Le Mans Université

Licence Informatique *3ème année*

Module Génie Logiciel 2

Système d'aide à la résolution de puzzle : HASHI

Cahier d'Analyse et de Conception

Groupe 4

Aaron AMANI, Axel JOURRY, Cindy CALVADOS, Clément JANVIER, Collins Soares, Florian DREUX, Rayyan LAJNEF, Thomas MALABRY, Willhem LIBAN

Table des matières

1. Classement

2. ListeNiveau

3. Sauvegarde

4. Ressources

VII – Tests unitaires

I - Rappel du Cahier des charges

Ce projet, réalisé dans le cadre d'un TP, doit être rendu sous la forme d'une image virtualBox. Réalisé suite à un module de Programmation Orientée Objet, le logiciel doit être codé en Ruby. Les notions assimilées lors du module devront également être mises en œuvre.

Les fonctionnalités suivantes doivent être disponibles :

- Gestion du chronomètre, de l'annulation des coups et de la mise en pause du jeu
- Enregistrement et chargement d'une partie
- Proposition d'un menu principal
- Proposition d'une aide lors du jeu
- Gestion d'un mode Hypothèse
- Gestion d'un classement
- Gestion des interactions grille-utilisateur

Le logiciel final se compose donc de nombreuses classes dont la description se trouve ci-après.

II - Les classes : le jeu

1 – Ile

La classe Île est une classe héritant de Gtk::Button, elle permet de représenter les îles présentes sur le plateau. Cette classe contient plusieurs variables d'instances (nous écrirons VI par la suite pour faciliter la lecture) disponible en lecture et écriture grâce au coding assistant. La méthode new initialise l'île comme suit :

- on initialise les VI définissant l'île : @gridRef, @degreeMax, @degree, @row, @column et @estComplet.

- on initialise les éléments entourant l'île : @northNode, @southNode, @eastNode et @westNode qui contiendront les références des voisins ; ainsi que @northEdge, @southEdge, @eastEdge et @westEdge qui représentent les statuts des ponts partant de l'île (0 = pas de pont, 1 = pont simple, 2 = double pont)

- on met à jour l'apparence de l'image : on utilise ici le fait que l'île soit un Gtk::Button pour lui attribuer une image de fond et un relief grâce au css et à Glade, et enfin une fonction à exécuter lors d'un clic.

La classe propose différentes méthode utiles pour prendre connaissance du plateau :

- getVoisins : retourne un tableau de références des voisins de l'île.

- pontRestants : retourne le nombre de ponts reliés à l'île

- pontAvecVoisins : vérifie si une île est reliée à tous ses voisins.

- compterVoisins : retourne le nombre de voisins d'une île.

- voisinsNonComplet : retourne un tableau de référence des voisins d'une île qui ne sont pas complets.

- compterVoisinsNonComplet : retourne le nombre de voisins non complets d'une île.

Elle propose ensuite des méthodes permettant la modification de l'île

- inc, dec et set_degree permettant respectivement d'incrémenter, de décrémenter et de set la valeur de @degree, représentant le nombre de ponts reliés à l'île

- update : met à jour l'île, l'affichant en rouge si trop de ponts y sont reliés, en vert si il a le bon nombre, en blanc sinon

- choixPossible : méthode permettant de mettre en évidence les îles voisines de l'instance, on modifie alors l'image de l'île pour passer en « _s », pour surbrillance

-enleverChoixPossible : permet de repasser l'île en normal, l'image utilisée n'a alors plus « _s »

2 – Pont

La classe Pont hérite de la classe Île vue dans la partie précédente. Elle permet de représenter les ponts qui relient les îles du plateau.

On retrouve trois variables d'instance :

- @typePont, qui correspond à sa représentation (0 = inexistant, 1 = pont simple, 2 = pont double)
- @directionPont (0 = inexistant, 1 = pont horizontal, 2 = pont vertical)
- @estDouble, un booléen indiquant si le pont est doublé ou non.

La méthode new initialise un Pont vide, qui n'a ni direction et un type à 0 car il n'a pas encore été créé.

La classe propose des méthodes d'accès aux VI :

- get_directionPont et set_directionPont permettent respectivement d'obtenir et de modifier la direction du Pont.

- get_typePont et set_typePont permettent respectivement d'obtenir et de modifier le type du Pont.

La classe propose également une méthode de mise à jour, update, qui permet de mettre à jour l'affichage du Pont sur le plateau.

3 – Sauvegarde

La classe sauvegarde permet l'enregistrement des coups du joueur sur le plateau.

Une première méthode, initialise, permet de créer les variables nécessaires à la sauvegarde, tel que

- @nomniv, contenant le nom du niveau à sauvegarder
- @dif, la difficulté du niveau afin de sauvegarder ou charger dans le répertoire correspondant
- @path, le chemin contenant la sauvegarde, fabriqué à l'aide des deux VI vues au dessus.

Ensuite, la classe propose une seconde méthode, *sauvegarder*, permettant comme son nom l'indique, de sauvegarder le plateau à chaque coup que le joueur effectue. Celle-ci créée elle-même le fichier de sauvegarde correspondant au niveau. Ces fichiers sont enregistrés au format .txt selon le dictionnaire suivant :

- '0' : case vide

- entier entre 1 et 8 : île correspondante

    - ‘-’ : simple pont horizontal

    - ‘I’ : simple pont vertical

    - ‘=’: double pont horizontal

    - ‘H’ : double pont horizontal

La dernière ligne de ce fichier txt représente le temps que le joueur a déjà passé sur le niveau.

La troisième méthode, *charge*, permet de charger les ponts et le chronomètre d’un niveau déjà commencé contenu dans un fichier txt, pour pouvoir reprendre une partie en cours.

Enfin, l’avant dernière méthode de classe, *saveTime*, à pour but d'écrire dans un fichier le temps et le nom de la personne ayant effectué ce temps, séparé par ‘:’. Il existe un fichier par niveau, comportant absolument tous les temps effectués sur ce niveau. Ces derniers ont pour objectif d’être lus pour pouvoir afficher le classement.

Pour finir, *saveDelete*, comme son nom l’indique, prend le nom d’un niveau en paramètre et sa difficulté afin d’en supprimer son fichier de sauvegarde. Elle est appelée quand le joueur termine un niveau, afin qu’il ne puisse plus recharger sa sauvegarde.

4 – Pause

La classe Pause, est la classe gérant le menu pause lors de la partie de Hashi. La classe ne comporte qu’une seule méthode d’initialisation. Elle a pour rôle de créer la fenêtre, dans laquelle elle place trois boutons contenant trois fonctionnalités :

- Reprendre : reprend la partie en remettant le chronomètre en route.

    - Recommencer : recommence la partie et le chronomètre via la méthode resetPlateau de la classe Plateau, qui sera vu en sous partie 8.

- Quitter : ferme la fenêtre en ouvrant une fenêtre de menu principal

5 – Hypothèse

La classe Hypothèse gère la fonctionnalité « hypothèse », qui fournit à l’utilisateur une copie de la grille pour qu’il puisse essayer des combinaisons de liaisons entre îles.

La méthode initialize recréé une copie de la grille passée en paramètre et l’affiche dans une nouvelle fenêtre. La modification principale par rapport à la classe Plateau (expliquée en sous partie 8) est l’ajout deux deux boutons annuler et quitter. Le premier a pour rôle de fermer la fenêtre d’hypothèse et d’annuler les actions du joueur, tandis que la seconde retourne la grille de l’hypothèse et appelle la méthode hypotheseValider du Plateau.

6 – HashiGrid

La classe HashiGrid, héritant de Gtk::Grid, représente la grille du plateau de jeu. Cette classe permet de gérer l'ensemble des actions se déroulant sur le plateau de jeu. Cette dernière fait donc appel aux classes vues précédemment afin de gérer les objets présents sur le plateau.

Elle possède plusieurs VI, qui sont toutes initialisées dans la méthode initialize, tel que :

- @diff : la difficulté du niveau
- @nomniv : le nom du niveau
- @x : le nombre de colonnes de la grille
- @y : le nombre de lignes de la grille
- @undoRedo : une instance de classe de UndoRedo
- @nb_iles : un compteur contenant le nombre d'îles contenu dans le niveau
- @sommets : une liste contenant tous les boutons dans lesquels sont contenus les îles
- @nodeLink : un tableau contenant temporairement les deux îles sélectionnées dans le but de réaliser des actions entre elles.
- @saveManager : une instance de la classe sauvegarde.

Elle propose donc des méthode de modification du plateau de jeu et de ce qu'il s'y passe :

- ajoutePont : méthode prenant 2 îles en paramètres, elle ajoute un Pont entre ces îles après avoir vérifié que ce Pont était valide. Si un pont est déjà présent, cela le transforme en double pont

- supprimePont : méthode prenant 2 îles en paramètres, elle supprime le Pont entre ces îles, si celui-ci existe.

- notify : méthode permettant de prévenir la grille de la case actuellement sélectionnée.

- chargeGrille : méthode qui, lorsque la classe sauvegarde emploie sa méthode de chargement, charge la grille contenue dans le fichier txt.

La classe HashiGrid propose ensuite des méthode de lecture de la grille :

- ajoutValid? : méthode vérifiant si l'ajout d'un pont entre deux îles passées en paramètres est valide, c'est-à-dire que les îles sont bien voisines et qu'aucun pont n'entrave la liaison entre ces îles

hargeVoisins : méthode chargeant les voisins accessibles d'une île

- getPontEntre : méthode retournant le tableau des cases du pont entre 2 îles passées en paramètres

- grilleFini? : méthode permettant de vérifier que toutes les îles possèdent leur nombre nécessaire de ponts.

La classe utilise ensuite la méthode handleClick pour gérer toutes les actions liées au passage de la souris sur les îles : ajout ou suppression d'un pont et récupération des cases cliquées conservées dans la classe UndoRedo.

De plus, elle possède deux méthodes undoPrevious et redoPrevious permettant d'activer les undo et redo contenus dans la classe UndoRedo.

7 – UndoRedo

La classe UndoRedo a pour but de gérer les fonctionnalités relatives à ces deux actions.

La classe contient 2 variables de classes, qui sont 2 tableaux, un stockant les coups joués(@undoStack) et l'autre stockant les « undo » (permettant ainsi les « redo », dans @redoStack).

Les méthodes proposées par la classe sont donc :

- cleanAll : méthode permettant de supprimer le contenu des deux tableaux
- saveUserClick : méthode permettant de stocker l'action du joueur dans le tableau d'undo.
- undoUserAction : méthode permettant de supprimer la dernière action du joueur du tableau et de le renvoyer au jeu (dans le but d'effectuer un redo).
- undoEmpty? : méthode permettant de vérifier si la pile est vide.

8 – Plateau

La classe Plateau propose des accès en lecture et en écriture à ses VI via le coding assistant.

La méthode new initalise le plateau de jeu ainsi :

- création de la fenêtre de jeu, à taille fixe, immobile centrée au milieu de l'écran

- création et ajout des boutons d'interface : pause, indice, undo, redo, hypothèse

- création et initialisation de la HashiGrid qui va gérer tous les événements du niveau

- initialisation et lancement du chronomètre.

La classe aide (qui gère les indices) est expliquée en III. Les classes Pause, Sauvegarde (pour le undo et redo) et Hypothèse sont expliquées plus tôt en I.

La classe propose également des méthodes agissant sur la grid du jeu :

- hypothèseValider : prend en paramètre la grille utilisée pour l'hypothèse et remplace la grille du jeu par celle ci.

- resetPlateau : Arrête le chronomètre, détruit le plateau et la grille actuelle et crée un nouveau plateau.

- lancerChrono : Méthode lançant un second thread en parallèle du plateau de jeu pour pouvoir faire fonctionner le chronomètre.

- partiFini : Méthode arrêtant le chronomètre et permettant de faire apparaître un écran par dessus le plateau lorsque le joueur termine la partie. Ce menu permet de de sauvegarder son temps dans le fichier de sauvegarde en entrant un nom pour le joueur, puis de recommencer la partie ou bien de revenir au menu principal.

III - Les classes : l’interface

1- Client

La classe Client fait office de main. C’est elle qu’il faut lancer afin de démarrer le programme. Elle a pour simple rôle de créer une instance de MainMenu que nous verrons dans la sous partie suivante et de l’afficher.

2- MenuLayout

La classe MenuLayout permet de créer une disposition de base pour la création des menus du jeu. Cela évite la duplication du code puisque tous les menus que nous verrons ci-après seront dépendant de cette classe.

La classe contient énormément de méthodes d’instances:

- @builder : un gtk builder permettant de créer la disposition de base
- @menu : la fenêtre du menu à afficher, qui viendra se compléter grâce aux classes ci-après
- @css_file : une variable contenant un fichier css qui permet de mettre en place le style des menus

La classe contient aussi des variables @btn_facile, @btn_moyen, @btn_difficile et @btn_retour qui serviront dans la sélection de niveau et dans le classement pour choisir la difficulté et revenir au menu principal.

Elle contient ensuite un @tableauBtn contenant tous les boutons de niveaux qui apparaîtront dans les classements et dans la sélection de niveau.

Enfin, elle contient un @tabfacile, @tabdimfacile ainsi que 2 VI similaires pour les niveaux normaux et deux autres pour les niveaux difficiles.
Les tab contiennent les chemins vers les fichiers permettant de charger les niveaux.
Les tabdim, eux, contiennent les tailles x et y permettant d’initialiser la grille de jeu à la bonne taille.

3- MainMenu

La classe gérant le menu principal est la classe MainMenu. Celle-ci ne produit que des effets de bords, sur la fenêtre en elle-même. Ainsi, la classe génère tout simplement le menu de sélection principal. Elle place tous les Gtk::Button de sélection sur une Gtk::Table afin de disposer les boutons proprement sur la fenêtre. Ceux-ci lancent les classes gérant les différentes fonctionnalités proposées par le système.

Ainsi, les différents boutons sont les suivants :

- btnSelection, qui ouvre le menu de sélection de niveau

- btnTuto, qui ouvre le tutoriel du jeu

- btnClassement, qui ouvre la fenêtre de classement des joueurs

- btnQuitter, qui ferme la fenêtre afin de quitter le jeu

Détaillons maintenant ces fonctionnalités et leurs implémentations.

4 – MenuSelection

Le menu de sélection ne propose pas de méthode de classe mais génère la fenêtre du menu de sélection de niveaux. La sélection se fait comme suit :

- on choisit la difficulté

- on choisit le niveau

Ainsi, une fois sur le menu de sélection, le joueur utilise une des box initialisées par Glade afin de déterminer s' il souhaite jouer à un niveau facile, normal et difficile. Les boutons se situant sur cette box modifient la fenêtre de sélection de difficulté. Cela met à jour les boutons pour qu'ils chargent les niveaux correspondants à la difficulté sélectionnée. Juste avant de lancer le niveau, les boutons ouvrent une fenêtre pop-up demandant à l'utilisateur s'il souhaite charger une partie en cours ou la recommencer. Si il n'y a pas de partie a charger, le bouton ne fera tout simplement rien.

Une fois le niveau sélectionné, les classes concernant le Jeu (expliquées dans la partie I) sont appelées.

5 – BoutonChoix

La classe BoutonChoix, représente les boutons reliant un niveau au menu de sélection. Cette classe utilise ces VI, via des méthodes getter, pour savoir si un bouton doit ou non s'afficher dans le MenuSelection. Les getters sont donc les suivants :

- isFacile? : retourne true si le niveau est un niveau facile

- isMoyen?: retourne true si le niveau est un niveau moyen

- isHard? : retourne true si le niveau est un niveau difficile

- getNiveau : retourne le niveau relié à l'instance du BoutonChoix

- getX : retourne la taille en X du niveau relié à l'instance

- getY : retourne la taille en Y du niveau relié à l'instance

La classe dispose aussi d'un setter nommé setPath, permettant de changer le chemin d'un bouton afin d'éviter de disposer de différents boutons réalisant des actions similaires.

Ces méthodes et l'utilisation de cette classe permet un affichage plus aisé dans MenuSelection.

6 – Tuto

La classe Tuto permet l'implémentation d'un tutoriel, accessible via le menu principal. On utilise une variable globale id afin de savoir où l'utilisateur se trouve dans la lecture du tutoriel.

La méthode initialize créer la fenêtre de tutoriel ainsi :

- on crée la fenêtre avec tout ce qui l'a définie (titre, taille ect..)

    - on crée un objet de type GTKImage.

    - on ajoute les boutons flèche gauche et droite qui permettront à l'utilisateur de naviguer dans le tutoriel

- on ajoute le bouton permettant de quitter la fenêtre

Dans cette fenêtre, chaque appuie sur les flèches permet de changer l'image contenu dans l'image GTKImage. Cela permet de n'avoir qu'un seul objet étant actualisé en temps réel.

7 – APropos

La classe APropos, ouvre le menu À Propos. Ce menu indique à l'utilisateur les noms des étudiants ayant travaillé sur le projet. La classe propose une seule interaction à l'utilisateur avec le bouton retour, qui ré-ouvre le Menu Principal.

IV – Les classes : l’aide

1 - Aide

L’aide proposée par le jeu est gérée par la classe du même nom. Cette classe propose différentes méthodes utiles à l’utilisateur ainsi que de nombreuses méthodes permettant de savoir quelle aide sera proposée à l’utilisateur.

Les instances de la classe sont initialisées avec comme seul paramètre la grille. Les autres VI sont les suivantes :

- @id : accessible en lecture grâce au coding assistant

- @nbVoisins : indique le nombre de voisins de chaque sommet de la grille

- @position : indique la position sur laquelle portera l’aide apportée

- @grille : la grille dans laquelle doit opérer l’aide

La classe dispose ensuite d’une méthode lireId, qui permet de mettre à jour l’id de l’aide qui sera apportée via la méthode definirCas. Cette méthode retourne un entier selon l’aide dont le joueur a besoin, les cas sont les suivants :

- 1 : île avec un seul voisin disponible

- 2 : île avec une seule île voisine

- 3 : île numérotée 8 encore vide sur la grille

- 4 : île numérotée 6 avec trois voisines

- 5 : île numérotée 4 avec deux voisines

- 6 : île numérotée 4 avec une voisine numérotée 1

- 7 : île numérotée 5 avec trois voisine dont une à 1 pont restants

- 8 : île numérotée 3 avec deux voisine dont une à 1 pont restants

- 9 : île numérotée 7 encore vide sur la grille

- 10 : île numérotée 5 avec trois voisines encore vide sur la grille

- 11 : île numérotée 3 avec deux voisines restantes sur la grille

- 12 : île numérotée 6 avec deux voisines numérotée 1

- 13 : île numérotée 4 avec deux voisines numérotée 1

- 14 : île numérotée 2 avec deux voisines dont une île numérotée 2

La classe est appelée via la méthode getMessageAide, qui appelle les méthodes précédentes afin de déterminer l’aide la plus utile pour l’utilisateur lors de l’appel. Ensuite, la méthode affiche l’aide correspondante, la table de traduction id/aide se trouve dans le fichier AideTexte.txt .

V – Les classes : le classement

1 - Classement

La classe Classement hérite de MenuLayout que nous avons vu en III. Elle sert d'écran d'affichage de tous les niveaux, triés comme dans le menu de sélection. En revanche, au lieu de lancer les niveaux, cette classe ouvre les fenêtres de classement, via la classe ClassementNiveau.

Il y a 2 VI dans cette classe

- @menu qui contient la fenêtre à afficher
- @builder le builder contenant le layout, récupéré à partir de l'héritage de MenuLayout.

La classe dispose d'un tableau de boutons par difficulté. Chacun de ses boutons est relié vers la classe classementNiveau. Elle dispose aussi d'un bouton quitter et d'un bouton par difficulté afin de charger le tableau nécessaire.

2 - ClassementNiveau

Cette classe hérite de Gtk::Window, et ne propose aucune méthode. La classe crée une fenêtre d'affichage qui est initialisée lors de l'appel évoqué précédemment. Cette classe utilise le paramètre *nomniv* de la méthode initialize pour afficher les classements d'un niveau précis. Nous utilisons également un dossier Classement, qui contient tous les classements des différents niveaux pour pouvoir stocker les classements d'un niveau donné.

VI – Les ressources

1 - Classement

Classement est un dossier contenant 3 sous-dossier, contenant les classements des niveaux correspondant à la difficulté :

- facile

- moyen

- hard

2 - ListeNiveau

ListeNiveau est un dossier comprenant, comme classement, 3 sous-dossiers correspondant aux niveaux de difficulté du jeu. Les niveaux sont ensuite nommés selon leur taille, sous format txt (ex : 10x10 est un niveau de 10 cases de haut et 10 cases de long).

Le format txt représente la grille case par case, 0 représentant une case vide, et un entier entre 1 et 8 représente une île de la valeur correspondante. Toutes les cases sont séparées par ‘:’ pour la lecture du fichier.

3 - Sauvegarde

Le dossier Sauvegarde contient également 3 sous-dossiers correspondant au niveau de difficulté. Ces dossiers contiennent les sauvegardes des niveaux déjà enregistrés. L’enregistrement des niveaux a été expliqué en I.

4 - Ressources

Le dossier ressources contient 6 sous-dossiers, qui possèdent les données suivantes :

- Ile : les pngs des différentes îles possibles, 0 étant une image vide ; entier.png une image de l’île en blanc, les ‘_s’ celles en surbrillance, ‘_v’ celles en vert, ‘_r’ en rouge

- MenuSelection : les pngs des boutons de sélection de la difficultés des niveaux, une étoile se remplissant selon la difficulté, ainsi que le bouton de retour, une flèche noire

- Pause : le png du logo affiché lors de la pause d’un niveau

- Plateau : contient les png des boutons d'interface de jeu (hypothèse, pause, undo, redo, aide)

- Pont : contient les png des différents ponts possibles, nommés de la manière suivante : 'h' ou 'v', respectivement pour horizontal et vertical + 'simple' ou 'double' + '' ou '_v' si ils doivent être en vert

- Tuto : contient les png des images affichées lors du tutoriel, numérotées de 0 à 6

5 - Documentation

Le dossier documentation comporte tous les fichiers générés par le système RDoc, permettant ainsi d'avoir une description détaillée des classes, méthodes et variables de chaque fichier.

5 - Glades

Ces fichiers contiennent les dispositions de chaque fenêtre générées par l'outil Glade, permettant ainsi à chaque élément du jeu d'être disposé de bonne façon. Ces fichiers se servent de ceux contenus dans le dossier CSS (voir sous partie suivante) afin de créer les éléments.

6 - CSS

Ces deux fichiers contiennent les différents styles utilisés par chaque élément du programme. Ils sont lus par les fichiers glades pour former les fenêtres.

VII – Tests unitaires

1 - Menu Principal

Utilisation normale du menu pour vérifier que chaque bouton emmène au menu lui correspondant : les appels de méthode correspondent bien aux boutons cliqués correspondants.

Résultat : Aucun problème.

Essayer de provoquer une anomalie dans le comportement du jeu en jouant avec les différents boutons très rapidement.

Résultat : Aucun problème.

2 - Menu de sélection

Utilisation normale du menu pour vérifier que chaque bouton emmène au menu lui correspondant : les appels de méthode correspondent bien aux boutons cliqués correspondants.

Résultat : Aucun problème.

Essayer de provoquer une anomalie dans le comportement du jeu en jouant avec les boutons de sélection de niveau très rapidement.

Résultat : Aucun problème.

Essayer de provoquer une anomalie dans le comportement du jeu en jouant avec les boutons de sélection de difficulté très rapidement.

Résultat : Aucun problème.

3 - Plateau

Grille de jeu :

Utilisation normale de la grille de jeu : cliquer sur deux îles voisines crée bien un pont et le clic sur deux îles non voisines ne crée pas de pont.

Résultat : Aucun problème.

Cliquer rapidement sur les îles pour vérifier que le jeu arrive à gérer plusieurs interactions rapidement et qu’il ne crée aucun problème, que ce soit dans l'affichage ou dans l’exécution des méthodes.

Résultat : Aucun problème.

Utilisation des boutons Undo/Redo afin d’observer si les coups joués peuvent être enlevés et remis correctement.

Résultat : Aucun problème.

Utilisation exagérée des boutons Undo/Redo pour voir si tout fonctionne correctement sans que cela ne crée d'erreur.

Résultat : Aucun problème.

Mode hypothèse :

Utilisation normale de la grille de jeu : cliquer sur deux îles voisines crée bien un pont et le clic sur deux îles non voisines ne crée pas de pont.

Résultat : Aucun problème.

Cliquer rapidement sur les îles pour vérifier que le jeu arrive à gérer plusieurs interactions rapidement et qu’il ne crée aucun problème, que ce soit dans l'affichage ou dans l’exécution des méthodes.

Résultat : Aucun problème.

Essayer d’interagir avec le plateau principal (ce qui n'est normalement pas possible avec le mode hypothèse) et vérifier que le bouton de validation réagit bien et retranscrit tout ce qu'il y avait dans la grille hypothèse sur la grille principale.

Résultat : Aucun problème.

Menu Pause :

Utilisation normale du menu pour voir si tout fonctionne bien : l'appel de méthode correspond bien au bon bouton cliqué.

Résultat : Aucun problème.

Tenter de ralentir et de faire bugger le logiciel en cliquant sur plusieurs boutons simultanément

Résultat : Aucun problème.