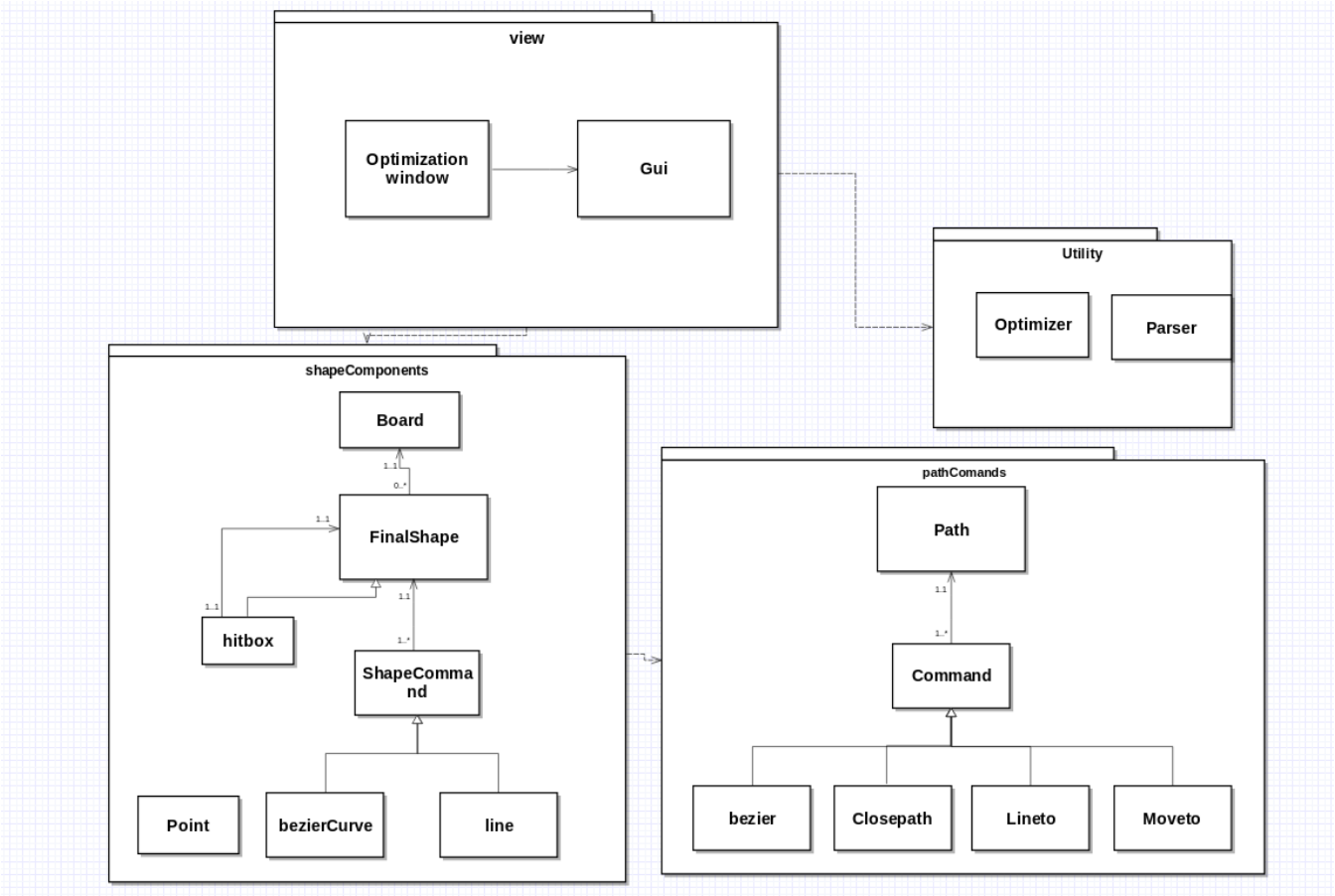
Van Assche Jules
Licence 3 Informatique

Compte rendu POO

Table des matières

Van Assche Jules
Licence 3 Informatique

UML



Van Assche Jules
Licence 3 Informatique

Explication des différentes classes/packages

Le programme fonctionne de la manière suivante :

1. Récupérer le chemin du fichier svg puis parser tout les path du fichier svg en objets Command.
2. Une fois une liste de path et de commandes (pour chaque path) obtenues, il faut passer les coordonnées des commandes path en valeurs absolues pour pouvoir traiter les formes (calcul et affichage) à l'aide d'un curseur représentant la position courante.
3. Une fois les coordonnées converties, on affiche à l'écran les différentes « formes » (ensembles de ligne et de courbe de béziers), et on calcule les hitbox de ses formes.
4. Ensuite l'utilisateur peut optimiser, ce qui a pour effet d'instancier un objet optimizer et se chargera d'optimiser le « board » (ensemble de formes). Puis on repeint le panel optimization pour afficher le résultat.

Parser

Etapes du parser :

1. Rechercher la taille et la hauteur du svg (voir algo recherche en annexe)
2. Compter le nombre de path
3. Chercher une balise <path (algo recherche)
4. Chercher la liste de commandes d= « (algo recherche)
5. Debut du parsing des commandes :

Le parser parse la liste de commandes d'un path tant qu'il ne rencontre pas de guillemets fermant.

Le parser distingue 2 type de commandes

- Chiffre ou signe - ➔ commande implicite
- Lettre ➔ commande explicite

Ensuite en fonction de la nature de la commande le traitement diffère.

Path command, shape command

Après avoir parser les path et les avoir transformé en objet path command, il faut transformer les cordonnées en cordonnées absolues en maintenant un curseur représentant la position dans l'espace courante. Une fois ceci fait, on se retrouve avec une liste de shape contenant des shape command qui ne sont que des segments et des courbes de Béziers exprimé en cordonnées absolues.

Van Assche Jules
Licence 3 Informatique

Toutes les formes calculé se retrouve dans une liste que contient l'objet Board, on va ensuite calculer pour toutes les formes des « hitbox » , un carré passant par le maximum et le minimum de chaque forme. Chaque forme à un attribut hitbox.

Affichage

Il suffit de passer les cordonnées dans le positif puis d'afficher les segment avec drawLine de Swing et les courbes de Béziers en utilisant l'équation paramétrique. On incrémente un t avec une précision de 0.01 et tant que t<1 on appelle une méthode d'instance de la classe bézier qui nous renvoie un point pour un t donné.

Optimisation

Ensuite l'utilisateur peut optimiser les formes affichés à l'écran pour qu'elle prenne le moins de place possible. . Un objet optimizer sera instancié qui aura pour attribut un board et comme tâche de l'optimiser.

Van Assche Jules
Licence 3 Informatique

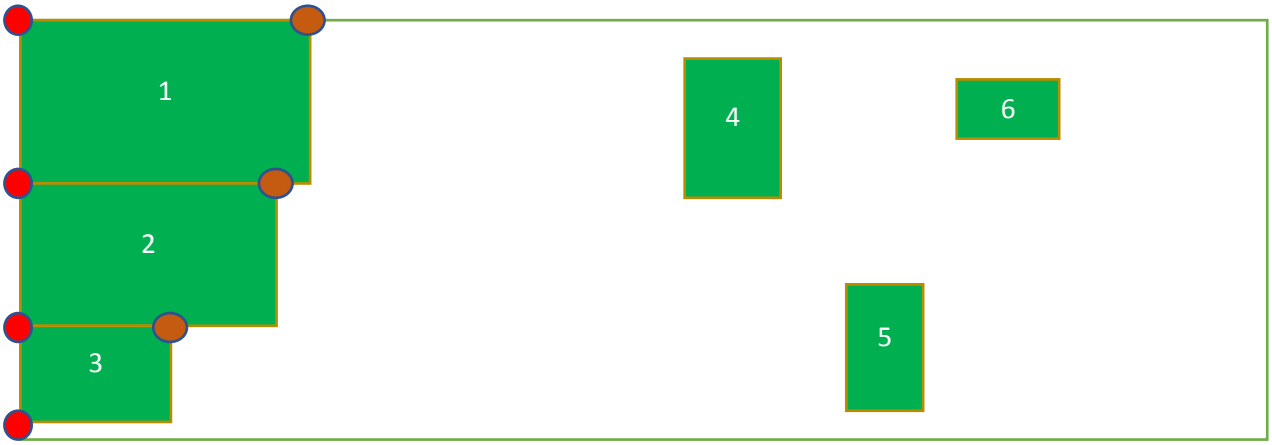
Algorithme optimisation

La première chose qui doit être faite est le tri de la liste des formes en fonction de la largeur de leur hitbox. L'algorithme est prévue pour fonctionner avec une liste d'hitbox triées par ordre décroissant de largeur. (l'algorithme peut aussi essayer d'optimiser avec une liste triée par ordre croissant mais il n'est pas fait pour cela à la base et donc quelques collisions peuvent survenir !).

Une fois la liste trié l'algorithme commence le travail , on distingue 3 type de phases :

- Une phase d'initialisation (aussi descendante)
- Une phase descendante
- Une phase ascendante

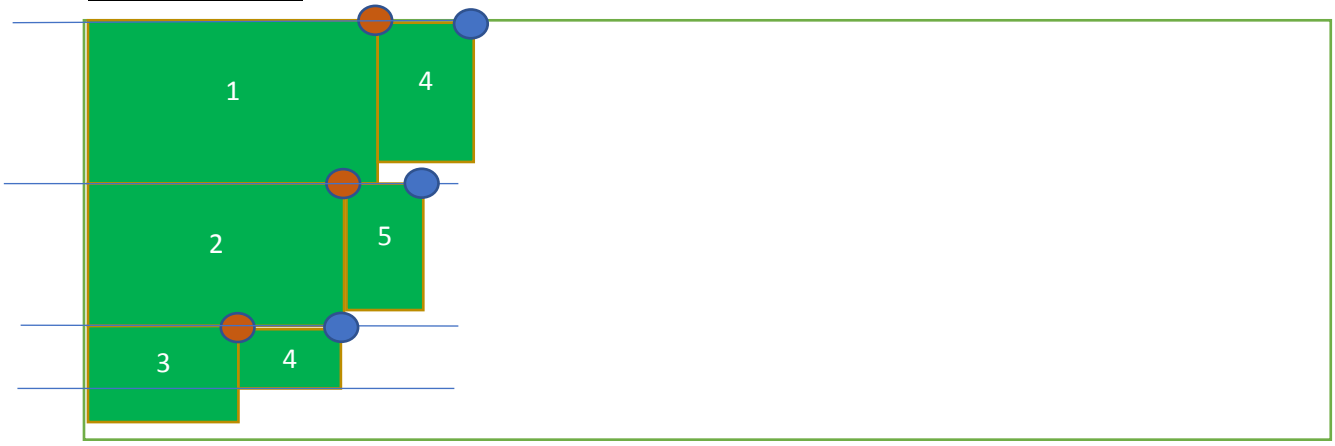
Phase d'initialisation



On initialise une position (1er point rouge en haut a gauche) puis on choisie dans la liste la 1ère hitbox (donc la plus large) et on la place, on actualise ensuite la position en ajoutant la hauteur de la hitbox. On continue jusqu'à ne plus pouvoir remplir le rouleau (en hauteur).

Tout en remplissant le rouleau , on va à chaque placement de forme et actualisation du curseur position remplir une pile de point d'accroche possible pour la phase ascendante.(point brun)

Phase ascendante

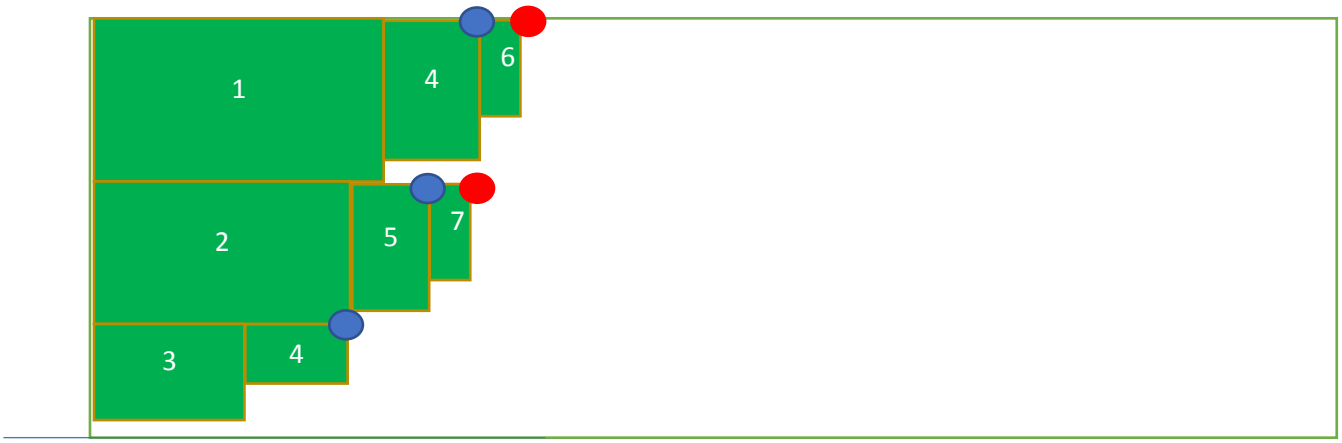


Van Assche Jules
Licence 3 Informatique

Maintenant on va remonter en testant pour chaque point d'accroche toutes les hitbox possibles. On initialise un Y max à ne pas dépasser qui sera actualisé si une forme est correctement placé. (barres bleues). Si jamais aucune forme ne rentre pour un point d'accroche on va simplement passer au point suivant (mais en gardant le y max) histoire d'avoir plus d'espace.

Comme avant on maintient une liste de point d'accroche mais pour la phase descendante cette fois ci. (point bleues)

Phase descendante



La phase descendante est très similaire à la phase d'initialisation, on depile la pile des possibles point d'accroche pour la phase descendante et comme pour la phase ascendante on test si la forme rentre (sauf que cette fois ci le y max est fixé à la hauteur du rouleau : ligne bleue).

Tout comme la phase d'initialisation, on mets à jour la pile de point possible pour la phase ascendante (point rouge)

Et on continue, phase ascendante, descendante, ascendante … Jusqu'à ce qu'il ne reste plus aucune forme à placer.

Rappel : l'origine (0.0,0.0) est situé en haut à gauche (swing)

On utilise :

- **shapes_toPlace** : Une liste de forme à placer (trié dans l'ordre décroissant de la largeur de leur hitbox) avec leur hitbox respective : initialisé avec la liste de formes de l'attribut board
- **shapes_Placed** : Une liste résultat contenant les formes avec leurs nouvelles positions : initialisé vide
- **positionCursor** : Une position (x,y) : initialisé à 0
- **YMax** : Un y maximum : initialisé à la hauteur du rouleau

- **ascendingPossiblePoint** : Une pile contenant les possible point d'accroches pour une phase ascendante : vide
- **descendingPossiblePoint** : Une pile contenant les possibles point d'accroches pour une phase descendante : vide
- index : L'index de la forme courante à placer : initialisé à 0
- **descendingPhase** : un boolean indiquant si nous sommes en phase descendante (sinon en phase ascendante) : true
- **initializationPhase** : un boolean indiquant si nous sommes en phase d'initialization : true

Condition d'arrêt :

- La liste de forme à placer est vide

Algo de recherche

```
public boolean isFound(FileReader b, int index, String pattern)
{
	int c = b.read();

	if(c== (int)pattern.charAt(index))
	{

		// last char of path, now we need to stop recursive call
		if(index==pattern.length()-1)
		{

			return true;
		}
		else
		{
			return isFound(b,index+1,pattern);
		}
	}
	else
	{
		return false;
	}
}
```