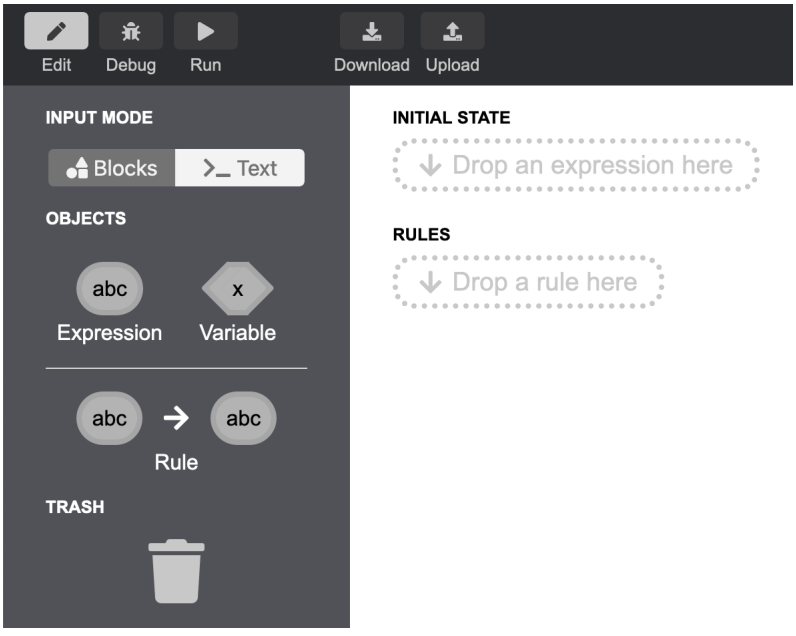
UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES

Département d'informatique
Université de Genève
Travail de bachelor

FunView

Extension de FunBlocks
github: MatthieuVos/FunView



Étudiant:
Matthieu Vos

Professeur:
Didier Buchs

semestre de printemps 2020

Table des matières

Chapitre 1: Introduction

1.1 Apprendre l'informatique

A l'heure actuelle, l'informatique est omniprésente. Qu'on l'utilise à des fins de loisirs ou de manière professionnelle, tout le monde utilise un outil informatique. Cependant, dans le système scolaire, l'informatique et son enseignement n'ont pas une très bonne image. Souvent relié aux mathématiques, beaucoup y voient un amas de formules . Cependant, l'informatique peut très bien être utilisé pour résoudre des problèmes sans calcul. Dans le cadre d'un remaniement de l'enseignement de l'informatique dans les collèges de Genève, des cours ayant pour but d'introduire l'informatique sans utiliser les mathématiques ont été aménagés. Pour ce faire, de nouveaux outils pédagogiques doivent être développés. Nous allons ici discuter du développement d'un de ces outils.

1.2 FunBlocks

FunBlocks est un outil développé dans le but de sensibiliser l'élève à la programmation informatique. Il consiste à manipuler des termes et des expressions, sous forme de block, afin de représenter des données et des instructions. Il se base sur la programmation fonctionnelle qui consiste à représenter le programme sous la forme de fonctions composées d'expressions plus complexes. Cette représentation se rapproche beaucoup de la manière dont l'algèbre est enseigné au collège de Genève.

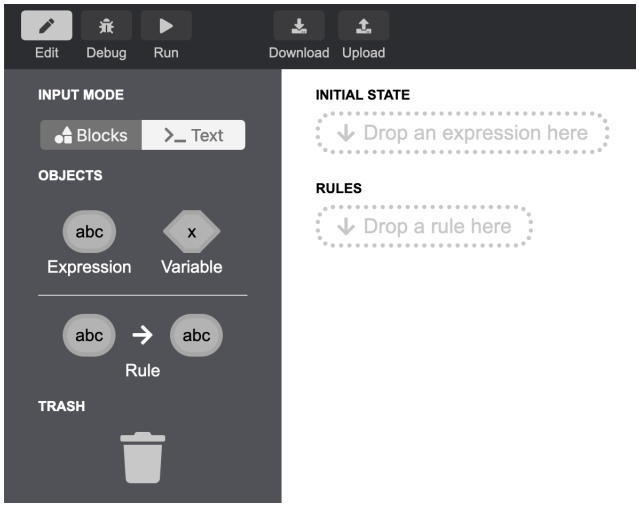


Figure 1.1: Interface FunBlocks

1.3 FunView

L'outil Funblocks permet d'avoir un moyen intuitif d'écrire un programme afin de représenter un état et ses dérivations. Cependant, une représentation graphique de cet état permettrait une meilleure compréhension de ce qui est vraiment modifié. Il a donc été suggéré d'introduire un système réagissant à l'état du programme afin de visualiser les changements et faire des parallèles entre le code et une image. Le but est que la fonction soit exprimé visuelement par la position de formes géométriques les une par rapport aux autres.

1.4 L'environnement

Funview est une extension de Funblocks, il semblait donc évident de coder dans le même environnement. Ainsi, Funview a été codé en Typescript, une amélioration du langage de programmation Javascript. Funview reprendra des éléments définis dans Funblocks dont je parlerai plus tard. La représentation graphique sera un canvas HTML. Je détaillerai plus tard les raisons de ce choix.

Chapitre 2: FunBlocks

FunView est un add-on(extension) de FunBlocks. Il a donc été nécessaire de comprendre comment fonctionnait FunBlock afin de pouvoir créer FunView.

2.1 L'interface de FunBlocks

Au début du projet, Funblocks se présentait comme suit:

- Un onglet Edit : Cet onglet de départ permet de créer son état initial et d'introduire des règles de transition. Il possède 2 modes, Blocks et Text. Les deux modes sont liés et un changement dans l'un modifie l'autre.
  - Blocks permet de manipuler directement les objets à partir de la liste présente sur la gauche.
  - Le mode Text est un éditeur de texte. Il permet de représenter les objets utilisés dans le mode Blocks sous forme de code.
- Un onglet Debug : Cet onglet est celui appelé Debug. Il permet de modifier l'état en fonction des règles établies dans le premier onglet. Pour appliquer une règle à un terme de notre état initial, il faut cliquer sur la règle puis cliquer sur le terme désiré. Un système d'historique permet de revenir à un état précédent.
- Un onglet Run : En cours de développement.

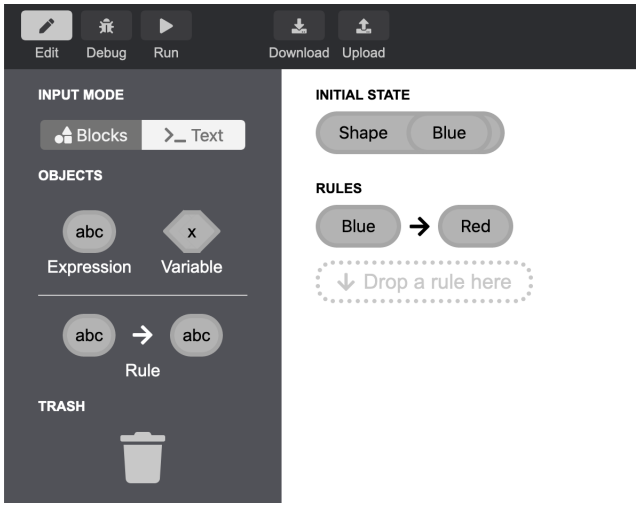


Figure 2.1: Onglet Edit dans le mode Blocks

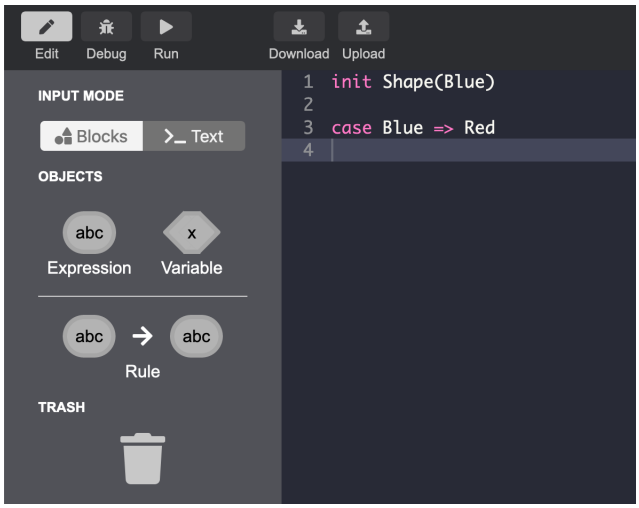


Figure 2.2: Onglet Edit dans le mode Text

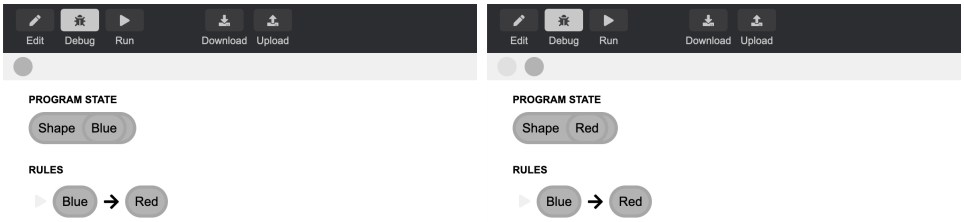


Figure 2.3: Onglet Debug avec l'état avant et après l'application d'une règle

2.2 L'état

L'état est représenté de trois manières différentes : sous forme de Blocks, sous forme de texte et de représentation graphique. Le but premier de FunView est de représenter à tout moment l'état décrit dans Funblocks sous forme graphique. L'état est donc un élément qui doit être le même dans FunBlocks et dans FunView, car tout ce qui est exprimé dans cet état doit avoir un sens dans sa représentation graphique.

2.2.1 L'état dans FunBlocks

Dans Funblocks, l'état est présent sous la forme d'un arbre de syntaxe abstraite, ou AST pour abstract syntax tree. C'est un arbre dont les noeuds sont les opérateurs et les feuilles sont les opérandes. Ici, les noeuds sont des expressions et les feuilles sont des termes. Une expression est une extension d'un terme. Je ne rentrerai pas dans les détails de tous les attributs d'un terme et d'une expression, car une grande partie n'est pas utile au développement de FunView. Je me concentrerai sur ce qui est utilisé pour FunView.

Un terme possède un id et un label. L'id permet au terme d'être unique. Le label est le texte écrit dans un Block. Une expression possède les mêmes attributs qu'un terme. Cependant, une expression possède en plus un tableau de termes qui sont ses sous-termes. Sur la figure 2.6, on peut voir la représentation en Blocks d'une expression avec le label Expr(expression) et ses sous-termes ayant chacun le label subterm.

Il est à noter que dans cette figure les sous termes ont les mêmes label, mais pas les mêmes id. On les différencie.

Figure 2.4: Une expression avec deux sous-termes

2.2.2 Le changement d'état

Lorsqu'une règle est appliqué à un état, ou plus précisément à un terme, l'état dans FunBlocks change. Ainsi, le changement aura un impact sur le label et les sous-termes du terme.

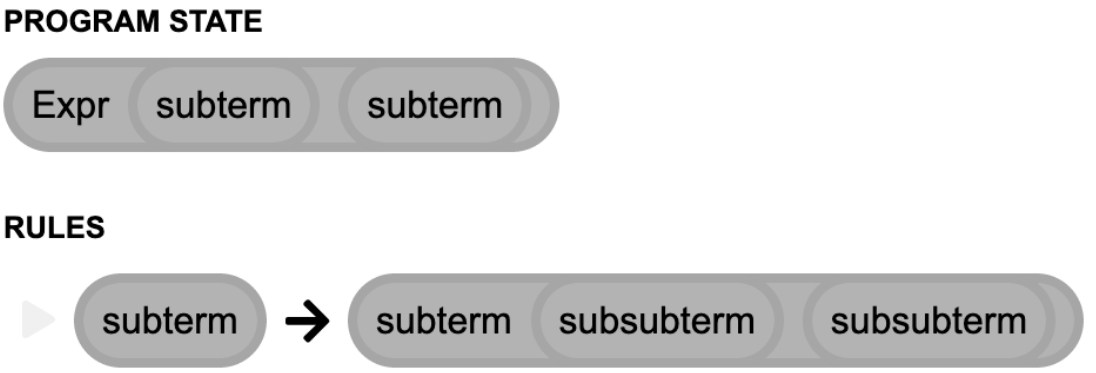


Figure 2.5: Une expression

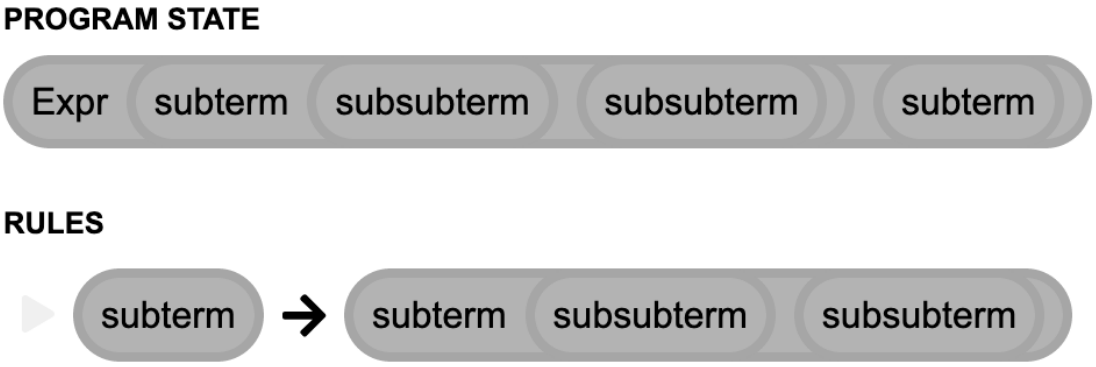


Figure 2.6: Une expression modifiée à l'aide d'une règle

Chapitre 3: FunView

Le principe de Funview est simple : un canvas html sur lequel on dessine nos formes et un parseur qui permet de traduire notre état en un ensemble de formes. On passe donc de l'état sous forme AST à un canvas HTML.

Le parseur prend en entrée l'expression parent de l'état. Il en récupérera les enfants grâce à la structure d'une expression. Le parseur accepte un état respectant une certaine grammaire. Un fois cela fait il transmet la suite d'action à prendre à une classe qui modifiera le canvas en conséquence.

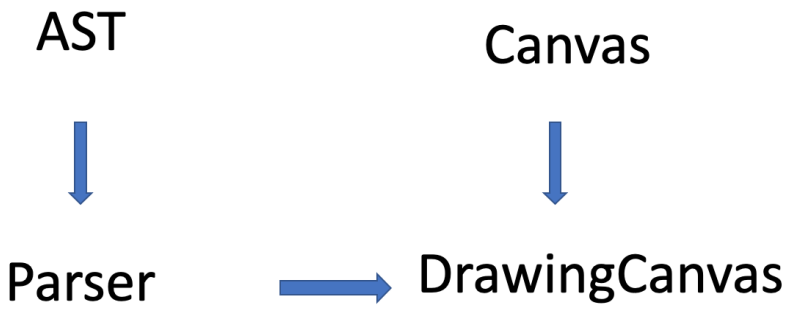


Figure 3.1: Le travail de Funview

3.1 La grammaire

L'objectif par l'ajout d'une représentation graphique, était de montrer l'état sous une autre forme. Il a fallu décider quelle forme allait prendre l'état. Il a été décidé de pouvoir afficher des formes géométriques et de pouvoir exprimer leurs positions par rapport à d'autres formes. Une grammaire régissant les formes et les positions a du être créée. L'idée est de créer un langage qui permet de traduire des expressions en images. L'utilisateur aura à sa disposition des formes, des couleurs et des fonctions pour assigner une position à chaque forme.

La grammaire a connu plusieurs versions. Les débuts étaient très basiques et ont surtout permis de se familiariser avec les règles du canvas HTML.

```
<etat> -> <terme>
<terme> -> <commande> ( <terme> , <terme> ) | Shape ( <form>, <
    color>)
<commande> -> Over | Under | LeftOf | RightOf
<form> -> Square | Circle | Triangle
<color> -> Red | Blue | Green | Yellow
```

Les commandes permettent de placer une forme en fonction d'une autre. *Shape* permet de déclarer une forme d'une certaine couleur.

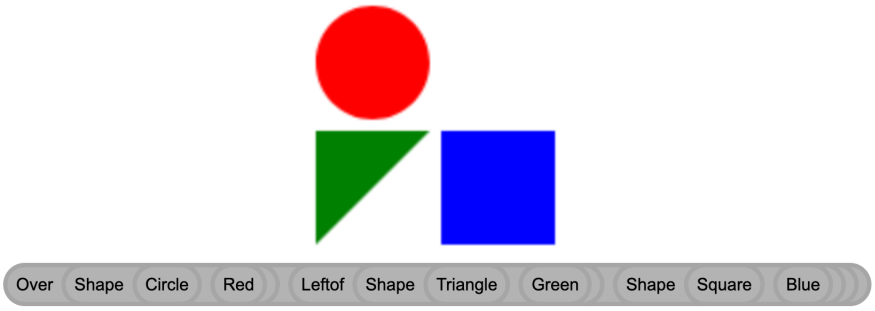


Figure 3.2: Un état accepté par la grammaire en représentation graphique et en block

La commande *Cons* a ensuite été rajoutée pour avoir un système de liste. Elle est semblable à *LeftOf*. Une forme *empty* a aussi été ajoutée afin de définir la fin d'une liste.

Par la suite, deux nouvelles commandes ont été rajoutées. Ces nouvelles commandes sont des versions améliorées des précédentes et bien plus puissantes. Elles ont une signature différente des anciennes commandes. La grammaire a donc été légèrement modifiée.

```
<etat> -> <expr>
<expr> -> <commande> ( <expr> , <expr> ) |
    Shape ( <form>, <color>) |
    Draw ( <expr>, <expr>, Offset(X,Y) ) |
    Group (Forms (<shapes>) , Pos( <pos>))
<shapes> -> <shapes> ,Shape( <form>, <color>) |
    Shape( <form>, <color>)
<pos> -> <pos> , X,Y | X,Y
<commande> -> Over | Under | LeftOf | RightOf
<form> -> Square | Circle | Triangle
<color> -> Red | Blue | Green | Yellow

<pos> doit avoir une taille deux fois plus grande que <shapes>
    afin d'avoir un X,Y par Shape.
```

Nous avons donc *Draw* 3.3 et *Group* 3.4 qui permettent respectivement de placer deux formes avec une certaine position de différence et de placer une liste de formes a des positions indépendantes entre elles.

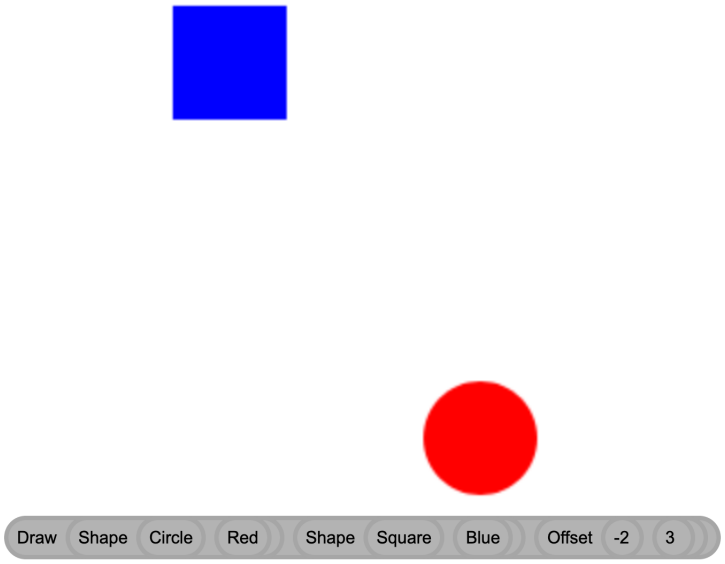


Figure 3.3: Exemple de la commande Draw

3.2 Le parseur

Le parseur applique donc nos règles de grammaire à l'état sous forme AST. Pour cela, il utilise une fonction récursive qui traduit chaque étage par une action. La première chose à savoir c'est que le canvas HTML fonctionne selon un certain ordre. Par exemple, si l'on veut colorier une forme il faut d'abord donner la forme à dessiner puis donner la couleur pour ensuite lui dire de remplir la forme. Si l'on veut déclarer la position d'une forme, il faut donner la position en premier. Ainsi, le parseur aura comme rôle principal d'établir l'ordre dans lequel les instructions sont envoyées au canvas.

La fonction récursive est appelée *explore* et est une méthode de la classe DrawnState. Cette classe contient uniquement la-dite méthode et un appel à la classe manipulant le canvas, DrawingCanvas. La manière de procéder d'*explore* est d'agir en fonction du label de l'expression courante (le noeud courant de l'AST). D'après le label, elle demandera à DrawingCanvas d'agir en conséquence ou bien elle se rappellera elle-même selon l'ordre adéquat.

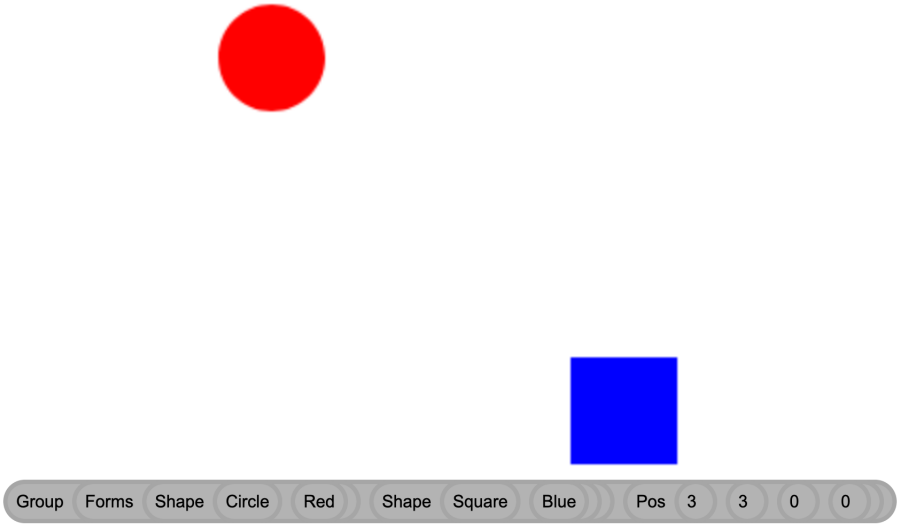


Figure 3.4: Exemple de la commande Group

3.3 Le canvas HTML

Le canvas HTML est un élément permettant de modifier une zone graphique à l'aide d'un script. Il est souvent utilisé pour modifier une image ou bien jouer des animations. Au début du projet, il avait été question d'utiliser le format SVG afin d'avoir des images vectorielles. Cependant, j'ai décidé d'utiliser le canvas HTML car il m'était déjà familier.

Le canvas nécessite une mise en place avant de pouvoir travailler dessus. En premier temps, un objet canvas est ajouté au fichier index.html. Puis il est modifié et manipulé dans la classe DrawingCanvas qui se charge de toute la partie dessin.

La partie la plus compliquée était le positionnement des formes. Il fallait déplacer le pointeur désignant le début du dessin à gauche, à droite, en haut et en bas et pouvoir retourner à la position précédente après avoir dessiné la forme. Les méthodes *X()* et *Y()* permettent de gérer les positions des formes comme voulu. Dans ces méthodes, j'ai rajouté un moyen de modifier la taille du canvas si l'ensemble de formes sortait des limites.

La classe DrawingCanvas contient également les méthodes *draw* et *paint* qui sont appelées par la classe Parser. La méthode *draw* permet de construire la figure en fonction de la chaîne de caractères donnée en argument. Cette méthode permet également d'afficher si une erreur a été commise lors de l'écriture des blocks. La méthode *paint* quand à elle assigne une couleur à la forme décrite précédemment. Elle prend également une chaîne de caractères en argument.

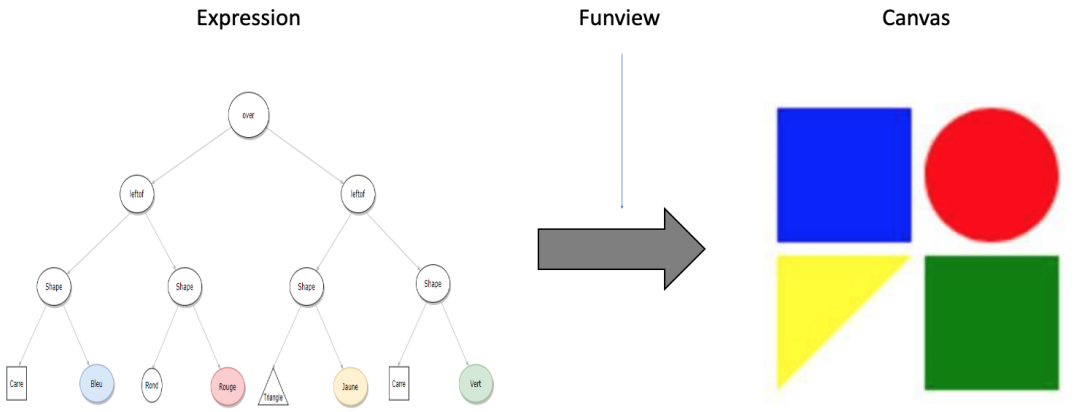


Figure 3.5: Exemple de l'état sous forme AST et sa représentation graphique

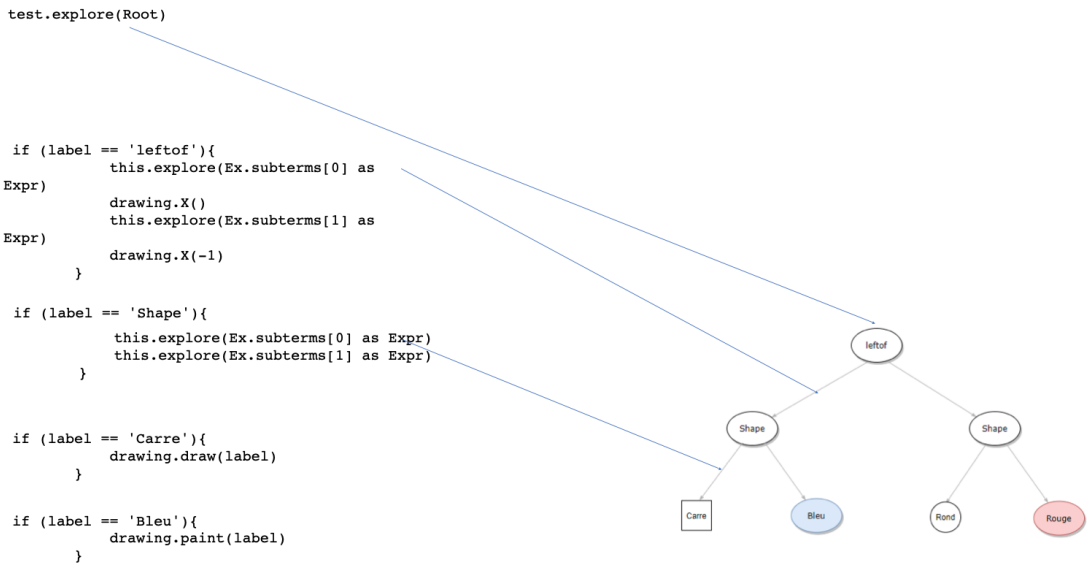


Figure 3.6: Exemple de parcours par la fonction recursive explore

Chapitre 4: Relier FunBlocks et FunView

4.1 Ajouter FunView au projet FunBlocks

Le projet FunBlocks est un projet Typescript avec différents modules. Il a été décidé, en accord avec Dimitri Racordon, de rajouter FunView à FunBlocks à la manière d'un module. Ainsi, dans chaque fichier faisant appelle à FunView, il a fallut importer le module *FUNVIEW*.

Il a fallu également rajouter le canvas HTML à l'ensemble FunBlocks. L'élément HTML canvas a donc été rajouté dans le fichier index.html.

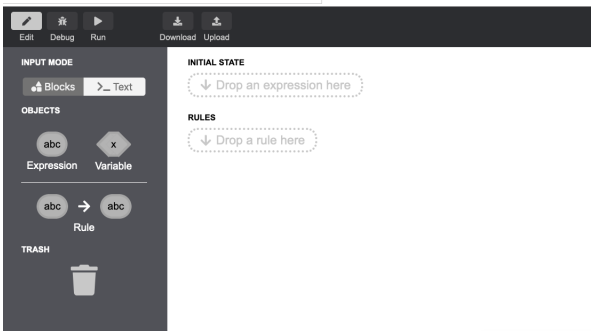


Figure 4.1: On voit ici la zone réservée au canvas HTML au dessus de l'interface FunBlocks

Cette arrangement simpliste a été choisi afin d'éviter de perdre du temps au détriment de la fonctionnalité de l'ensemble.

Durant le développement isolé de FunView, la structure d'un état défini dans le module *AST* de FunBlock a été simulé en créant une version plus simple des expressions. Lors de l'ajout du module *FUNVIEW*, le module AST a été importé dans *FUNVIEW*.

Dès lors, FunView fonctionne dans FunBlocks, mais il faut qu'il soit réactif aux changements faits sur l'état dans FunBlocks.

4.2 Les appels à FunView

FunBlocks utilise la bibliothèque javascript React qui permet de créer une application web sur une seule page. La page réagit aux actions performées sur la dite page. Ces actions produisent ce qu'on appelle un changement d'état. Ainsi, on peut naviguer entre plusieurs *View* grace à certains changements d'état. Dans FunBlocks, ces *View* sont appelée *Workspaces*.

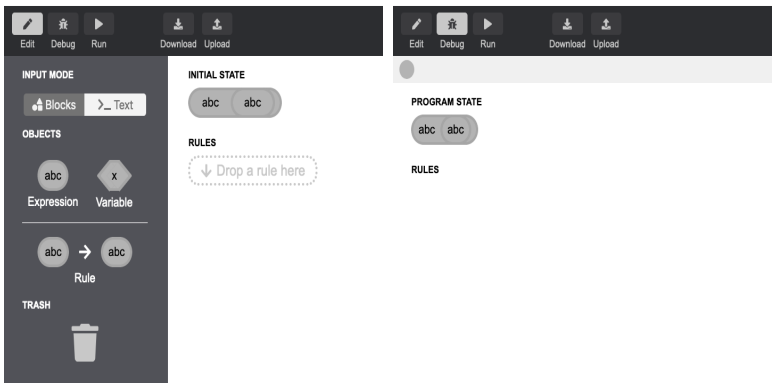


Figure 4.2: Le *Workspace* Edit et le *Workspace* Debug

Ainsi, déterminer quand FunBlock doit appeler FunView se résume à trouver quels changements d'état doivent avoir un impact sur FunView. Ces choix ont été fait de manière arbitraire et peuvent être sujets à de futurs changements. Il a donc été décidé d'appeler FunView lorsque :

- Le *Workspace* Debug est affiché;
- Une règle est appliquée à l'état du programme;
- On change l'état du programme en naviguant dans l'historique.

Les fonctions liées à ces trois actions se trouve dans le dossier Reducers du module *UI*. La première, liée au *Workspace* Debug, se trouve dans le fichier index.ts, lorsque le changement vers le *Workspace* Debug est traité. L'application d'une règle et la navigation dans l'historique se trouvent dans le fichier DebugContext.ts. L'un se trouve dans le cas où un état est poussé et l'autre se trouve dans le cas du changement d'index d'historique.

Dans les trois cas, on instancie les classes DrawingCanvas et Parser. On vide le canvas car on veut éviter de superposer plusieurs formes. Enfin, on appelle la méthode *explore* de la classe Parser avec l'état du programme lors de l'action. Ainsi, FunView est opérationel en collaboration avec FunBlocks.

Chapitre 5: Les difficultés

Dans ce chapitre, je vais décrire les problèmes rencontrés lors de ce projet.

5.1 Typescript

J'avais auparavant déjà travaillé sur un projet en Javascript. Cependant, Typescript est un langage avec ses propres spécificités. Il m'a fallu un temps d'adaptation avant de pouvoir manipuler du code aisément. C'est pourquoi les fonctions LeftOf, RightOf, over et under ont été développées au début. Elles étaient très basiques et sont, à la fin du projet, obsolètes. Elles ont néanmoins servi de terrain d'entraînement. Une fois mes connaissances en Typescript acquises, mon rythme a assez rapidement augmenté et je n'ai pas rencontré de difficultés particulières dans ma capacité à réaliser mes objectifs grâce à Typescript

5.2 Compréhension de Funblocks

FunView est une extension, voire même un simple module de FunBlocks. Dès le début du projet, il dépendait de FunBlocks. Le parser doit utiliser un élément déjà créé afin de comprendre la grammaire. Il a donc fallu devenir familier avec la structure des *Term* et des *Expr*. Comme dit précédement, j'avais déjà eu l'occasion de travailer sur un projet javascript. Cependant, ce projet avait été élaboré par un amateur d'informatique. Il n'avait ainsi pas la structure normalisé d'un vrai projet. Ce fut donc la première fois que je me retrouvais face à un projet avec une telle architecture. Aussi, à force de recherches et d'échanges avec son auteur, Dimitri Racordon, j'ai pu comprendre l'essentiel à la réalisation de mon projet.

5.3 COVID-19

Le projet n'a pas été ralenti par l'arrivée du COVID-19. En effet, la partie coding n'était pas affectée. Cependant, les moyens d'échanges et de retours se sont retrouvés limités. Bien que les réunions Zoom aient permis de continuer à montrer l'avancée du projet, cette méthode avait ses limites. Des échanges en face à face auraient été bien plus rapides.

Chapitre 6: Conclusion

6.1 Le travail fait

Durant ce projet, j'ai créé un service permettant d'avoir une représentation graphique d'un état au sens sémantique du terme. J'ai créé, avec la supervision de l'enseignant et son assistant, une grammaire permettant un parallèle entre un état possiblement abstrait et un arrangement de forme. Cette grammaire possède plusieurs fonctions, bien qu'une seule soit essentielle et peut générer toutes les autres.

6.2 Gains personnels

C'est la première fois que je travaille sur un projet dont les fondations ont déjà été posés et qui avait une solide architecture. J'ai ainsi pu apprendre beaucoup de chose en lisant le travail déjà fait et en analysant sa structure. Pour revenir à la comparaison avec l'ancien projet Javascript auquel j'avais participé, celui-ci ne contenait qu'un seul fichier .html avec le code Javascript inclus dans les balises.
J'ai également appris beaucoup d'astuces sur Typescript ainsi que sur React. Bien que ma partie n'a pas consisté à coder du React, je devais comprendre certaines fonctionnalité afin de pouvoir ajouter ma partie.

6.3 La suite

6.3.1 La grammaire

Un des points ayant le plus de potentiel est la grammaire. En effet, durant ce projet, la question a été dès le départ que peut-on faire? Ce projet a apporté un début de réponse avec ce système de forme et de positionnement. Cependant, ce n'est qu'une réponse parmi pleins d'autres. Un autre parallèle peut être ajouté

entre un état et sa représentation graphique. D'autres idées pourraient émerger et devenir plus pertinentes.

6.3.2 Le Canvas

Actuellement, le canvas est simplement défini au-dessus de l'interface FunBlock. Une meilleure intégration du canvas serait un grand ajout. Une solution alternative au canvas serait également concevable. Il avait été fait mention du format SVG, mais l'utilisation de fichier d'image peut être prometteuse.

6.3.3 FunBlock dépendant de Funview

Une idée de fonctionnalité à rajouter serait de permettre une modification de la représentation graphique. Ce qui aurait un impact sur l'état représenté sous forme de block dans FunBlocks. Cette idée ne serait pas forcément compatible avec un canvas HTML, mais un développement de ce côté serait envisageable.