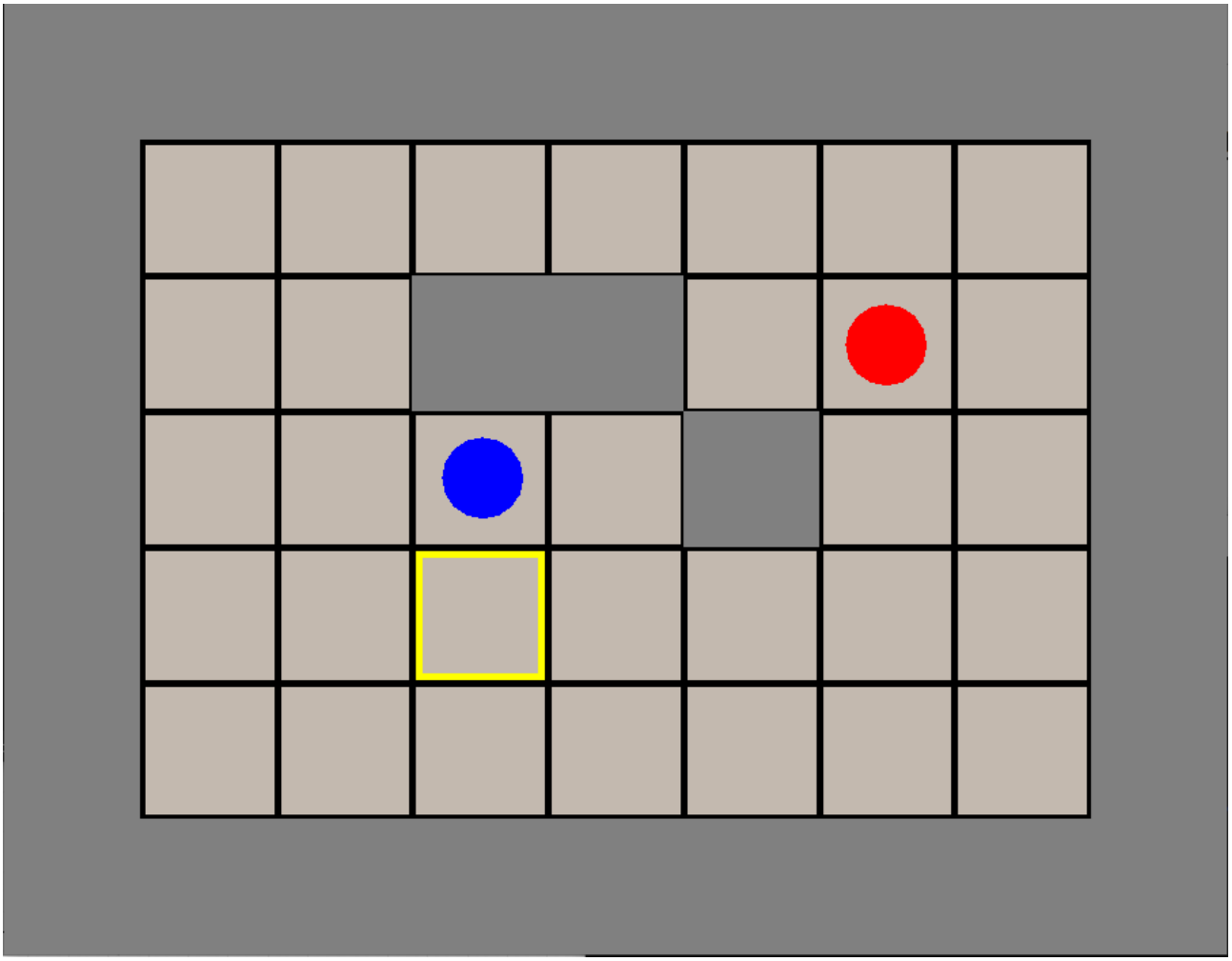
Rapport - Projet Java M3105



Jeu Isola programmé par moi-même

Étudiant : **Florian Cassagne** – TP1
Module : **M3105**
Sujet : **Isola** (N°5)

Sommaire :

I. Introduction

1) Présentation du projet :

Dans un premier temps, nous allons expliquer le principe du projet, notre projet consiste à reproduire le jeu de plateau Isola sous forme de logiciel informatique. Le but du projet est aussi d'appliquer au maximum les connaissances de programmation orientée-objet (en java) apprises durant les cours du module M3105.

Qu'est ce que le jeu Isola ?

Le jeu Isola est constitué d'une grille, en général de 5x5 cases, mais cela peut varier selon les modèles de ce jeu, dans notre programme on utilise une grille de 7x5 cases.

Dans le jeu Isola, il y a obligatoirement **2 joueurs**, représentés par des pions et chacun joue à tour de rôle. A chaque tour de rôle, le joueur peut faire **2 actions** :

-La première action est **se déplacer** d'une case (également les cases "diagonales" par rapport au joueur).

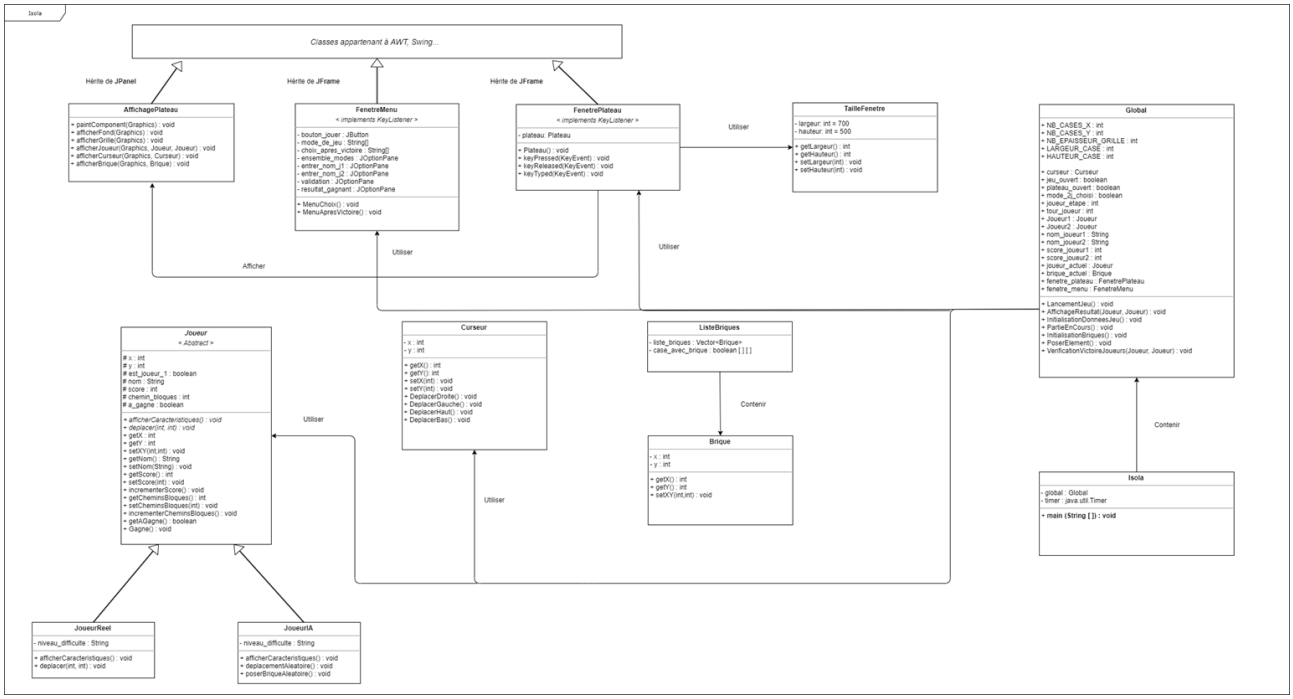
-La deuxième consiste à **poser une brique** (selon les gens, on peut dire que le joueur pose une brique ou qu'il cree un trou) sur n'importe quelle cases du plateau, et cela permet de rendre la case inaccessible par la suite.

-Après les 2 actions du joueur, c'est au tour de l'autre joueur de jouer, puis on recommence ainsi tant que personne a gagné.

-Pour qu'un joueur gagne, il faut qu'il arrive à **enfermer l'autre joueur par des briques,** de manière à ce qu'il ne puisse pas s'en sortir.

II. Analyse

1) Diagramme de classe UML :



Pour mieux le voir, je vous invite à aller dans le dossier "***_Rapport Technique***", ouvrir l'image "***Diagramme UML v1***", puis zoomer sur les parties qui vous intéressent.

2) Classes utilisés + leur fonctionnement :

-> Classe **FenetreMenu :**

• Cette classe permet d'afficher les différents menus proposés :
-Au lancement du programme, permettant à(aux) l'utilisateur(s) de choisir son mode de jeu et le nom des joueurs.
-Après la victoire d'un joueur, afin de proposer à l'utilisateur de relancer une nouvelle partie ou de quitter le programme + pour afficher le score de chaque joueur (1 victoire = 1 point en plus pour le score global de chaque joueurs).
• Hérite de ***JFrame***

-> Classe **FenetrePlateau :**

• Cette classe permet d'afficher le plateau de jeu Isola, mais également de gérer les appuis sur certaines touches de clavier, par exemple si on appuie sur la touche "Entrée", ça appelera une fonction d'une autre classe (qu'on verra par la suite) permettant de poser une brique.
• Elle gère aussi à dessiner les éléments (joueurs, cases...) grâce à un ***Jpanel*** (cf. Classe ***AffichagePlateau***).
Les actions comme les appuis sur des touches de clavier sont gérés avec une implémentation de l'interface ***KeyListener***.
• Hérite de JFrame

-> Classe **AffichagePlateau :**

• Cette classe est indispensable pour dessiner les objets nécessaires : pions (joueurs), grille, briques et le curseur de l'utilisateur, qui se mettent à jour grâce à la méthode "*repaint()*" qui sera utilisé en boucle dans la classe principale du jeu.
• Hérite de ***Jpanel***.

-> Classe **TailleFenetre :**

• Contient 2 variables statiques pour définissant la largeur et hauteur de la fenêtre affichant le plateau.
• Il y a également des getters et setters pour obtenir ou modifier la taille de la fenêtre.

-> Classe **Joueur :**

• Nécessaire pour la création des joueurs, cependant pour créer un joueur, il sera nécessaire d'instancier deux classes filles parmi :
-JoueurReel : Pour les 2 joueurs du mode "Joueur Vs Joueur" ou le 1er joueur du mode "Joueur Vs Ordinateur".
-JoueurOrdi : Pour le 2eme joueur du mode "Joueur Vs Ordinateur" uniquement.

-> Classe **JoueurReel :**

• Permet de créer un joueur "réel" qui peut être contrôlé par les utilisateurs du programme.
• Hérite de la classe ***Joueur***

-> Classe **JoueurOrdi :**

• Permet de créer un joueur contrôlé par l'ordinateur.
• Hérite de la classe ***Joueur***

-> Classe **Curseur :**

• Permet de créer 1 seul curseur qui s'affichera à l'écran et qui permettra de sélectionner une case, auquel l'utilisateur du programme pourra déplacer les joueurs ou poser des briques.

-> Classe **Brique :**

• Permet de créer une brique à partir de coordonnés X et Y

-> Classe **ListeBriques :**

• Permet de stocker une liste de briques dans un Vecteur et aussi un tableau de booléen.

-> Classe **Global :**

• Elle contient beaucoup d'attributs et méthodes dédiées au fonctionnement du jeu en lui-même : Analyser la position des joueurs selon les briques, l'affichage des résultats lors de la victoire d'un joueurs, le posage d'un élément (pion ou brique) et bien plus. La quasi totalité des attributs sont statiques afin qu'ils soient accessibles facilement par les autres classes.

-> Classe **Isola :**

• Elle sert juste a redessiner le plateau toute les 10 millisecondes (0,01 secondes) et contient la fonction "main" qui est indispensable au fonctionnement du programme.

3) Relations entre les classes :

• On va déjà commencer avec les classes d'affichage graphiques : Elles dépendent de classes mères venantes de AWT ou Swing :
-FenetrePlateau et FenetreMenu viennent de Jframe.
-AffichagePlateau vient de Jpanel
-FenetrePlateau à un lien avec l'interface KeyListener afin de gérer les actions lors des appuis sur des touches

• Les classes FenetrePlateau et AffichagePlateau dépendent de la classe TailleFenetre, puisque ces classes récupère la largeur et hauteur de la fenêtre (définis dans TailleFenetre) pour pouvoir créer la fenêtre.

• Toutes les classes ont des liens avec Global, puisqu'elle contient des attributs statiques qui peuvent être mises à jour facilement et des méthodes statiques utilisables partout.

• La classe AffichagePlateau fait appel à plusieurs instances de diverses classes, par exemple Brique pour pouvoir dessiner des briques.

• La classe Joueur possède 2 classes filles : JoueurReel et JoueurOrdi.
La classe Joueur est une classe mère abstraite qui contient tous les attributs et méthodes nécessaires pour la création de joueurs (JoueurReel et JoueurOrdi).
Leur classes filles ont aussi leur propre constructeur qui sont chacun différent.

• La classe Brique a un lien avec la classe Curseur puisqu'il est nécessaire de récupérer les coordonnés du curseur pour créer une brique.

• La classe ListeBriques nécessite des briques (Brique) obligatoirement pour que les vecteurs de briques fonctionnent.

• Dans la boucle main de la classe Isola, la classe indispensable pour faire fonctionner le programme regroupe toutes les classes cités au dessus, dont les instances créées via les différentes méthodes/attributs.

4) Fonctionnement global :

Comme décrit au dessus, le but est de reproduire Isola, cependant on peut expliquer étape par étape comment il fonctionne.

• Lancement du programme : Il suffit de lancer le ***Isola.jar***
-Une fois lancé, plusieurs boîtes de dialogues vont s'ouvrir, la première vous demandera de choisir le mode de jeu parmi "Joueur Vs Ordinateur" (Seul 1 joueur peut être contrôlé par l'utilisateur, et le joueur 2 est contrôlé par l'ordinateur) ou "Joueur Vs Joueur" (L'utilisateur/Les utilisateurs peuvent jouer avec les 2 joueurs à tour de rôle)
-Les autres boîtes de dialogue vont vous demander de rentrer des noms pour chaque joueur (sauf le joueur géré par l'ordinateur, si le mode "Joueur Vs Ordinateur" est choisi)
Les boîtes de dialogues se créent grâce à la méthode MenuChoix() (dans la classe FenetreMenu)

• **Lancement de la partie :** le plateau de jeu s'ouvre, 2 joueurs (le joueur 1 est bleu, et le joueur 2 est rouge) vont apparaître dont le premier tout à gauche et le deuxième tout à droite. Il y a aussi une bordure remplie de briques (épaisseur de 1 case) auquel les joueurs ne pourront évidemment pas y accéder. Il y a aussi le curseur jaune placé au milieu
Des instances de Briques, Joueur et Curseur sont déclarés et initialisés dans le diverses classes de Global. Exemple : lancementJeu() permet justement de créer une instance de FenetreMenu et FenetrePlateau et les afficher, puis de créer les joueurs une fois leur noms et le mode de jeu choisi.

• **Fonctionnement du jeu avec mode "Joueur Vs Joueur"** : Le principe est le même qu'Isola, cependant si vous avez chosi le mode "Joueur Vs Joueur", les joueurs se jouent à tour de rôle, avec chaque tour, 2 actions à faire : se déplacer et poser une brique (une fois posé, une instance de Brique est créée aux coordonnés du curseur). Le curseur permet justement de naviguer dans le plateau (plus d'infos sur comment jouer dans la partie IV. Mode d'emploi).
Pour effectuer une action (déplacement ou posage briques), vous devrez appuyer sur la touche "***Entrée***". Cela va créer un cycle :
-Joueur 1 action 1 (déplacement)
-Joueur 1 action 2 (posage brique)
-Joueur 2 action 1
-Joueur 2 action 2
-Joueur 1 action 1 (Recommence)
...

• **Fonctionnement du jeu avec mode "Joueur Vs Ordinateur"** : Si vous avez choisi le mode "Joueur Vs Ordinateur", vous devrez jouer à la place du joueur 1 (en bleu), puis le joueur 2 jouera automatiquement grâce à la méthode *OrdiActionAleatoire()* (dans *Global*), et ainsi de suite. Le joueur 2 n'est pas affecté par l'utilisation du curseur, justement il se déplace par lui-même, et il pose une brique sur une case aléatoire du plateau, à part une case avec une brique déjà existante ou une case contenant l'un des 2 joueurs.
Attention cependant, en appuyant sur la touche "***Entrée***" vous allez jouer ce cycle :
-Joueur 1 action 1
Joueur 1 action 2
-Passer le joueur 2 (laissez le faire, c'est l'ordinateur)

-Joueur 1 action 1
...

• **Condition de victoire** : Grâce à la méthode *VerificationVictoireJoueurs(Joueur joueur_qui_joue, Joueur joueur_en_attente)*, on peut définir le nombre de chemins possibles pour chaque joueur, et si un joueur à 0 chemins accessible (bloqué par des briques ou l'autre joueur), alors le programme donne la victoire à l'autre joueur et accumule les scores selon qui gagne.

• **Affichage des résultats et choix après une partie** : Grâce à la méthode *VerificationVictoireJoueurs(Joueur joueur_qui_joue, Joueur joueur_en_attente)*, on recrée une instance de FenetreMenu, et on affiche cependant l'autre menu qui celui ci affiche le joueur qui a gagné, le score de chacun et propose au joueur soit de : "Rejouer", "Changer Mode" ou "Quitter".
-"Rejouer" permet de relancer une partie du même mode de jeu.
-"Changer Mode" permet de revenir au menu principal (l'autre lancé dès le lancement du programme) et repropose de choisir le mode de jeu à l'utilisateur.
-"Quitter" ferme le programme.

• **Fonctionnement du curseur** : Il en existe un seul dans le programme et il permet de naviguer entre les cases. Pour se déplacer d'une case à l'autre, vous devrez utiliser les touches ► ▲ ◄ ▼ du pavé directionnel de votre clavier, qui permettent doit de déplacer le curseur d'une case à droite, en haut... Ensuite comme expliqué au dessus, il est possible d'effectuer une action comme déplacer un joueur (mais faut que l'écart entre coordonnés X du joueur et du curseur soit compris entre -1 et 1, et l'écart entre coordonnés Y du joueur et du curseur soit compris entre -1 et 1, sinon on respecte pas les règles du jeu !), puis poser une brique, il n'y a pas de restriction au niveau du lacement dans la brique à part qu'on ne puisse pas poser une brique sur un joueur ou une brique déjà existante). Le curseur ne peut pas être contrôlé pendant que le joueur 2 joue.

III. Réalisation

1) Choix techniques :

Nous allons parler des choix techniques dans différents points.

• Les types de joueurs :
-Pour respecter le sujet à réaliser, il est demandé de faire 2 modes de jeu : "Joueur Contre Joueur" et "Joueur Contre Ordinateur", il est donc utile de créer 2 sous-classes héritant de Joueur, (qui sont *JoueurReel* et *JoueurOrdi*), pouvant avoir un comportement différent à travers différentes méthodes et attributs propres à chaque type de joueur. Même si finalement j'ai très peu ajouté d'attributs et méthodes propres à chaque type de joueur (la classe mère Joueur avait tout ce dont elle avait besoin), je pense qu'il est préférable de distinguer ces 2 types de joueurs par des clases différentes (au lieu de tout mettre dans la classe *Joueur)*, surtout pour d'éventuelles futures fonctionnalités à ajouter dans le futur (au cas où je garde ce programme et que je compte l'améliorer ou le publier en ligne).

• Mode de jeu "Joueur Vs Joueur" :
-Concernant ce mode de jeu, j'ai implémenter un système de fonctionnement simple avec 2 actions par joueur, et les joueurs qui jouent à tour de rôle.
-Le désavantage que j'ai pas pu prendre en compte le fait que ce soit toujours le Joueur 1 qui commence, à la place j'aurais pu faire en sorte qu'il y a 50% de chances que ce soit le joueur 1 qui commence, et 50% le joueur 2 afin qu'il y ait un équilibre

• Mode de jeu "Joueur Vs Ordinateur" :
A propos de ce mode de jeu, on peut parler de différents points :
-Pourquoi doit-on taper sur la touche "Entrée" pour faire jouer le joueur 2, alors qu'il pourrait être joué directement après la dernière action du joueur 1 ? Cela est du à l'algorithme que j'ai utilisé, qui oblige à taper sur "Entrée" pour lancer la méthode Global.ActionJoueur() , cette méthode concerne autant les 2 joueurs (Humains ou ordinateur) mais lorsque c'est au tour de l'ordinateur, cette méthode exécute une autre méthode *OrdiActionAleatoire()* qui fait exécuter les 2 actions (déplacement et posage brique) en une seule fois.

• Système de score : Un choix pour rajouter de la compétitivité entre les joueurs et je pense que c'est un bon choix. Facile à implémenter grâce à des attributs statiques et la méthode qui vérifie si un joueur a gagné : *VerificationVictoireJoueurs(Joueur joueur_qui_joue, Joueur joueur_en_attente)*.

• Attributs/Méthodes publiques et statiques dans les classes ***Global*** et ***ListeBriques*** :
Même si on ne respecte pas la règle d'encapsulation avec ces attributs, d'ailleurs cela est du en partie à un manque de temps pour faire une meilleur implémentation, le fait que ces attributs et méthodes soient statiques est un avantage pour le fait qu'on peut les déclarer facilement dans toutes les classes du programme en faisant par un exemple un ***Global***.*MonAttribut* ou ***Global***.*MaMethode*().

• Choix d'un curseur plutôt que cliquer directement sur des cases à la souris : j'ai eu plus de facilités à implémenter un système de curseur de cette manière, en plus le fait qu'il y ait un curseur permet de montrer ce que tu sélectionnes.

• Choix des menus : Une série de plusieurs petits menus (plutôt des boites de dialogues) à la suite me semble être plus clair et indiquer étape par étape ce qu'il faut faire aux utilisateurs.

• Choix des couleurs : Mieux distinguer les éléments, surtout les joueurs.

• Ajout d'une classe TailleFenetre : Pourrait être un moyen simple et utile pour pouvoir modifier la taille des fenêtres si jamais on pourrait implémenter un système de responsive design dans notre programme, de manière à créer une fenêtre de 300x300 pixels ou 1200x1200 pixels selon l'appareil utilisé.
En plus de cela, la taille des cases, de la grille, des joueurs, des briques et du curseur

s'adapte facilement à la taille de la fenêtre, vu que les valeurs (taille x, taille y) sont obtenues grâce à des calculs à partir de la taille de la fenêtre ou de constantes.

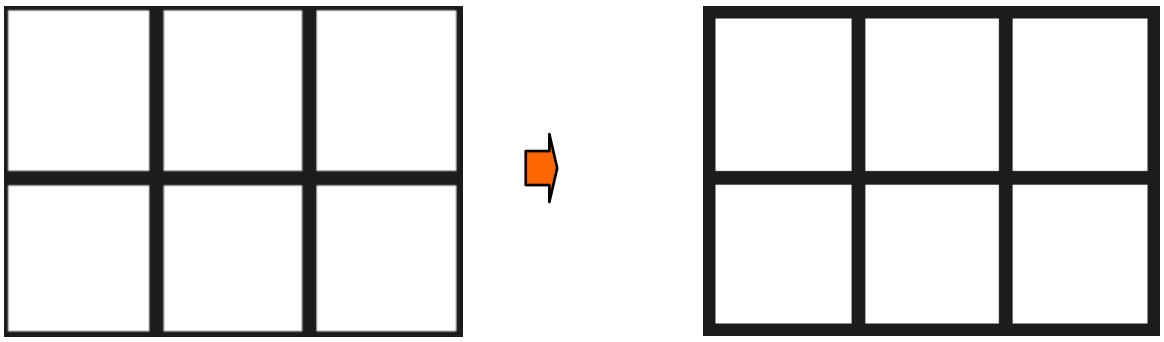
• L'utilisation de bibliothèques d'affichage graphique comme AWT ou Swing semble être un très bon choix puisqu'elles sont simple d'utilisation et complète.

2) Présentation des algorithmes complexes :

• Explication des actions aléatoires faites par les joueurs :
-Géré par la méthode OrdiActionAleatoire().
-Dans un premier temps, elle vas faire déplacer le joueur de manière aléatoire (*Math.random()*) avec des coordonnés (avec un écart entre la position de destination et actuelle qui respecte : **X - 1 <= position joueur X <= X + 1** et **Y - 1 <= position joueur Y <= Y + 1**) mais en vérifiant aussi si l'un des 9 choix possible contient des briques, auquel on se sert du tableau booléen *case_avec_brique[y][x]* et si il correspond à "true", alors il y a la présence d'une brique, donc le joueur pourra pas se déplacer à une case contenant une brique, cela se fait aussi en vérifiant la position de l'autre joueur pour pas que les 2 joueurs soit superposés.
-Dans un second temps, j'ai fait en sorte que le joueur 2 place une brique sur une case aléatoire mais en respectant les conditions : Une brique ne doit en aucun cas être posé sur un joueur ou une autre brique. Cette implémentation est facile grâce au tableau case_avec_brique[][] et case_avec_joueur[][] vérifiant si il y a des briques ou des joueurs et en respectant **X - 1 <= position joueur X <= X + 1** et **Y - 1 <= position joueur Y <= Y + 1**.

• Vérifier si un joueur est totalement bloqué :
-Utilisation de la méthode *VerificationVictoireJoueurs(Joueur joueur_qui_joue, Joueur joueur_en_attente)* (classe ***Global***).
-On déclare 2 fois cette méthode dans la méthode d'exécution du jeu (*PartieEnCours()*) avec pour chacune les joueurs qui diffèrent de place.
-On vérifie si un joueur est bloqué par des briques grâce à 2 boucles itératives qui parcourent toutes les cases autour du joueur (8 cases + celle du milieu où est placé notre joueur), et on vérifie (pour **X - 1 <= position joueur X <= X + 1** et **Y - 1 <= position joueur Y <= Y + 1**) si pour chaque case il y a *case_avec_brique[][]* qui correspondent à "true" et aussi *case_avec_joueur[][]* qui correspond à "true", et pour chaque case contenant une brique ou un joueur (soi-même aussi), on incrémente *chemins_bloques* de 1, et si la variable cases_*bloques* est égale à 9, alors c'est que le joueur est enfermé. Cet algorithme est plutôt simple
-Si un des joueur a atteint les 9 cases bloqués, alors l'autre joueur va gagner tout simplement en lui affectant la méthode *Gagne()*. Le score du joueur qui a gagné est incrémenté de 1 également.

• Ajustement des éléments (grille, joueur...) :
-Pour que l'épaisseur de la grille paraisse être la même partout, pour pas que la moitié de la grille soit caché sur les bordures de l'écran.



-Cela peut se faire en récupérant la taille de la hauteur et largeur de la fenêtre et avec diverses adaptations.
-On peut faire pareil maintenant avec les joueurs, le curseur et les briques pour mieux les aligner dans une case.

• Affichage de la série de boîte de dialogue au lancement du programme :
-On initialise tout les panels d'options (JOptionPane) et aussi des tableaux de chaines de caractères qui représente tout une série de choix, à ajouter dans une boîte de dialogue (représentés sous forme de boutons par la suite).
-Grâce aux attributs créés avec JoptionPane, il est possible de faire afficher des boîtes de dialogues, une par JoptionPane, puis récupérer tous les choix faits par l'utilisateur. Selon les choix, on peut créer des conditions permettant de rediriger l'utilisateur à chaque fois selon ses choix.

• Affichage du menu de victoire (boîte de dialogue) :
-Reprend le même fonctionnement que pour les boîtes de dialogues du dessus.

3) Formats de fichier de données utilisés :

• Ce logiciel n'utilise pas de fichiers pour stocker des données (Joueurs : noms, scores...) ou de base de données. Pour l'instant pas vraiment d'intérêt.

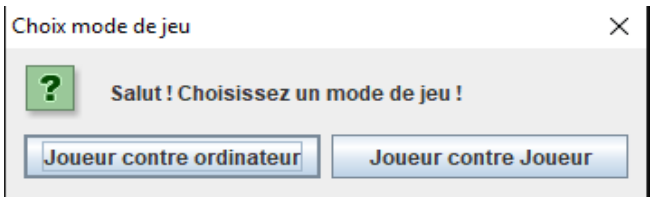
IV. Utilisation

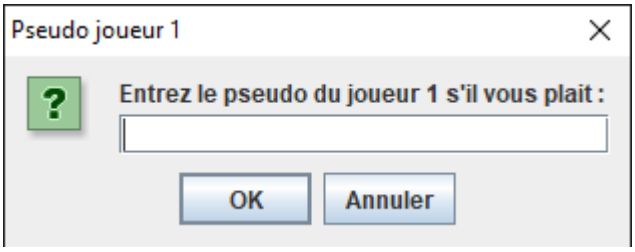
1) Mode d'emploi :

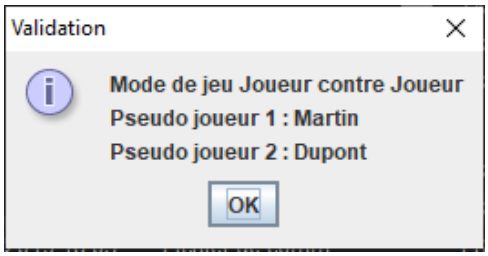
Le fonctionnement du programme avec de l'aide pour l'utilisation a été expliqué dans la partie II 4). Mais on peut faire un récapitulatif ici, avec quelques illustrations

1) Lancement du logiciel **Isola.jar**

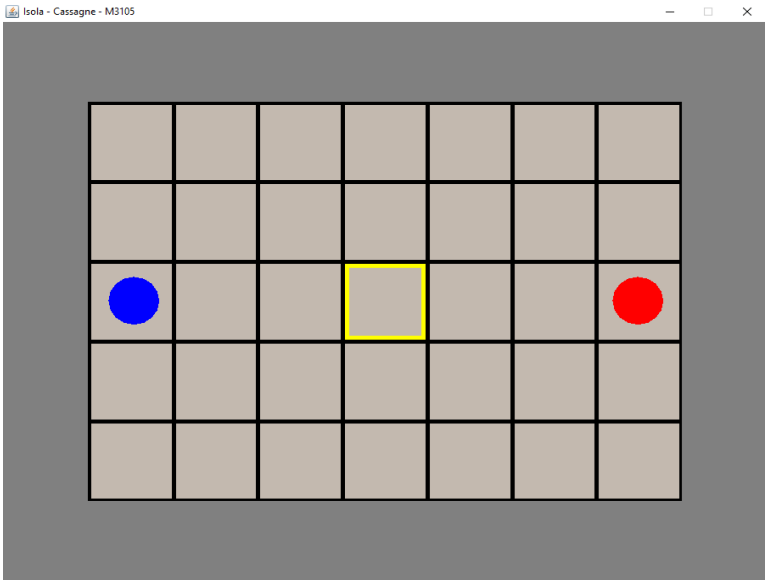
2) On arrive sur une série de boîtes de dialogues, vous devrez faire vos choix :







3) Après vous arrivez sur le plateau de jeu



4) Les commandes de jeu sont simples :
-Pour naviguer entre les cases, utilisez les touches du pavé directionne ► ◄ ▲ ▼
-Pour effectuer une action comme faire déplacer le joueur ou poser une brique à la case que vous sélectionnez, appuyez sur la touche "Entrée".
-Si vous connaissez les règles du jeu Isola (expliqués au début du rapport technique) et que vous savez si vos actions sont valides ou non (exemple : vouloir déplacer un joueur sur une brique), il n'y a pas de problèmes.

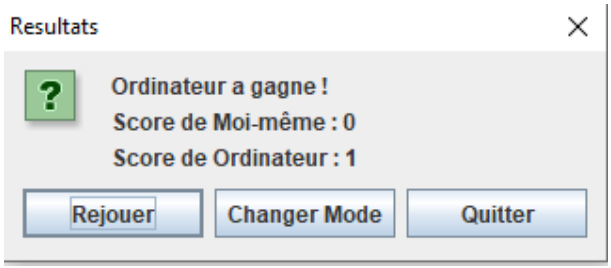
5) Cycle de jeu si le mode "Joueur Vs Joueur" est choisi:

-Joueur 1 étape 1
-Joueur 1 étape 2
-Joueur 2 étape 1
-Joueur 2 étape 2
-Et ça recommence

6) Cycle de jeu si le mode "Joueur Vs Ordinateur" est choisi:
-Joueur 1 étape 1
-Joueur 1 étape 2
-Joueur 2 étape 1 et 2 (vous ne jouez pas !)
-Et ça recommence

Il faut appuyer sur "Espace" pour passer d'une étape à l'autre tout simplement.

7) Affichage des résultats et proposition de continuer à jouer, changer de mode de jeu ou autre :



2) Configuration requise :

• **Utilisation d'un PC obligatoire ! Sous Windows, Mac ou Linux.**

• **Une résolution d'affichage d'au moins 700x700 pixels !**

• Les performances de votre PC importent peu.

3) Lien vers la documentation créé :

Pour créer la documentation (si non existante), vous pouvez utiliser la commande :
javadoc -d doc *.java

Excusez-nous pour les erreurs présentes ou warnings, mais la documentation HTML reste fonctionnelle à 95 %, il manque juste des liens vers les autres classes (balises *@see*) et sinon c'est bon.

Sinon un fichier en .zip qui contient toute la documentation générée sera envoyé à vous par mail.

V. Conclusion

1) Bilan :

Il s'agit d'un programme plutôt fonctionnel, malgré quelques bugs dont certains bugs qui font par exemple qu'un joueur perd si il n'est pas totalement coincé, et le mode "Joueur Vs Ordinateur" reste dans l'ensemble assez buggé mais cela ne vous empêchera pas d'enchainer plusieurs parties sans trop de problèmes.

2) Optimisation possibles et extensions possibles :

• Correction des bugs présentés au dessus.

• Faire en sorte que le jouer Ordinateur puisse placer des briques plus intelligemment de manière à coincer au maximum son adversaire plutôt que poser des briques aléatoirement.

• Mieux organiser les classes, principalement en mettant tous les attibuts privés (ce qui n'est pas le cas pour *Global* et *ListeBriques*) et mettre en place des getters et setters pour y accéder à ces attributs.

• Peut-être ajouter une option pour choisir le mode de difficulté de l'adversaire (mode ordinateur), et la difficulté dépendra de l'"intelligence" de l'adversaire à piéger le joueur à se déplacer.

3) Autres :

• Nous sommes désolé pour les bugs que vous pouvez rencontrer, surtout en mode Ordinateur, je n'ai pas réussi à les corrigerr tous, mais j'ai réussi à déjà en corriger pas mal depuis l'implémentation du programme.