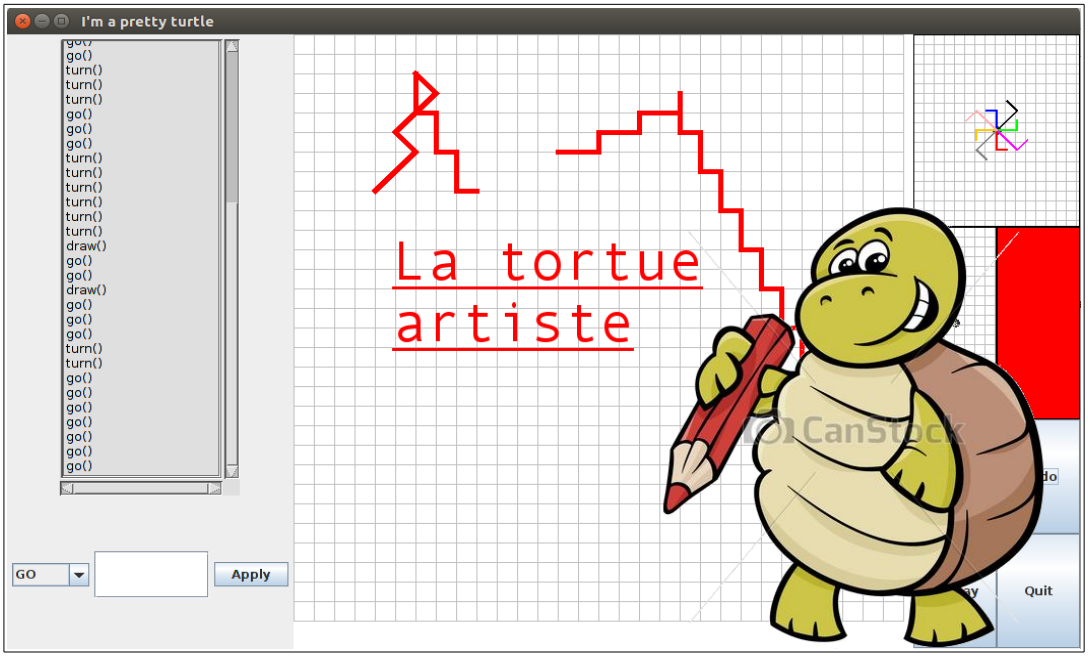
CHARPENTIER Romain
WEHOWSKI Etienne

L3 Informatique

Rapport : Interface Homme Machine

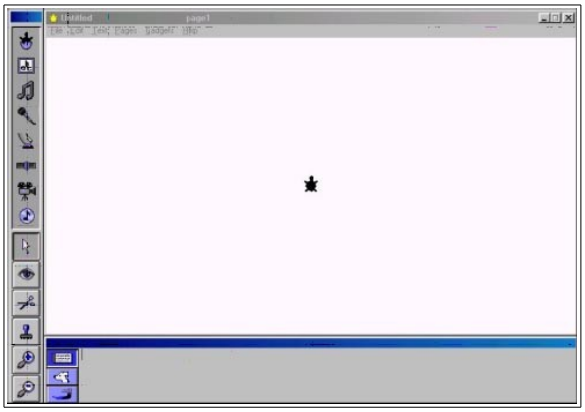


Date : 01/04/2017

Sommaire

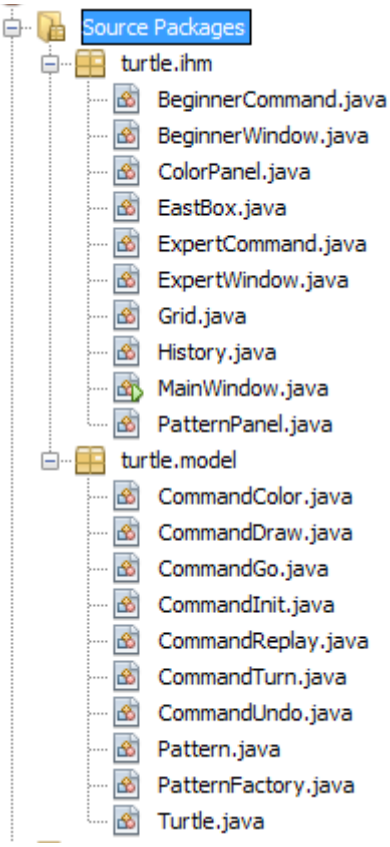
I. Présentation générale

Pour conclure notre UE traitant des Interfaces entre l'Homme et la Machine et pour vérifier nos connaissances et acquis sur la Programmation Orientée Objet en Java, il nous a été demandé de réaliser un ersatz du langage de programmation LOGO.



Ainsi nous avons réaliser notre propre version de ce logiciel selon certaines normes qui nous ont été spécifiée et selon des choix que nous avons fait pour répondre aux demandes du projet.

Nous avons donc d'abord découper le projet en deux sous-dossiers, le premier étant indépendant de l'autre :
- le package « Model » qui contient le noyau fonctionnel de notre projet
- le package « IHM » qui contient l'interface que l'utilisateur verra et certaines autres fonctionnalités en rapport avec l'affichage



Le premier package (« Model »)contient 10 fichiers java :
- 7 fichiers sont utilisés pour gérer les différentes commandes du projet
- 2 fichiers servent à créer et stocker les différents motifs que peut réaliser la tortue
- 1 fichier initialise la tortue et gère ses coordonnées, sa couleur, ses modèles, etc

Le second package contient également 10 fichiers java :
- 3 fichiers permettent l'affichage de l'interface en fonction du mode de fonctionnement choisi par l'utilisateur (« Beginner » ou «Expert ») au démarrage de l'application (création des fenêtres)
- 5 fichiers correspondant aux différents éléments sur les fenêtres et pouvant interagir avec l'utilisateur
- 2 fichiers permettent l'exécution des instructions saisies par l'utilisateur.

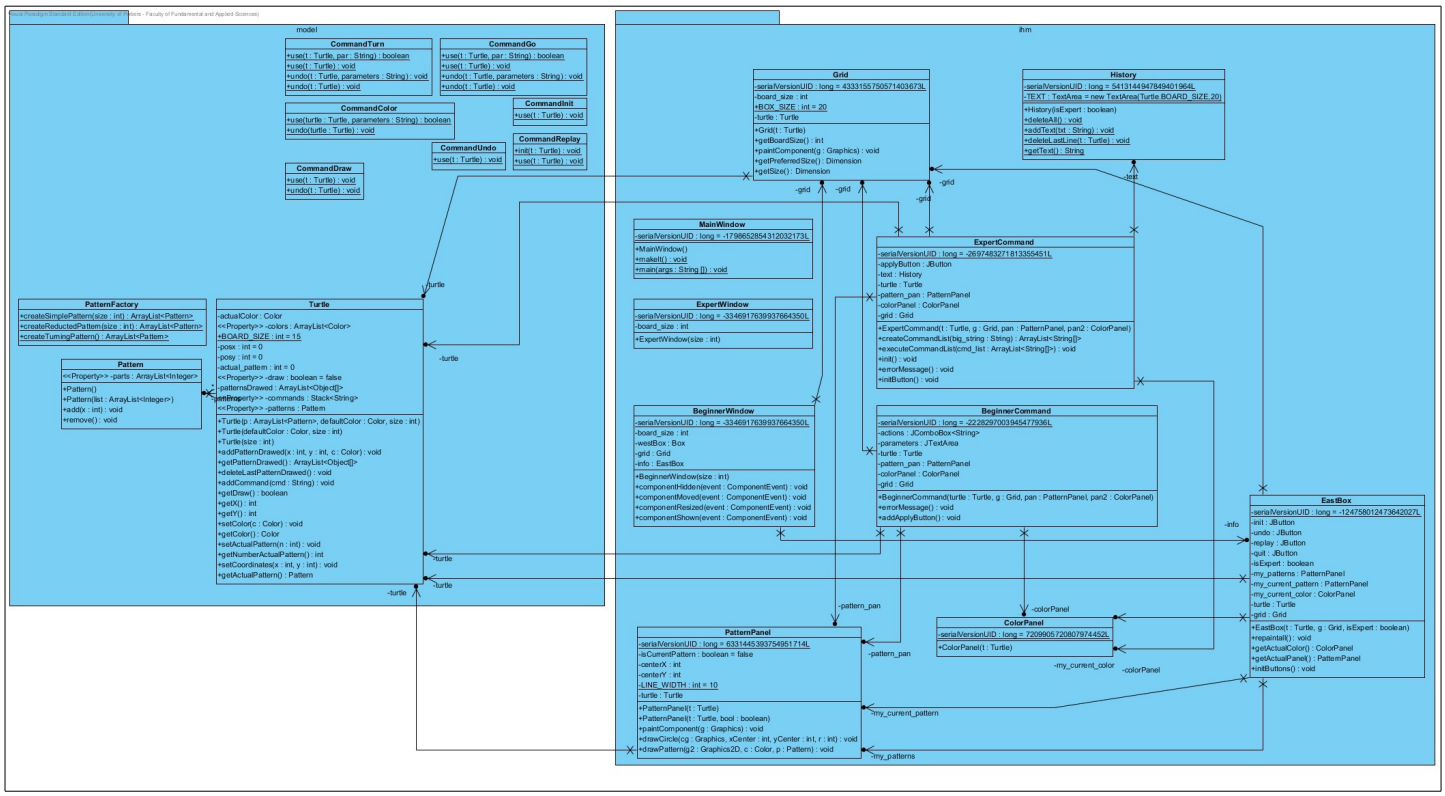


Diagramme UML de notre projet réaliser avec Virtual Paradigm (l'image en taille réelle est dans le dossier du projet)

II. Package Model

Le package « Model » correspond au noyau fonctionnel de notre projet. C'est ici que sont gérés les différents éléments de notre projet tels que la création de la tortue ou encore les commandes qui devront être effectuées.

a) La tortue

```
//----------------------Attributes----------------------//
private Color actualColor;
private ArrayList<Color> colors;
public static int BOARD_SIZE = 15;      //height = width
private int posx=0, posy=0;     //turtle's position
private int actual_pattern=0;
private boolean draw = false;
private ArrayList<Object[]> patternsDrawed;
private Stack<String> commands;

/*
 * List of possible patterns
 */
private ArrayList<Pattern> patterns;
```

Le fichier « Turtle.java » nous permet de créer la tortue, en lui spécifiant ses attributs tels que sa couleur, sa position ou encore la liste des différents motifs qu'elle sait dessiner

Ce fichier contient également les différents constructeurs de cette classe ainsi que ses méthodes que l'on réutilisera par la suite pour permettre les différentes actions de notre tortue.

Pour réaliser toutes les commandes nécessaires, il faut aussi que la tortue connaisse la liste des commandes utilisées (on utilise un pile de string), la taille de la grille de déplacement, la liste des dessins, la liste des couleurs et le motif actuel. Pour retrouver le motif actuel, on utilise un integer qui est l'index du motif dans la liste des motifs.

Les méthodes contenues dans cette classe servent seulement à interagir avec les variables de la classe (par exemple : rajouter un élément à une liste). Pour la liste des dessins (« addDrawedPattern »), on utilise un tableau d'objets. Celui-ci contient 3 cases avec les coordonnées (x,y) et la couleur. Il faudra donc lire ce tableau de 2 en 2 pour avoir les coordonnées de début du dessin et les coordonnées de fin (ce qui sera géré dans d'autres fonctions).

b) Les classes de commandes

Comme expliqué précédemment, nous avons créer 7 classes qui nous permettent de gérer les différentes actions de la tortue. Il y a les commandes fondamentales (turn, go, draw et color) et les autres commandes (undo, replay, init). Pour les commandes fondamentales, nous y ajoutons par exemple une méthode « undo » qui permet de l'annuler. Sinon chacune des commandes comporte une méthode « use » qui permet d'utiliser la commande (si c'est une commande fondamentale, on ajoute aussi le nom de la commande à la liste des commandes de la tortue). Nous tenons aussi à préciser qu'après chaque méthode, nous devons repeindre certains éléments de l'interface mais ces actions sont présentes dans l'« IHM » et non dans le « Model » pour rendre le « Model » indépendant de l'« IHM ». Ces classes sont les piliers centraux de notre projet.

Il faut préciser que les commandes nécessitant un paramètre (pour « use » et « undo ») contiennent une méthode qui va lire ce paramètre (au format string) et essayer de le convertir dans le format nécessaire. On vérifie que cela est possible en mettant tout cela dans un « try...catch ». Dans le « try », on va utiliser le paramètre et renvoyer true tandis que dans le « catch », la méthode va retourner false pour indiquer à l'interface que la commande ne s'est pas correctement exécutée (ce sera à elle d'afficher un message d'erreur). À noter que si le paramètre est vide ou null, on utilise la commande et renvoie tout de suite vrai sans passer par le « try...catch ».

i) CommandTurn

Cette classe permet d'effectuer le changement de motif lorsqu'une de ses méthodes est appelée. On peut retrouver 4 méthodes dans cette classe :

- Pour utiliser la commande, il y a la variable « actual_pattern » de la tortue, il ne suffit que de l'incrémenter ou de revenir au premier si le motif actuel est le dernier. En fonction du paramètre, on peut répéter k fois cette opération.

- Pour annuler la commande, on utilise la même variable où il faut simplement décrémenter cette fois-ci.

ii) CommandGo

Cette classe permet à la tortue de se déplacer selon le motif courant. La encore 4 méthodes ont été créée avec la même logique que la CommandTurn :

- Pour utiliser la commande, il faut interpréter le motif actuel de la tortue en fonction de sa position

dans la grille et de la taille de cette dernière (pour bien comprendre comment sont fait les motifs, voir la partie du rapport correspondante). On parcourt ainsi le motif actuel qui est en fait une liste d'entiers et en fonction de l'entier et des coordonnées de la tortue, on la déplace ou non dans la direction du motif. Si le mode « draw » est actif, il ne faut pas oublier d'ajouter les coordonnées et la couleur à la liste des motifs dessinés (cette liste est déjà expliquée dans la partie sur la tortue). En fonction du paramètre, on peut répéter cette commande k fois (on précise qu'on convertit une chaîne de caractère en entier avec « Integer.parseInt() »).

- Pour annuler la commande, on inverse le motif dessiné dans une variable temporaire pour ainsi effacer dans leur ordre d'apparition. Il n'y a plus qu'à faire la même chose que la méthode « use » mais en inversé (donc -1 au lieu de +1, supprimer les motifs dessinés, etc).

iii) CommandDraw

Cette classe permet de changer le booléen « draw » de la classe tortue définissant si elle est en mode « dessin » ou non. Elle contient 2 méthodes, « use » et « undo » qui servent simplement à modifier ledit booléen. La seule différence est qu'il ne faut pas ajouter « draw() » à la liste des commandes de la tortue si on utilise « undo ».

iv) CommandColor

Cette classe nous sert à affecter a la tortue une couleur passée en paramètres lors de l'exécution de cette commande. Deux méthodes sont présentes dans cette classe :

- Pour utiliser la méthode, il faut nécessairement un paramètre qui sera le nom de la couleur. On convertit la chaîne de caractère en couleur avec :

« `Field fieldClass.forName("java.awt.Color").getField(parameters);`

`color = (Color)field.get(null);` »
Si le paramètre est correct, il ne suffit ensuite que de changer la couleur de la tortue, ajouter la couleur à l'historique de couleurs et la commande à l'historique des commandes.

- Pour annuler la commande, on supprime la dernière couleur de l'historique des couleurs puis on attribue la dernière couleur de l'historique à la tortue.

v) CommandInit

Lors d'un clique sur le bouton « init » de l'interface de dessin la seule méthode de cette classe, intitulée « use » prenant en paramètre une tortue, est appelée et permet de réinitialiser les paramètres de la tortue avec ceux par défaut. Il faut également supprimer tous les éléments des historiques.

vi) CommandReplay

Cette classe contient deux méthodes, la méthode « init » qui ressemble à la commande init sauf qu'on ne supprime pas l'historique des commandes pour pouvoir toutes les refaire dans la méthode « use ».

La seconde méthode, « use » qui prend également une tortue en paramètre, nous permet de rejouer toutes les commandes qui ont été effectuées, on utilise un thread pour effectuer les commandes avec un certain laps de temps entre. Cela permet surtout un meilleur affichage pour l'utilisateur. Il n'y a

plus qu'à copier l'historique des commandes à l'envers dans une autre pile et de les supprimer une à une. Ainsi nous avons toutes les commandes, il ne faut plus que les exécuter une par une.

vii) CommandUndo

La seule méthode de cette classe s'appele « use », prend en paramètre un objet de type Tortue et est appelé lorsque l'utilisateur clique sur le bouton « Undo » de l'interface.

Cette méthode regarde la dernière commande de l'historique de commandes de la tortue et appelle la méthode « Undo » de la classe correspondant à cette dernière, nous permettant ainsi d'annuler la dernière action effectuée par l'utilisateur.

c) Les classes de motifs

i) Pattern

Cette classe nous sert à stocker la liste des motifs connues par la tortue.

Elle contient un seul attribut « parts » qui est une liste d'entiers dont chaque entier correspond à une direction possible (voir schéma). Il est ainsi possible de créer n'importe quel motif.

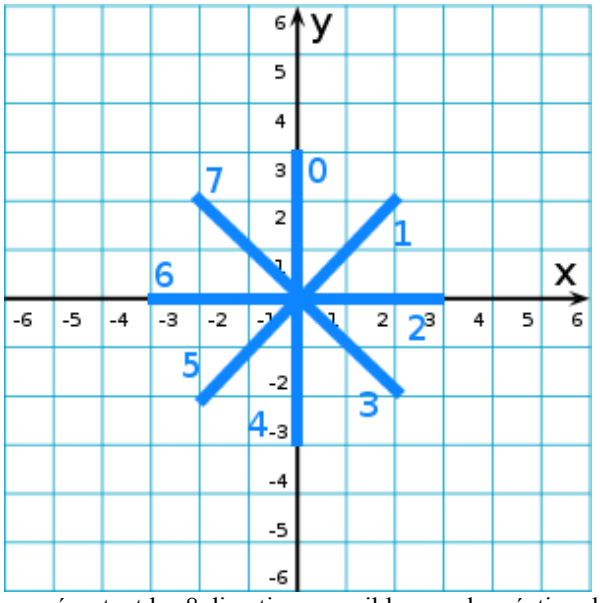
Deux constructeurs existent dans cette classe. Le premier ne prend pas de paramètre et associe à « parts » une nouvelle liste d'entiers vide. Le second prend une liste d'entiers en paramètre et associe à « parts » cette liste.

Trois méthodes sont présentes dans cette classe :

- « Add » qui ajoute un entier à « parts »

- « remove » qui enlève le dernier élément de « parts » si cette liste n'est pas vide

- « getParts » qui retourne l'attribut « parts »



(Schéma représentant les 8 directions possible pour la création de motif)

ii) PatternFactory

La classe suivante nous permet de créer les différent motifs de la tortue. C'est une classe de même type que « BorderFactory » qui utilisent des méthodes qui renvoie certaines listes de motifs. Nous avons choisi d'en créer 3 :

- les 4 directions « classiques »
- les 4 directions « classiques » + toutes les diagonales
- un motif complexe : 2 pas en avant puis 1 pas vers la gauche

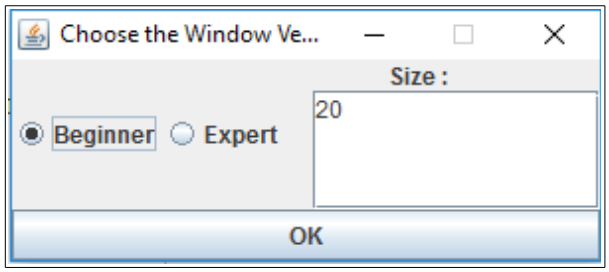
III Package IHM

Le package « IHM » de notre projet correspond à l'interface visible par l'utilisateur et avec lequel il va interagir. Il a été demandé de réaliser deux vues différentes du projet, nous avons choisi de faire la vue numéro 2 du mode débutant et la vue mode expert.

Ces deux vue possèdent un tronc commun mais diffèrent sur certains points.

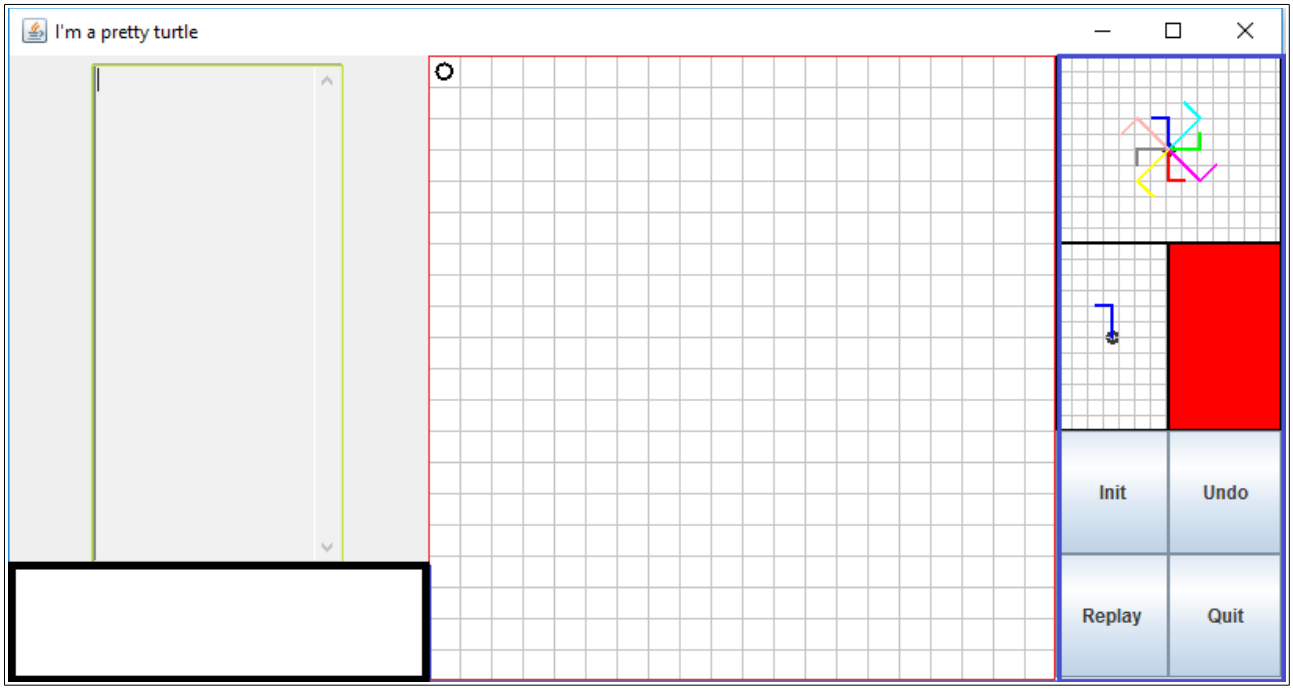
a) Tronc commun

Premièrement, nous avons gérer l'affichage d'un menu de sélection de « difficulté » au lancement du programme. Sur ce menu, il est demandé de choisir si l'on veut la vue « Expert » ou bien la vue « Beginner » ainsi que la taille de la grille sur lequel l'utilisateur va être amené à dessiner.



(le menu principal)

Les deux boutons radios sont liés dans un groupe de boutons qui permet de n'en sélectionner qu'un seul. Pour la taille, il s'agit d'un « TextArea ». Lors de l'appui sur le bouton « OK », nous allons vérifier la taille pour que celle-ci soit un entier compris entre 10 et 40 (ce choix nous permet d'avoir des tailles de grilles raisonnables). Si la condition n'est pas remplie, on affiche un message d'erreur pour avertir l'utilisateur. Sinon on crée une deuxième fenêtre en fonction de l'interface et de la taille puis on ferme la fenêtre actuelle.



(Tronc commun de l'interface utilisateur)

Deuxièmement, les vues possèdent une interface quasiment similaire. On retrouve sur les deux vues :

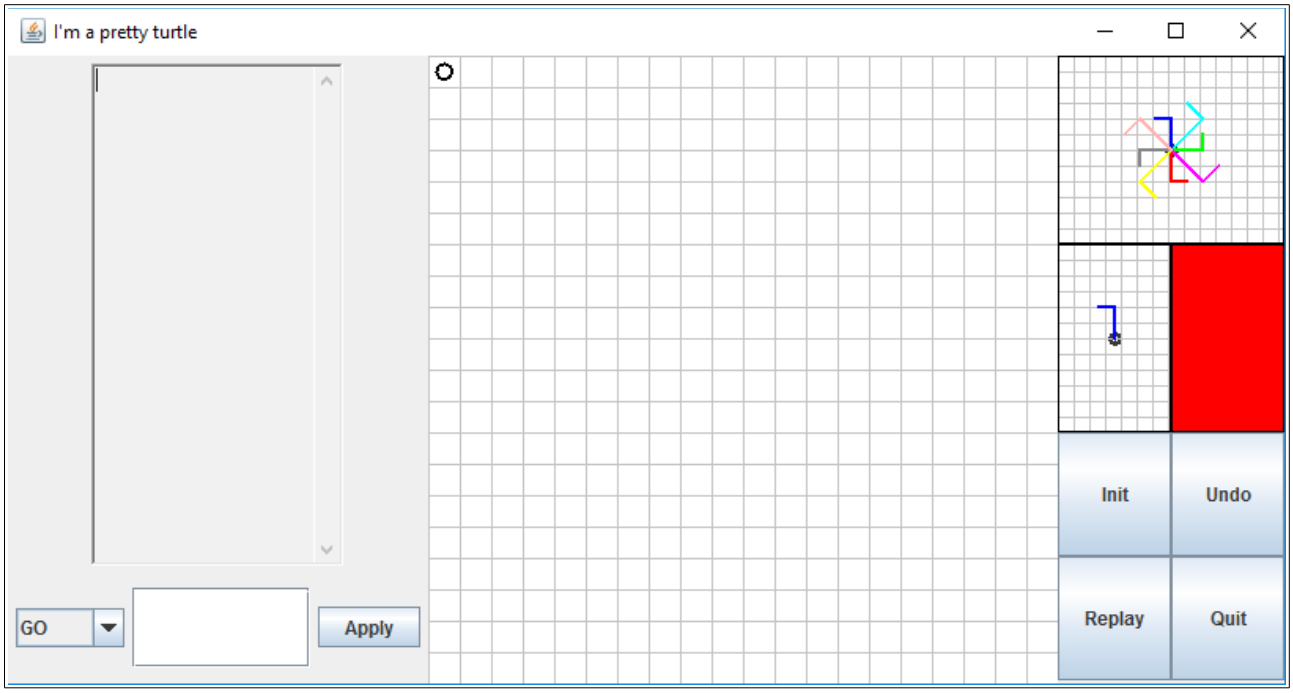
- Entourée de rouge, une grille issue de la classe « Grid » (un JPanel) sur laquelle vont être dessinés les motifs et l'affichage de la tortue. Pour dessiner les motifs, on utilise la méthode « paintComponent() » qui va en fonction des motifs dessinés et de leur placement, les représenter graphiquement avec leur couleur sur la grille. Cela explique certaines variables présentes dans la classe « Turtle ». En fonction de la taille de la fenêtre, les dimensions des cases varient (variable BOX_SIZE) pour permettre un affichage optimal en fonction de la fenêtre (la fenêtre qui possède une taille minimal pour que tout reste visible).

-Entourée de bleu, un panneau issue de la classe « EastBox » qui permet d'afficher les motifs connus par la tortue, le motif en cours d'utilisation, la couleur actuelle et 4 JButton qui permettent respectivement de réinitialiser l'application, d'annuler la dernière action, de rejouer toutes les commandes qui ont été jusqu'alors effectuée avec une pause entre chaque et de quitter l'application. L'affichage des motifs se fait dans des JPanel qui peuvent être « peints » pour ainsi afficher les motifs. On utilise pour cela la même méthode que pour dessiner sur la grille principale. On y dessine également le point central et un quadrillage. Il ne faudra aussi pas oublier de les repeindre en cas de changement de motifs.

-Entourée de vert, une text area issue de la classe « History » qui correspond à l'historique des commandes effectuées. En fonction de la vue choisie, cette zone est éditable ou non. Si elle n'est pas éditable, alors il existe une méthode qui écrit une chaîne de caractère dans la TextArea en faisant aussi un saut de ligne.

La seule différence entre les deux vues est la zone entourée de noir qui va servir à interpréter les commandes fondamentales.

b) Interface « Beginner »



(l'interface « Beginner » de base)

Si l'on choisit d'utiliser une interface « Beginner » alors le menu principal se ferme et est créer une instance de la classe « BeginnerWindow ».

L'interface débutant est celle dont l'historique des commandes n'est pas éditable. On y voit s'y afficher petit à petit les commandes qui sont sélectionnées grâce au menu déroulant situé juste en dessous.

(les différentes options du menu déroulant)

La zone de sélection de commande est composée de 3 éléments :

- Un menu déroulant qui permet de sélectionner l'action que doit effectuer la tortue.

- Une text area éditable dans laquelle est lu le texte et est interprété.

- Un JButton qui va permettre d'interpréter l'action choisie et y associée le texte de la text area. Si le texte ne corresponds pas au paramètre nécessaire à l'action choisie, rien ne se passe et une erreur est affichée. Si aucun texte n'est présent alors le résultat dépendra de l'action choisie :

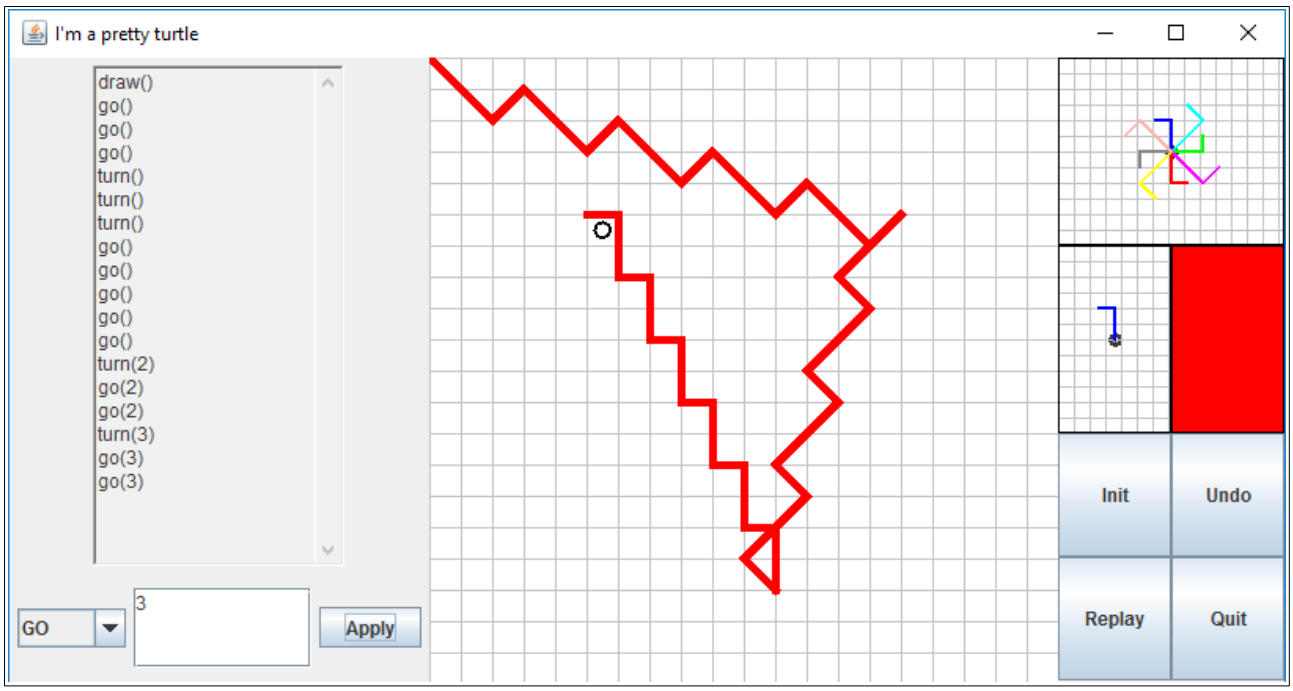
    -si l'action choisie est « go » alors la tortue se déplacera une fois selon le motif courant.

    -si l'action choisie est « turn » alors la tortue passera au motif suivant de sa liste.

    -si l'action choisie est « draw » alors la tortue passera ou non en mode dessin

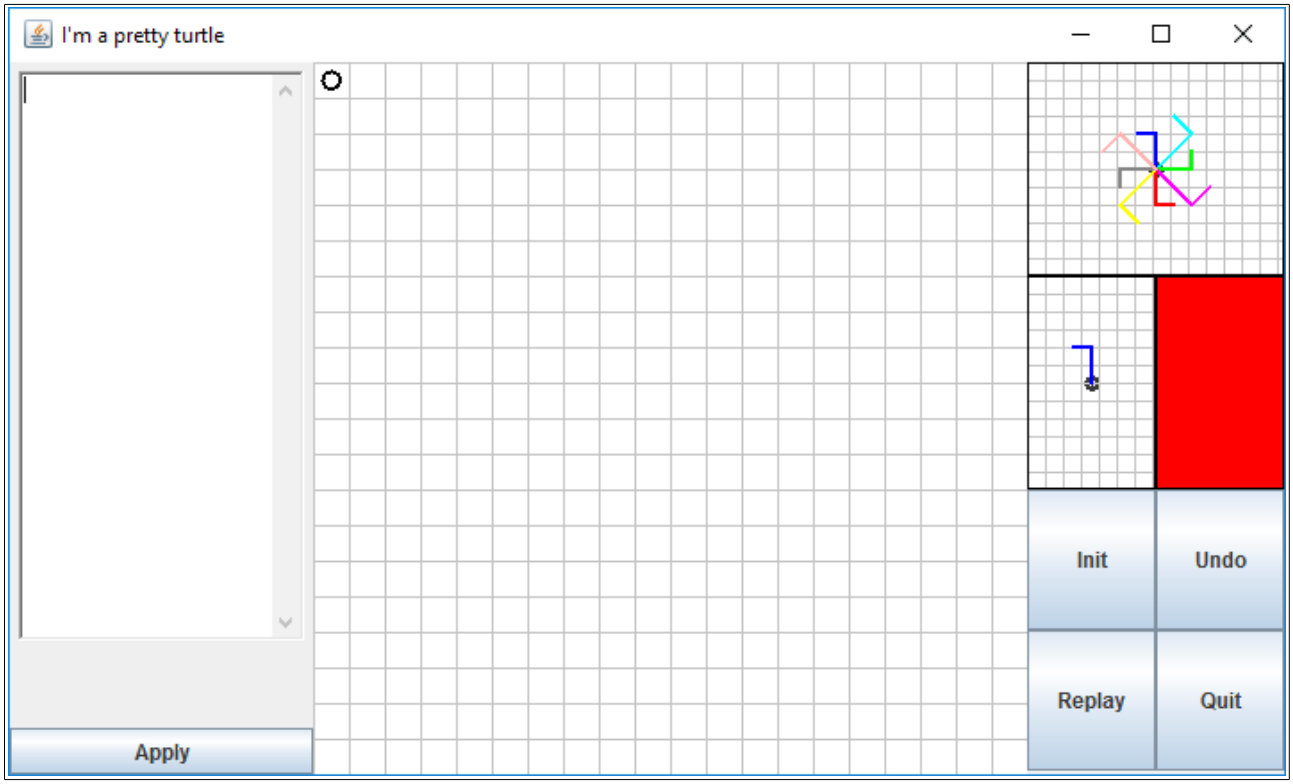
    -si l'action choisie est « color » en revanche alors un message d'erreur sera affiché puisqu'il faut passer à cette action une couleur de java.awt.Color

Toutes ces actions sont gérée par la méthode « addApplyButton » de la classe BeginnerCommand.



(un exemple d'utilisation de la vue « Beginner »)

c) Interface « Expert »



(l'interface « Expert » de base)

Si l'on choisie cependant d'utiliser une interface « Expert » alors le menu principal se ferme et est créer une instance de la classe « ExpertWindow ».

L'interface expert a son historique des commandes éditable. Contrairement à l'interface « Beginner », il n'y a pas de menu déroulant contenant les actions possibles ou une text area mais seulement le même JButton « Apply ». De plus, la commande « Init » ne supprime pas l'historique.

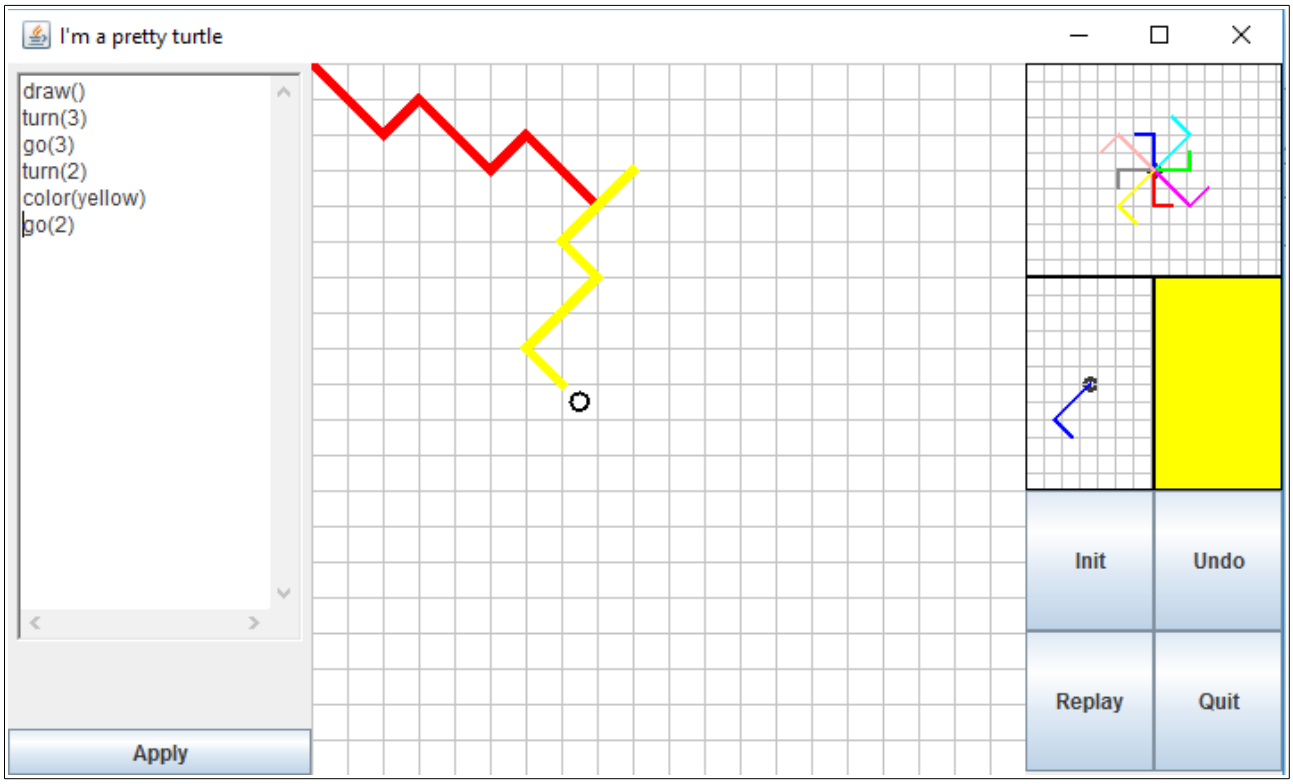
Le principe de fonctionnement est légèrement différent de l'autre interface. Ici, l'utilisateur rentre, dans la text area, toutes les commandes qu'il souhaite effectuer les unes à la suite des autres, avec des paramètres pour chaque type d'actions s'il le souhaite.

Lorsque le bouton « Apply » est pressé, tout est réinitialisé, le texte est récupéré et est analysé grâce à la méthode « createCommandList » de la classe « ExpertCommand ».

Cette méthode va découper toute la chaîne de caractère de la zone texte et va retourner une liste de chaîne de caractères plus courtes, chaque sous-chaîne correspondant à une instruction qui a été reconnue par la méthode.

Ensuite, tant que cette liste de commande n'est pas vide, on exécute chaque commande une à une pour obtenir le dessin de l'utilisateur.

Une fois que toute les instructions ont été suivie et qu'aucune erreur n'ai été remarquée (un mauvais paramètre telle qu'une couleur dans un « go() » ou encore un entier dans un « color() »), si l'utilisateur rentre de nouvelle instructions, toute la nouvelle chaîne de caractère est analysée et effectuée. Sinon le processus se stoppe et annule tous les changements.



(un exemple d'utilisation de la vue « Expert »)

IV. Informations complémentaires

Outre l'UML, vous retrouverez dans le dossier de notre projet tout le code de l'application ainsi qu'un fichier .jar pour l'exécuter et d'un dossier contenant toute la javadoc.

Pendant ce projet, certains choix ont été fait délibérément. C'est pourquoi la grille dans le JPanel n'est pas parfaitement centré et qu'elle ne prend pas tout l'espace. En effet, la tortue se déplace sur les lignes du tableau et donc il serait impossible de la voir en position (0,0) (par exemple) sinon. Et la taille des cases de la grille varie selon la taille de la fenêtre, c'est pourquoi il y a une taille minimale à la fenêtre car sinon les cases ne seraient vite plus visibles. De plus nous considérons l'application inutilisable en dessous d'une certaine taille de fenêtre.

La tortue est symbolisée sur la grille par un cercle qui est vide si elle ne dessine et est remplie de la couleur de la tortue si elle dessine. C'est une option intéressante ajoutée pour savoir quand elle peut dessiner et ne pas à avoir à vérifier l'historique pour cela.