

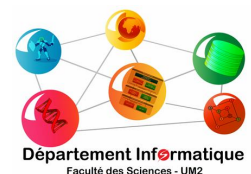
Stibbons

5 juin 2015

Julia Bassoumi - julia.bassoumi@etud.univ-montp2.fr
Florian Galinier - florian.galinier@etud.univ-montp2.fr
Adrien Plazas - adrien.plazas@etud.univ-montp2.fr
Clément Simon - clement.simon@etud.univ-montp2.fr

Encadrant : Michel Meynard

<https://gitlab.info-ufr.univ-montp2.fr/florian.galinier/stibbons.git>



Résumé

Ce projet vise à la création d'un langage de programmation multi-agents pour programmeurs débutants et avancés : le Stibbons. Nous l'avons réalisé en C++ et ses applications utilisent le framework Qt. Deux applications sont proposées pour répondre à deux cas d'utilisation différents : une application graphique permettant de développer des programmes Stibbons et de les voir s'exécuter directement, et une application en ligne de commande simplifiant l'exécution d'un programme et permettant un export régulier de données du modèle exécuté. Ce rapport expose le fonctionnement du langage Stibbons et de ses applications, ainsi que l'organisation que nous avons eu tout au long de la réalisation de ce projet.

Remerciements

Nous tenons tout particulièrement à remercier Michel Meynard pour avoir accepté de nous encadrer et pour son aide apportée tout au long du projet. Merci à lui de nous avoir accordé de son temps.

Nous souhaitons également remercier Jacques Ferber pour sa brillante introduction à la programmation multi-agents, ainsi que Sandrine Maton pour son aide apportée sur les méthodes agiles.

Le langage Stibbons tire son nom de la série de romans *Les Annales du Disque-Monde* de Terry Pratchett.

Table des matières

1	Introduction	10
1.1	Sujet	10
1.1.1	Objectif	10
1.1.2	Systèmes multi-agents	10
1.2	Cahier des charges	10
1.2.1	Fonctionnalités attendues	10
1.2.2	Contraintes	10
2	Analyse de l'existant	12
2.1	Logo	12
2.2	NetLogo	13
2.3	StarLogo	13
3	Analyse des outils	15
3.1	Outils de gestion de projet	15
3.1.1	Méthodes agiles	15
3.1.2	Git	17
3.1.3	Make	18
3.2	Tests unitaires	18
3.2.1	CppUnit	18
3.3	C++11	19
3.4	GDB	19
3.5	Outils d'analyse	19
3.5.1	Flex	19
3.5.2	Bison	20
3.6	Qt	22
3.6.1	Multi plateforme et multi langages	22
3.6.2	Modules	22
3.6.3	Concepts fondamentaux	22
3.6.4	Outils	23
3.6.5	Dessiner avec Qt	23
3.7	Latex	23
3.7.1	Généralités	23
3.7.2	Beamer	24
3.8	JSON Spirit	24
4	Modèle	25
4.1	Les classes d'agents	25
4.2	Les types	26

5	Interprète	30
5.1	Analyseur lexical	30
5.1.1	Jetons	30
5.1.2	Fonctionnement	31
5.2	Analyseur syntaxique	31
5.2.1	Arbre abstrait	32
5.2.2	Fonctionnement	33
5.3	Analyseur sémantique	33
5.3.1	Fonctionnement	33
5.3.2	Fonctionnalités	34
5.3.3	Organisation	34
6	Applications	36
6.1	Application graphique	36
6.1.1	Version 0.1	36
6.1.2	Version 0.2	36
6.1.3	Version 0.3	37
6.1.4	Version 0.4	38
6.1.5	Version 1.0	38
6.2	Application en ligne de commande	38
6.2.1	Utilisation	39
6.2.2	Fonctionnement	39
7	Conclusion	40
A	Documentation	44
A.1	Syntaxe	44
A.1.1	Flex	44
A.1.2	Bison	44
A.2	Propriétés standard	44
A.2.1	Attributs communs à tous les agents	44
A.2.2	Fonctions communes à tous les agents	45
A.2.3	Attributs du monde	46
A.2.4	Fonctions du monde	46
A.2.5	Attributs des tortues	46
A.2.6	Fonctions des tortues	46
A.2.7	Attributs des zones	47
B	Tutoriel	48
B.1	Tutoriel	48
B.1.1	Salut, monde!	48
B.1.2	Les premiers agents	48
B.1.3	Dessiner un carré	48
B.1.4	Répéter	49
B.1.5	Boucler	49
B.1.6	Agents typés	49
B.1.7	Fonctions	49
B.1.8	Couleurs	50
B.1.9	Propriétés d'autres agents et parent	50
B.1.10	Le monde	51
B.1.11	Les zones	51
B.1.12	Directives de monde	52

B.1.13 Messages	52
C Résumés des réunions	54
C.1 27 janvier 2015	54
C.2 3 février 2015	54
C.3 10 février 2015	55
C.3.1 Analyse de l'existant	55
C.3.2 Analyse des outils	55
C.4 24 février 2015	55
C.5 16 mars 2015	56
C.6 17 mars 2015	56
C.7 7 avril 2015	57
D Hierarchical Index	58
D.1 Class Hierarchy	58
E Class Index	61
E.1 Class List	61
F File Index	64
F.1 File List	64
G Class Documentation	66
G.1 stibbons : :Agent Class Reference	66
G.1.1 Detailed Description	66
G.1.2 Constructor & Destructor Documentation	66
G.1.3 Member Function Documentation	68
G.2 stibbons : :Application Class Reference	69
G.2.1 Detailed Description	70
G.2.2 Constructor & Destructor Documentation	70
G.2.3 Member Function Documentation	70
G.3 stibbons : :AskZonesFunction Class Reference	71
G.3.1 Detailed Description	72
G.3.2 Member Function Documentation	72
G.4 stibbons : :Boolean Class Reference	72
G.4.1 Detailed Description	72
G.4.2 Member Function Documentation	72
G.5 BorderType Class Reference	73
G.5.1 Detailed Description	73
G.6 stibbons : :Breed Class Reference	73
G.6.1 Detailed Description	74
G.6.2 Constructor & Destructor Documentation	74
G.6.3 Member Function Documentation	74
G.7 stibbons : :Changeable Class Reference	75
G.7.1 Detailed Description	75
G.7.2 Constructor & Destructor Documentation	75
G.7.3 Member Function Documentation	75
G.8 stibbons : :Color Class Reference	76
G.8.1 Detailed Description	76
G.8.2 Constructor & Destructor Documentation	76
G.8.3 Member Function Documentation	77
G.9 stibbons : :DistanceToFunction Class Reference	80

G.9.1 Detailed Description	81
G.9.2 Member Function Documentation	81
G.10 stibbons : :exit_requested_exception Class Reference	81
G.11 stibbons : :FaceFunction Class Reference	81
G.11.1 Detailed Description	81
G.11.2 Member Function Documentation	82
G.12 stibbons : :FlexScanner Class Reference	82
G.12.1 Detailed Description	82
G.12.2 Constructor & Destructor Documentation	82
G.12.3 Member Function Documentation	82
G.13 stibbons : :Function Class Reference	83
G.13.1 Detailed Description	83
G.13.2 Constructor & Destructor Documentation	83
G.13.3 Member Function Documentation	84
G.14 stibbons : :GenericValue< T > Class Template Reference	85
G.14.1 Detailed Description	85
G.14.2 Member Function Documentation	85
G.15 stibbons : :InboxFunction Class Reference	85
G.15.1 Detailed Description	86
G.15.2 Member Function Documentation	86
G.16 stibbons : :InRadiusFunction Class Reference	86
G.16.1 Detailed Description	86
G.16.2 Member Function Documentation	86
G.17 stibbons : :Interpreter Class Reference	87
G.17.1 Detailed Description	87
G.17.2 Constructor & Destructor Documentation	87
G.17.3 Member Function Documentation	88
G.17.4 Member Data Documentation	91
G.18 stibbons : :InterpreterException Class Reference	91
G.18.1 Detailed Description	91
G.18.2 Constructor & Destructor Documentation	91
G.18.3 Member Function Documentation	92
G.19 stibbons : :InterpreterManager Class Reference	92
G.19.1 Detailed Description	92
G.19.2 Constructor & Destructor Documentation	93
G.19.3 Member Function Documentation	93
G.19.4 Member Data Documentation	93
G.20 stibbons : :Line Class Reference	94
G.20.1 Detailed Description	94
G.20.2 Constructor & Destructor Documentation	94
G.20.3 Member Function Documentation	94
G.21 lineNumberArea Class Reference	96
G.21.1 Detailed Description	96
G.21.2 Constructor & Destructor Documentation	96
G.22 stibbons : :Nil Class Reference	96
G.22.1 Detailed Description	97
G.22.2 Member Function Documentation	97
G.23 stibbons : :Number Class Reference	97
G.23.1 Detailed Description	97
G.23.2 Member Function Documentation	98
G.24 stibbons : :Parser Class Reference	102

G.24.1 Detailed Description	102
G.24.2 Constructor & Destructor Documentation	102
G.24.3 Member Function Documentation	102
G.25 stibbons : :Point Class Reference	102
G.25.1 Detailed Description	103
G.25.2 Constructor & Destructor Documentation	103
G.25.3 Member Function Documentation	103
G.26 stibbons : :PrintFunction Class Reference	107
G.26.1 Detailed Description	107
G.26.2 Member Function Documentation	107
G.27 stibbons : :PrintlnFunction Class Reference	108
G.27.1 Detailed Description	108
G.27.2 Member Function Documentation	108
G.28 stibbons : :RandFunction Class Reference	108
G.28.1 Detailed Description	108
G.28.2 Member Function Documentation	109
G.29 stibbons : :RandomFunction Class Reference	110
G.29.1 Detailed Description	110
G.29.2 Member Function Documentation	110
G.30 stibbons : :Runner Class Reference	110
G.30.1 Detailed Description	111
G.30.2 Constructor & Destructor Documentation	111
G.30.3 Member Function Documentation	111
G.31 stibbons : :SemanticException Class Reference	112
G.31.1 Detailed Description	112
G.31.2 Constructor & Destructor Documentation	112
G.31.3 Member Function Documentation	113
G.32 stibbons : :SendFunction Class Reference	114
G.32.1 Detailed Description	114
G.32.2 Member Function Documentation	114
G.33 stibbons : :SimpleValue< T > Class Template Reference	114
G.33.1 Detailed Description	115
G.33.2 Constructor & Destructor Documentation	115
G.33.3 Member Function Documentation	115
G.34 stibbons : :Singleton< T > Class Template Reference	116
G.34.1 Detailed Description	116
G.34.2 Member Function Documentation	116
G.35 stibbons : :Size Class Reference	117
G.35.1 Detailed Description	117
G.35.2 Constructor & Destructor Documentation	117
G.35.3 Member Function Documentation	118
G.36 stibbons : :SizeFunction Class Reference	118
G.36.1 Detailed Description	119
G.36.2 Member Function Documentation	119
G.37 StibbonsEditor Class Reference	119
G.37.1 Detailed Description	119
G.37.2 Constructor & Destructor Documentation	119
G.37.3 Member Function Documentation	120
G.38 StibbonsHighlighter Class Reference	120
G.38.1 Detailed Description	120
G.38.2 Constructor & Destructor Documentation	120

G.38.3 Member Function Documentation	120
G.39 stibbons : :String Class Reference	121
G.39.1 Detailed Description	121
G.39.2 Member Function Documentation	121
G.40 stibbons : :SyntaxException Class Reference	122
G.40.1 Detailed Description	123
G.40.2 Constructor & Destructor Documentation	123
G.40.3 Member Function Documentation	123
G.41 stibbons : :Table Class Reference	123
G.41.1 Detailed Description	124
G.41.2 Constructor & Destructor Documentation	124
G.41.3 Member Function Documentation	124
G.42 stibbons : :TeleportFunction Class Reference	126
G.42.1 Detailed Description	126
G.42.2 Member Function Documentation	127
G.43 stibbons : :Tree Class Reference	127
G.43.1 Detailed Description	127
G.43.2 Constructor & Destructor Documentation	127
G.43.3 Member Function Documentation	128
G.44 stibbons : :Turtle Class Reference	129
G.44.1 Detailed Description	130
G.44.2 Constructor & Destructor Documentation	131
G.44.3 Member Function Documentation	132
G.45 stibbons : :TurtleInterpreter Class Reference	137
G.45.1 Constructor & Destructor Documentation	137
G.45.2 Member Function Documentation	137
G.46 Type Class Reference	138
G.46.1 Detailed Description	138
G.47 stibbons : :TypeOfFunction Class Reference	138
G.47.1 Detailed Description	138
G.47.2 Member Function Documentation	139
G.48 stibbons : :TypeValue Class Reference	140
G.48.1 Detailed Description	140
G.48.2 Member Function Documentation	140
G.49 stibbons : :UserFunction Class Reference	141
G.49.1 Detailed Description	141
G.49.2 Constructor & Destructor Documentation	141
G.49.3 Member Function Documentation	142
G.50 stibbons : :Value Class Reference	142
G.50.1 Detailed Description	143
G.50.2 Member Function Documentation	143
G.51 stibbons : :Window Class Reference	146
G.51.1 Detailed Description	146
G.51.2 Constructor & Destructor Documentation	146
G.52 stibbons : :World Class Reference	147
G.52.1 Detailed Description	147
G.52.2 Constructor & Destructor Documentation	148
G.52.3 Member Function Documentation	148
G.53 stibbons : :WorldInterpreter Class Reference	152
G.53.1 Detailed Description	152
G.53.2 Constructor & Destructor Documentation	152

G.53.3 Member Function Documentation	153
G.54 stibbons : :WorldPainter Class Reference	154
G.54.1 Detailed Description	154
G.54.2 Constructor & Destructor Documentation	154
G.54.3 Member Function Documentation	155
G.55 stibbons : :WorldView Class Reference	155
G.55.1 Detailed Description	155
G.55.2 Constructor & Destructor Documentation	156
G.55.3 Member Function Documentation	156
G.56 stibbons : :Zone Class Reference	156
G.56.1 Detailed Description	157
G.56.2 Constructor & Destructor Documentation	157
G.56.3 Member Function Documentation	157
H File Documentation	160
H.1 src/interpreter/flex-scanner.h File Reference	160
H.1.1 Detailed Description	160
H.2 src/interpreter/interpreter-exception.h File Reference	160
H.2.1 Detailed Description	161
H.3 src/interpreter/interpreter-manager.h File Reference	161
H.3.1 Detailed Description	161
H.4 src/interpreter/parser.h File Reference	162
H.4.1 Detailed Description	162
H.5 src/interpreter/semantic-exception.h File Reference	162
H.5.1 Detailed Description	162
H.6 src/interpreter/syntax-exception.h File Reference	163
H.6.1 Detailed Description	163
H.7 src/interpreter/tree.h File Reference	163
H.7.1 Detailed Description	164
H.8 src/model/agent.h File Reference	165
H.8.1 Detailed Description	165
H.9 src/model/boolean.h File Reference	165
H.9.1 Detailed Description	166
H.10 src/model/border-type.h File Reference	166
H.10.1 Detailed Description	166
H.11 src/model/breed.h File Reference	166
H.11.1 Detailed Description	167
H.12 src/model/changeable.h File Reference	167
H.12.1 Detailed Description	167
H.13 src/model/color.h File Reference	167
H.13.1 Detailed Description	168
H.14 src/model/function.h File Reference	168
H.14.1 Detailed Description	168
H.15 src/model/line.h File Reference	168
H.15.1 Detailed Description	169
H.16 src/model/nil.h File Reference	169
H.16.1 Detailed Description	169
H.17 src/model/number.h File Reference	170
H.17.1 Detailed Description	170
H.18 src/model/point.h File Reference	170
H.18.1 Detailed Description	170
H.19 src/model/simple-value.h File Reference	171

H.19.1 Detailed Description	171
H.20 src/model/singleton.h File Reference	171
H.20.1 Detailed Description	171
H.21 src/model/size.h File Reference	172
H.21.1 Detailed Description	172
H.22 src/model/standard-function.h File Reference	172
H.22.1 Detailed Description	173
H.23 src/model/string.h File Reference	173
H.23.1 Detailed Description	173
H.24 src/model/turtle.h File Reference	174
H.24.1 Detailed Description	174
H.25 src/model/type-value.h File Reference	175
H.25.1 Detailed Description	175
H.26 src/model/user-function.h File Reference	175
H.26.1 Detailed Description	175
H.27 src/model/value.h File Reference	176
H.27.1 Detailed Description	176
H.28 src/model/world.h File Reference	176
H.28.1 Detailed Description	177
H.29 src/model/zone.h File Reference	177
H.29.1 Detailed Description	178
H.30 src/qt/line-number-area.h File Reference	178
H.30.1 Detailed Description	178
H.31 src/qt/runner.h File Reference	179
H.31.1 Detailed Description	179
H.32 src/qt/stibbons-highlighter.h File Reference	179
H.32.1 Detailed Description	180
H.33 src/qt/window.h File Reference	180
H.33.1 Detailed Description	180
H.34 src/qt/world-painter.h File Reference	181
H.34.1 Detailed Description	181
H.35 src/qt/world-view.h File Reference	181
H.35.1 Detailed Description	181
I Listing	183
I.1 Flex	183
I.2 Bison	188
I.3 CppUnit	203
I.4 Json Spirit	205
Bibliographie	208

Chapitre 1

Introduction

1.1 Sujet

1.1.1 Objectif

Le projet Stibbons vise à créer un interprète d'un dérivé de Logo, un langage de programmation permettant l'animation d'un agent mobile, dit « tortue » (cf. 2.1). Nous avons choisi de développer un langage multi-agents, à l'instar de NetLogo ou StarLogo (cf. 2.2 et 2.3), rendant ainsi l'objectif de ce projet double : d'une part la réalisation d'un interprète capable d'analyser du code écrit dans un dérivé de Logo, d'autre part la réalisation d'une application graphique capable de représenter l'évolution des agents.

1.1.2 Systèmes multi-agents

Les systèmes multi-agents sont une approche de l'intelligence artificielle visant à faciliter la résolution d'un problème en le découplant en plusieurs sous-problèmes. Ainsi, plutôt que de chercher à développer une intelligence unique complexe capable de résoudre le problème, l'approche multi-agents vise plutôt à créer des multitudes d'intelligences capables de résoudre une petite partie du problème, et de compter sur l'intelligence collective émergente pour voir apparaître la solution au problème [Ferber, 1995].

Ainsi, on peut prendre en exemple les fourmis qui, bien que n'ayant individuellement qu'une capacité limitée, sont capables de survivre grâce à la synergie de leurs colonies.

1.2 Cahier des charges

1.2.1 Fonctionnalités attendues

Bien que la méthode de développement utilisée ait fait apparaître de nombreuses fonctionnalités souhaitables au fur et à mesure du projet, un certain nombre d'entre elles nous ont semblé indispensables dès le début :

- chaque agent devait être capable de communiquer avec les autres agents ;
- chaque agent devait pouvoir modifier d'une certaine façon le monde ;
- le langage devait comprendre des structures des langages de programmation impérative modernes, telles que des conditionnelles, des boucles, des fonctions, etc. ;
- l'utilisateur devait pouvoir voir le monde où évoluent ces agents.

1.2.2 Contraintes

Nous avons dès le début isolé un certain nombre de contraintes que nous souhaitons pour ce projet :

- l'utilisation d'un langage non interprété, de type C ou C++, pour son développement ;
- chaque agent devait évoluer parallèlement aux autres agents ;
- l'utilisation d'une méthode agile pour développer le projet.

Chapitre 2

Analyse de l'existant

2.1 Logo

Le Logo est un langage de programmation apparu dans les années 60 dont l'objectif était alors de permettre à des personnes possédant peu de connaissances en informatique et en programmation (des enfants par exemple) de découvrir ce domaine de manière ludique et interactive. Le langage permettait de contrôler une tortue (un robot) capable d'avancer, de tourner, et d'abaisser un crayon ou un feutre pour dessiner sur une feuille placée au sol. Les instructions entrées permettaient ainsi de tracer des formes, permettant une représentation très visuelle du code (la tortue physique est remplacée dans les implémentations modernes par une tortue virtuelle).

Le langage Logo en lui-même est un dérivé du Lisp (il est d'ailleurs parfois nommé « Lisp sans parenthèses ») et possède deux types de données : les mots (chaînes de caractères) et les listes.

Du fait du public visé, les instructions de base (par exemple **forward**, **left**, **pendown**, etc.) et les structures du type procédures, boucles ou conditionnelles sont écrites de façon à être clairement explicites (cf. 2.1, 2.2 et 2.3). Cependant, comme expliqué dans l'article [Pea, 1987], des études sur Logo ont montré que les enfants, hormis quelques exceptions, n'arrivent pas à créer un programme entier et codent « ligne par ligne » ce qui les empêche de créer un modèle complexe et de cerner l'ensemble de la syntaxe de Logo.

```
to <nom de la procédure> :<paramètre>
  <instructions>
  output <valeur à retourner>
end
```

Listing 2.1 – Procédure en Logo

```
repeat <nb fois> [liste d'instructions]
```

Listing 2.2 – Boucle en Logo

```
if <test> [liste d'instructions si vrai]
ifelse <test> [liste d'instructions si vrai] [liste d'instructions si faux]
```

Listing 2.3 – Conditionnelles en Logo

Les instructions amènent la tortue à se déplacer suivant une certaine distance et un certain angle. On l'oriente ainsi suivant des coordonnées polaires.

2.2 NetLogo

NetLogo est un environnement de modélisation programmable développé à la Northwestern University (ref. [Wilensky, 1999]) et permettant de simuler et d’observer des phénomènes naturels et sociaux au fil du temps. Il permet de donner des instructions à des agents et d’observer leur évolution et les connexions inter-agents au niveau micro (individu par individu) comme au niveau macro (monde global). Il est aussi utilisé dans de nombreux domaines de recherche comme l’économie, la biologie, la physique, la chimie, etc. et de nombreux articles ont été publiés à son sujet.

Au niveau du programme en lui-même, NetLogo est composé de 3 onglets :

- l’onglet info : la documentation du code ;
- l’onglet code : le code permettant de créer le modèle du monde ainsi que le comportement des agents y sont implémentés ;
- l’onglet interface :
 - la partie « observation » qui est représentée par une fenêtre où l’on verra notre modèle évoluer dans le temps (le rendu pouvant être effectué en 2D comme en 3D) ;
 - la partie « construction » où l’on peut ajouter des widgets interagissant avec le code.

Les widgets de la partie construction permettent d’interagir avec des procédures ou des variables du code. Par exemple, l’ajout d’un slider « nb_population » pourrait permettre de définir le nombre de tortues à créer sans modifier le code. On pourrait également y faire apparaître des boutons pour démarrer ou arrêter des procédures, des graphes pour observer des variations, des interrupteurs pour gérer des variables globales, des notes, etc.

On peut également contrôler le temps, ralentir pour mieux observer, accélérer pour voir ce que produit le modèle.

NetLogo permet donc une interaction très rapide entre le code et le rendu graphique, permettant au développeur de « jouer » avec le modèle en modifiant facilement certaines conditions et donc d’ajuster immédiatement le code comme il le souhaite.

Une riche documentation et de nombreux tutoriels sont fournis sur le site officiel du langage, ce qui permet une prise en main simple et ludique.

NetLogo est un logiciel libre et open source, sous licence GPL. Il fonctionne sur la machine virtuelle Java, et est donc opérable sur de nombreuses plateformes (Mac, Windows, Linux, etc.). D’après son site officiel, NetLogo est décrit comme la prochaine génération de langages de modélisation multi-agents, tout comme StarLogo.

2.3 StarLogo

Tout comme NetLogo, StarLogo est un environnement de modélisation programmable permettant de simuler et d’observer des phénomènes naturels et sociaux au fil du temps. Ils ont les mêmes objectifs d’étude, d’éducation et de « programmation facile » ainsi qu’un aperçu direct du rendu (ref. [Resnick et al., 2008]).

Là où StarLogo diffère est qu’il n’est pas nécessaire de connaître une seule ligne de code. En effet, StarLogo se base sur un principe de bouton ; pour une partie de code donnée, un bouton y correspond. Les boutons peuvent être liés entre eux, permettant de créer des actions plus complexes. Tout bouton peut être positionné dans une page, qui correspond aux différents « environnements » du modèle : le monde, un certain type de tortues, les patches, etc. Par exemple, pour créer douze tortues lors d’une initialisation globale du modèle, il faut se positionner sur la page `setup`, y placer le bouton `setup` attaché du bouton `create Turtles - num`, puis modifier le `num` en 12.

StarLogo est composé de deux fenêtres :

- la fenêtre code : c’est ici que les boutons sont placés dans les différentes pages pour générer le code ;

— la fenêtre vue : on y aperçoit les résultats du code généré, le rendu pouvant être en 2D comme en 3D.

Au niveau historique, StarLogo avait d’abord été créé pour Mac lors de la première version, puis après plusieurs années, une version pour tout type d’environnement a vu le jour et la version uniquement pour Mac fut rebaptisée MacStarLogo. Plusieurs versions sont apparues mais c’est la version 2.1 de 2004 qui reste la plus récente.

StarLogo est disponible sous la même licence et pour le même environnement d’exécution que son confrère NetLogo : c’est un logiciel libre sous licence GPL qui fonctionne sur la machine virtuelle Java, d’où son gain de portabilité après la version MacStarLogo.

Chapitre 3

Analyse des outils

3.1 Outils de gestion de projet

3.1.1 Méthodes agiles

Pour réaliser ce projet, nous avons choisi d'utiliser la méthode agile SCRUM. Nous nous sommes principalement servi des cours de gestion de projet de ce semestre, en particulier des interventions de Sandrine Maton. Utiliser une méthode agile permet d'avoir un rendu fonctionnel à chaque fin de sprint, et donc de s'assurer d'avoir un rendu final. De plus, ces méthodes étant de plus en plus populaires dans le milieu des entreprises, nous souhaitons donc l'expérimenter.

La méthode Scrum est une méthode itérative, dont chaque itération est nommée « sprint ». Un sprint se décompose en trois parties : une réunion initiale, où les objectifs du sprint sont définis et les tâches réparties, le sprint en lui-même, durant lequel le développement a lieu, et une réunion pour faire le point sur le sprint écoulé.

Backlog initial

Le sprint 0 correspond à la première période du projet, durant laquelle nous avons fait notre analyse de l'existant et des outils. C'est également durant ce sprint que nous avons choisi les différentes fonctionnalités que nous voulions voir dans le projet. Pour chacune d'entre elles, nous écrivons un scénario utilisateur, une courte description de ce que l'utilisateur doit pouvoir faire via une fonctionnalité.

Une fois ces fonctionnalités définies, nous avons priorisé celles-ci et établi le « backlog » : la feuille de route de notre projet (cf. Table 3.1).

id	Scénario utilisateur	Priorité	Tests	Estimation	Sprint	Statut	Temps réel
1	L'utilisateur écrit du code dans un éditeur	200					
2	L'utilisateur importe du code dans le logiciel	1100	Importer du code Stibbons depuis un fichier, vérifier que le code obtenu est bien identique à celui du fichier.	4h	1	Fini	1h
3	L'utilisateur visualise les rapports d'erreurs du code	400					
4	L'utilisateur visualise l'évolution du modèle	1400	Vérifier que l'interprétation d'instructions données fait bien évoluer comme prévu la tortue dans son environnement.	24h	1	Fini	70h
5	L'utilisateur modifie la vitesse (pause, pas à pas, parallèle)	700					
6	Le « dieu-tortue » interprète le code de l'utilisateur	1500	Lancer l'interprétation pour : repeat 4 fd 1 rt 90 (suivant syntaxe) ainsi que pour du code avec des erreurs : repeat 4 ... par exemple ou repeat 4 fd 1 rt	12h	1	Fini	32h

TABLE 3.1: Le backlog au début du sprint 2

id	Scénario utilisateur	Priorité	Tests	Estimation	Sprint	Statut	Temps réel
7	L'utilisateur crée une nouvelle tortue (avec un code)	600	Lancer l'interprétation pour : create-turtle et observer une nouvelle tortue apparaître dans l'interface graphique.	7,5h	2		
8	Les tortues s'exécutent en parallèles	1200	Lancer l'interprétation pour deux tortues d'un bout de code et observer l'exécution parallèle via des écritures dans le terminal (Je suis tortue 1 et Je suis tortue 2 par exemple)	42,5h	2		
9	Les tortues communiquent directement entre elles	900					
10	Une tortue se déplace dans l'environnement	1300	Ecriture d'instructions simples : repeat 4 fd 1 rt 90 (suivant syntaxe) - Renvoi de la position de la tortue après chaque déplacement : where_am_i(); (suivant syntaxe)	16h	1	Fini	24h
11	Les tortues communiquent avec les zones de l'environnement	1000					
12	L'utilisateur exporte le code	500					
13	L'utilisateur exporte le modèle	300					
14	L'utilisateur ajoute une entrée	100					
15	L'utilisateur remet à zéro l'environnement	800					
16	L'utilisateur utilise des variables dans le code	1275	Ecrire a = 90 fd a et observer la tortue qui avance.	12h	2		
17	L'utilisateur définit des fonctions personnalisées dans le code	1250	Ecrire fonction f () fd 90 f () et observer la tortue qui avance.	23,5h	2		
18	Les tortues communiquent via l'environnement	950					
19	L'utilisateur utilise des conditionnelles	550	Ecrire if(false) fd 90 et observer que la tortue ne bouge pas. Refaire le même test avec if(true) et observer que la tortue bouge.	3,5h	2		
20	L'utilisateur utilise des boucles	575	Ecrire pd repeat 4 fd 40 rt 90 et observer que la tortue dessine un carré.	7h	2		

TABLE 3.1: Le backlog au début du sprint 2

Lors de la réunion de début de sprint, les fonctionnalités à ajouter sont choisies et pour chacune d'entre elles une description du test à effectuer afin de la valider était ajoutée au backlog. Nous faisons également une estimation du nombre d'heures nécessaires à chacune des nouveautés. À la fin d'un sprint, nous estimions le temps passé sur chaque tâche et nous faisons le point sur notre avancée dans le projet.

Backlog final

Sur le backlog final (cf. Table 3.2), nous pouvons voir les ajouts effectués, le changement de statut des fonctionnalités réalisées, et le temps réel passé à les rendre utilisables.

id	Scénario utilisateur	Priorité	Tests	Estimation	Sprint	Statut	Temps réel
1	L'utilisateur écrit du code dans un éditeur	200	L'utilisateur écrit du code dans un éditeur intégré	20h	5	Fini	19h
2	L'utilisateur importe du code dans le logiciel	1100	Importer du code Stibbons depuis un fichier, vérifier que le code obtenu est bien identique à celui du fichier.	4h	1	Fini	1h
3	L'utilisateur visualise les rapports d'erreurs du code	400	Exécuter pd 50 et constater une erreur.	8h	4	Fini	8h
4	L'utilisateur visualise l'évolution du modèle	1400	Vérifier que l'interprétation d'instructions données fait bien évoluer comme prévu la tortue dans son environnement.	24h	1	Fini	70h
5	L'utilisateur modifie la vitesse (pause, pas à pas, parallèle)	700	Faire varier la vitesse et observer le changement dans l'exécution des tortues.	12h	4	Fini	14h

TABLE 3.2: Le backlog à la fin du sprint 5

id	Scénario utilisateur	Priorité	Tests	Estimation	Sprint	Statut	Temps réel
6	Le « dieu-tortue » interprète le code de l'utilisateur	1500	Lancer l'interprétation pour : repeat 4 fd 1 rt 90 (suivant syntaxe) ainsi que pour du code avec des erreurs : repeat 4 ... par exemple, ou repeat 4 fd 1 rt	12h	1	Fin	32h
7	L'utilisateur crée une nouvelle tortue (avec un code)	600	Lancer l'interprétation pour : create-turtle et observer une nouvelle tortue apparaître dans l'interface graphique.	7,5h	2	Fin	4h
8	Les tortues s'exécutent en parallèles	1200	Lancer l'interprétation pour deux tortues d'un bout de code et observer l'exécution parallèle via des écritures dans le terminal (Je suis tortue 1 et Je suis tortue 2 par exemple)	42,5h	2	Fin	40h
9	Les tortues communiquent directement entre elles	900	Ecrire send(t,"Je suis là") avec t une tortue qui a pour code : m = recv() if (m == "Je suis là") fd 50 et observer la tortue avancer	30h	3	Fin	16h
10	Une tortue se déplace dans l'environnement	1300	Ecriture d'instructions simples : repeat 4 fd 1 rt 90 (suivant syntaxe) - Renvoi de la position de la tortue après chaque déplacement : where_am_i(); (suivant syntaxe)	16h	1	Fin	24h
11	Les tortues communiquent avec les zones de l'environnement	1000	Ecrire if(zone.color == red) color = blue sur une zone de couleur rouge et observer la tortue changer de couleur.	30h	3	Fin	12h
12	L'utilisateur exporte le code	500	L'utilisateur sauvegarde son code dans un fichier externe	1h	5	Fin	1h
13	L'utilisateur exporte le modèle	300	Sauvegarder le modèle et observer les données en sortie (JSON)	30h	4	Fin	22h
14	L'utilisateur ajoute une entrée	100					
15	L'utilisateur remet à zéro l'environnement	800	Remettre à zéro après une exécution et observer que le monde est vierge.	20h	4	Fin	23h
16	L'utilisateur utilise des variables dans le code	1275	Ecrire a = 90 fd a et observer la tortue qui avance.	12h	2	Fin	3h
17	L'utilisateur définit des fonctions personnalisées dans le code	1250	Ecrire function f () fd 90 f () et observer la tortue qui avance.	23,5h	2	Fin	11h
18	Les tortues communiquent via l'environnement	950	Ecrire broadcast(20,"Je suis là") et dans une autre tortue dans le rayon avec le code m = recv() if (m == "Je suis là") fd 50 et observer la tortue avancer.	10h	3	Fin	2h
19	L'utilisateur utilise des conditionnelles	550	Ecrire if(false) fd 90 et observer que la tortue ne bouge pas. Refaire le même test avec if(true) et observer que la tortue bouge.	3,5h	2	Fin	3,5h
20	L'utilisateur utilise des boucles	575	Ecrire pd repeat 4 fd 40 rt 90 et observer que la tortue dessine un carré.	7h	2	Fin	3,5h
21	L'utilisateur définit des fonctions avec paramètres	1225	Ecrire function f(a) fd a f(90) et observer la tortue avancer.	10h	3	Fin	16h
22	L'utilisateur instancie des agents avec paramètres	560	Ecrire agent wolf (a) fd a new wolf (50) et observer la nouvelle tortue avancer.	10h	3	Fin	16h
23	L'utilisateur modifie la couleur d'un agent (tortue ou zone)	450	Ecrire new agent color = red et observer la nouvelle tortue rouge.	3h	3	Fin	2h
24	La tortue accède aux données de son parent	50	Taper parent.color= blue et observer le parent devenir bleu	1h	5	Fin	1h
25	L'utilisateur peut stocker des valeurs dans une table	540	On stocke le retour dans t= {1,2}	10h	4	Fin	9h
26	L'utilisateur parcourt un tableau	530	Faire foreach(f : {1,2}) println(f)	4h	5	Fin	4h
27	L'utilisateur lance l'application sans interface graphique	250	Lancer l'application et regarder les fichiers d'exportations	16h	5	Fin	15h

TABLE 3.2: Le backlog à la fin du sprint 5

3.1.2 Git

Afin de permettre une meilleure gestion du projet (travail parallèle, gestion de bugs, etc.) nous avons décidé d'utiliser le gestionnaire de version Git ainsi que la forge GitLab mise à la disposition des étudiants par le SIF. La prise en main fut facile, les membres du groupe ayant pratiquement tous déjà utilisé cet outil. Nous avons également utilisé le système de suivi de bugs intégré à Gitlab (cf. Figure 3.1).




<input type="checkbox"/> ID dans l'export	CLOSED
#29 1	updated 11 days ago
<input type="checkbox"/> Permettre au monde d'initialiser les zones	CLOSED
#28 1	updated 11 days ago
<input type="checkbox"/> Les appels de fonction récursifs font déborder la pile	
#27 improvement	updated 22 days ago
<input type="checkbox"/> Fonctions standard: un paramètre d'un mauvais type fait planter l'application	CLOSED
#26 assigned to  Plazas Adrien 1	updated 24 days ago
<input type="checkbox"/> String: supprimer les guillemets	CLOSED
#25 1	updated 23 days ago
<input type="checkbox"/> Utiliser pointeurs intelligents	CLOSED
#24 1 improvement	updated about a month ago
<input type="checkbox"/> Application "headless"	CLOSED
#23 1 improvement	updated 11 days ago
<input type="checkbox"/> Sérialisation / désérialisation	CLOSED
#22 1 improvement	updated 23 days ago
<input type="checkbox"/> Point : fuite mémoire	CLOSED
#21 assigned to  Plazas Adrien 1 bug model	updated 23 days ago
<input type="checkbox"/> Monde : ajouter des fonctions prédéfinies	CLOSED
#20 1 improvement	updated about a month ago
<input type="checkbox"/> Monde : ajouter des couleurs prédéfinies	CLOSED
#19 assigned to  Plazas Adrien 1 improvement	updated about a month ago

FIGURE 3.1 – Système de suivi de bugs de Gitlab

Notre organisation des branches fut la suivante :

- la branche master devait contenir une version fonctionnelle compilable ;
- des branches de développement étaient créées pour chaque nouvelle fonctionnalité, et n'étaient fusionnées sur la branche master que lorsqu'elles étaient pleinement fonctionnelles ;
- des branches spécifiques à chaque version ont été dérivées de master lors de l'officialisation desdites versions.

Nous utilisons également Gitg, pour avoir une meilleure vue de l'état de notre dépôt (cf. Figure 3.2).

3.1.3 Make

Make est un utilitaire de construction de fichiers qui nous a été utile tout au long du projet pour construire les applications (stibbons et stibbons-cli), les tests unitaires, ou encore la documentation L^AT_EX du projet. Il est également utile à l'installation des applications.

Il fonctionne en laissant l'utilisateur définir des règles de construction de fichiers en définissant les commandes nécessaires à celle-ci ainsi que les fichiers dont elle dépend.

Make est alors appelé à construire une cible (un fichier ou non), construisant son arbre de dépendances, vérifiant l'existence de ces dernières, les construisant ou reconstruisant au besoin. Ainsi Make permet d'éviter les compilations ou recompilations inutiles, accélérant et automatisant la construction de logiciels ou de documents.

3.2 Tests unitaires

3.2.1 CppUnit

CppUnit (ref. [Navarro, 2003]) est un outil permettant d'organiser des tests unitaires. On définit une classe de tests avec les attributs leurs étant nécessaires et des méthodes les réalisant. Cette classe est ensuite enregistrée dans le registre des tests pour être exécutée (cf. I.3).

CppUnit possède des macros pour simplifier les tests comme :

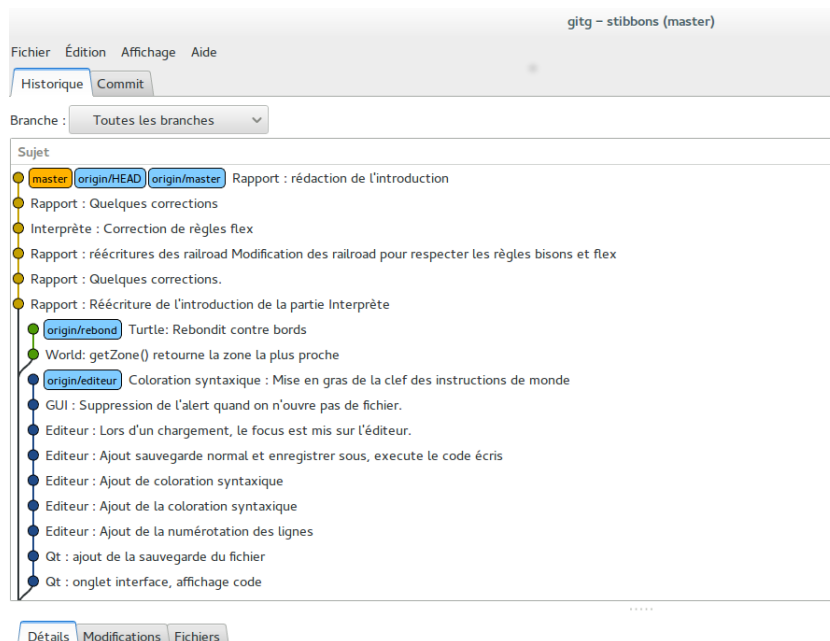


FIGURE 3.2 – Vue des branches dans Gitg

```
— CPPUNIT_ASSERT(test) ;
— CPPUNIT_ASSERT_EQUAL(v1,v2) ;
— etc.
```

Lors de l'exécution, il indique combien de tests ont réussi et échoué. Son écriture est simple, et son organisation permet une lecture rapide des tests.

3.3 C++11

C++11 est la version du standard C++ que nous avons choisi d'utiliser. En effet, cette version a vu apparaître de nombreux ajouts, comme par exemple les `std::thread` et les `std::mutex`, qui nous ont permis de gérer le multithreading sans avoir à nous soucier de leur implémentation (on n'a ainsi pas de problèmes de portabilité liés à ceci).

En outre, d'autres ajouts de C++11, tel que l'inférence de type (avec le mot-clef `auto`) ou les boucles traversant une collection (`for (auto i: collection)`), nous ont permis une écriture plus souple et plus moderne du code.

3.4 GDB

Le « GNU Project Debugger » (ref. [Stallman, 1988]) est un programme de débogage permettant notamment de tracer les appels de fonctions et de spécifier des conditions d'arrêts lors de l'exécution d'un programme. Il peut donc permettre d'avancer pas à pas dans un programme et de repérer l'endroit où un bogue se situe. Il est très utilisé pour résoudre les erreurs de segmentation grâce à sa précision de traçage : la ligne exacte de l'erreur est fournie.

3.5 Outils d'analyse

3.5.1 Flex

Flex est une version libre de l'analyseur lexical Lex (ref. [Paxson, 2014]). Il est généralement associé à l'analyseur syntaxique GNU Bison, la version GNU de Yacc. Il lit les fichiers d'entrée

donnés pour obtenir la description de l'analyseur à générer. La description est une liste de paires d'expressions rationnelles et de code C, appelées règles.

Un fichier Flex est composé de plusieurs parties. La première contient une partie optionnelle de définition, encadrée par les symboles `%{ %}` (cf. Listing 3.1), ainsi que des options pour Flex (cf. Listing 3.2). La seconde partie est une partie obligatoire de règles, commençant par `%%` (cf. Listing 3.3), tandis que la dernière partie est une nouvelle partie optionnelle, débutée par `%%`, pouvant contenir des fonctions C/C++ définies par l'utilisateur (cf. Listing 3.4).

```
%{
    int yyFlexLexer::yywrap() {
        return 1;
    }
}%
```

Listing 3.1 – Partie définition d'un fichier Flex

```
%option c++
%option nodefault
```

Listing 3.2 – Options de Flex

```
%%
#([a-f0-9]{6}|[a-f0-9]{3}) {
    pyylval->v=make_shared<stibbons::Color>(yytext);
    return yy::parser::token::COLOR;
}
```

Listing 3.3 – Partie règles de Flex

```
%%
int main() {
    // ...
}
```

Listing 3.4 – Partie fonctions de Flex

La transformation en code C++ se fait par compilation via l'appel à l'application `flex -+ exemple.1+`. La fonction d'analyse ainsi générée se nomme `yylex()`. Il faut par la suite penser à compiler le programme en liant la bibliothèque Flex via le flag `-lfl`.

3.5.2 Bison

GNU Bison est une version de Yacc (ref. [bis, 2015]), un outil d'analyse syntaxique (cf. 5.2). Il génère un analyseur syntaxique ascendant utilisant un automate à pile (dérivation à droite, remplaçant le symbole non terminal le plus à droite).

Son fonctionnement est le suivant : à chaque règle de grammaire, on associe des actions (instructions d'un langage). L'analyseur généré essaie de reconnaître un mot du langage défini par la grammaire et exécute les actions pour chaque règle reconnue.

Comme pour Flex, un fichier Bison est composé de trois parties : la première partie, facultative, contient une liste de définition C/C++, d'options Bison ainsi que de définition de jetons (cf. Listing 3.5), la seconde partie, contenue entre `%%`, contient les règles (cf. Listing 3.6) et une dernière optionnelle de C/C++.

```
%skeleton "lalr1.cc"
%defines
%locations
```

```

%parse-param { stibbons::FlexScanner &scanner }
%parse-param { stibbons::TreePtr t }
%parse-param { stibbons::TablePtr w }
%lex-param { stibbons::FlexScanner &scanner }

%code requires {
    namespace stibbons {
        class FlexScanner;
    }

    std::string toString(const int& tok);
}

%token IF "if"
%token ELSE "else"
%token FCT "function"

```

Listing 3.5 – Definition C++ en bison

```

%%

//Storage of conditionnal expression
selection : IF expr statement
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::IF, nullptr);
    t1->addChild($2);
    t1->addChild($3);
    t1->setPosition({@1.begin.line, @1.begin.column});
    $$ = t1;
}
| IF expr statement ELSE statement
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::IF, nullptr);
    t1->addChild($2);
    t1->addChild($3);
    t1->addChild($5);
    t1->setPosition({@1.begin.line, @1.begin.column});
    $$ = t1;
};

%%

```

Listing 3.6 – Règles de grammaire en bison

Les différents types de jeton sont déclarés via l'instruction `%token NOM_DU_JETON` dans la première partie. On peut également définir le type C/C++ de la valeur du jeton via l'instruction `%union`, ou en redéfinissant la macro `YYSTYPE` (cf. 3.7 et 3.8).

```

%union{
    stibbons::Value* v;
    stibbons::Tree* tr;
}

```

```
| }
```

Listing 3.7 – Exemple du type des valeurs des jetons avant la version 0.3

```
| #define YYSTYPE struct { stibbons::ValuePtr v; stibbons::TreePtr tr;  
|     int tok; }
```

Listing 3.8 – Exemple du type des valeurs des jetons en version 1.0

Si on veut un analyseur syntaxique en C++, il faut utiliser un squelette de parseur C++ en utilisant soit l'option bison `-skeleton=lalr1.cc`, soit en utilisant la directive `%skeleton "lalr1.cc"`.

La transformation en code C++ se fait par compilation via l'appel à l'application `bison -ydt exemple.y+`. La fonction d'analyse ainsi générée se nomme `yyparse()` et fait appel en interne à `yylex()`.

3.6 Qt

Qt est un framework d'application multi-plateformes écrit en C++ principalement utilisé pour la création d'interfaces graphiques.

3.6.1 Multi plateforme et multi langages

Qt est utilisable sur de nombreuses plateformes telles que Windows, Mac OS X, X11, Wayland, Android ou iOS.

De plus, bien que Qt soit développé en C++, ce n'est pas le seul langage depuis lequel il est utilisable, on trouve notamment des liaisons pour Python, JavaScript, Go, Ruby, Haskell ou encore Ada.

3.6.2 Modules

Qt comprend de nombreux modules afin d'aider tant que possible le développement d'applications. On peut particulièrement citer :

Core : une implémentation des types de base (`QString`, etc.), conteneurs, parallélisme, entrées-sorties, système d'événements, etc. ;

Widgets : des widgets pour le développement d'interfaces graphiques ;

Network : support de divers protocoles réseau (TCP, UDP, HTTP, SSL, etc.) ;

Multimedia : lecture audio et vidéo ;

SQL : accès à des bases de données comprenant SQL ;

WebKit : moteur de rendu HTML ;

3.6.3 Concepts fondamentaux

Widgets et layouts

Qt propose un système de widgets complet et puissant. Il propose de nombreux widgets classiques tels que des boutons, des choix à puce, des onglets, des étiquettes, des images, etc.

Pour Qt, tout widget peut contenir des enfants et les arranger selon une disposition qui lui est affectée. Un widget n'ayant pas de widget parent sera considéré comme étant une fenêtre. Son fonctionnement est ainsi assez différent de son concurrent Gtk+.

Signaux et slots

Qt propose également un système de signaux et de slots permettant d'implémenter le modèle observateur de manière efficace.

Ainsi un widget peut émettre des signaux contenant ou non des données (par exemple, pour signaler le changement de valeur d'une entrée) et un autre widget peut réceptionner ce signal dans un de ses slots, l'exécutant alors.

Une extension à C++

Qt propose une extension à C++ : il y ajoute des mots-clés pour permettre de simplifier la définition d'objets descendants de `QObject`, tout particulièrement en spécifiant un ensemble de slots d'une certaine visibilité. Ainsi lors de la déclaration d'une classe, il est possible de déclarer une liste de slots publics ou de signaux en les précédant des mentions `public slots` et `signals`, respectivement. Le Meta-Object Compiler de Qt est alors utilisé pour convertir ces définitions en C++ classique à la compilation.

Qt propose également un système permettant d'embarquer des ressources (images, sons, etc.) directement dans le binaire produit via la définition d'un fichier de collection de ressources (`.qrc`) et l'utilisation d'un Ressource Compiler.

qmake est un générateur de Makefiles permettant de simplifier l'utilisation du Meta-Object Compiler et du Ressource Compiler.

3.6.4 Outils

Qt permet de développer des interfaces dans un langage déclaratif basé sur XML. Pour utiliser ce puissant atout, il est conseillé d'utiliser Qt Designer, une application permettant de dessiner des interfaces graphiques de manière intuitive, en plaçant manuellement les widgets les uns dans les autres.

Qt Creator va encore plus loin puisqu'il est un IDE assez complet centré sur le développement d'applications avec Qt. En effet, il permet de gérer des projets, d'éditer du code, de concevoir des interfaces comme Qt Designer et même de créer des slots de manière graphique. Bien entendu il permet également de compiler, exécuter et déboguer le projet.

3.6.5 Dessiner avec Qt

Il est possible de réaliser des widgets personnalisés en redéfinissant la méthode `paintEvent` d'un widget et en utilisant la classe `QPainter` pour dessiner sur le widget.

Une instance `QPainter` est liée à un widget et propose diverses méthodes permettant de dessiner :

- des lignes ;
- des arcs de cercle ;
- des polygones ;
- des ellipses ;
- des images ;
- du texte.

3.7 Latex

3.7.1 Généralités

L^AT_EX est un langage de rédaction de document qui force à avoir une structure sur la forme et le contenu. Il est notamment utilisé lors d'écritures de documents scientifiques car son écriture de contenus complexes (équations, bibliographie, etc.) se manie facilement. Contrairement à

d'autres logiciels de rédaction tel que LibreOffice, OpenOffice, etc., \LaTeX n'est pas de type WYSIWYG (What You See Is What You Get). Il faut donc expliciter la mise en page du document, d'où sa catégorie de langage.

\LaTeX possède des implémentations libres, dont notamment TeX Live.

3.7.2 Beamer

Beamer est un paquet de \LaTeX spécialisé pour la création de présentations sous forme de diapositives. Plusieurs thèmes existent pour la mise en forme et, comme \LaTeX , Beamer n'est pas de type WYSIWYG.

3.8 JSON Spirit

JSON Spirit (ref. [Wilkinson, 2014]) est une bibliothèque C++ qui permet de manipuler des fichiers JSON avec du C++.

Il utilise des structures de données C++ comme les `std::vector` ou les `std::map`, pour stocker les objets JSON.

C'est un outil nouveau, mais assez puissant. Il n'est pas difficile à prendre en main : une classe `Value` existe, et représente n'importe quels types de données (tableaux, objets, etc.) et sert de base à la construction d'objet comme les `Array` de JSON Spirit (un `vector` de `Value`), ou les `Object` (un `vector` de `std::pair` C++).

Grâce à ces emboîtements, on peut représenter un fichier JSON avec exactitude.

Chapitre 4

Modèle

4.1 Les classes d'agents

Sprints 1 et 2

Lors du premier sprint, nous avons commencé par mettre en place la base du programme. Nous avons prévu de réaliser l'UML présent en figure 4.1.

Pour le modèle, il s'agissait de créer les classes :

- **Turtle**, qui représente une tortue ;
- **Point**, pour stocker les coordonnées d'une tortue ;
- **Line**, qui est composé de points, et qui permet aux tortues de laisser une trace (instruction `pen_down`) ;
- **World**.

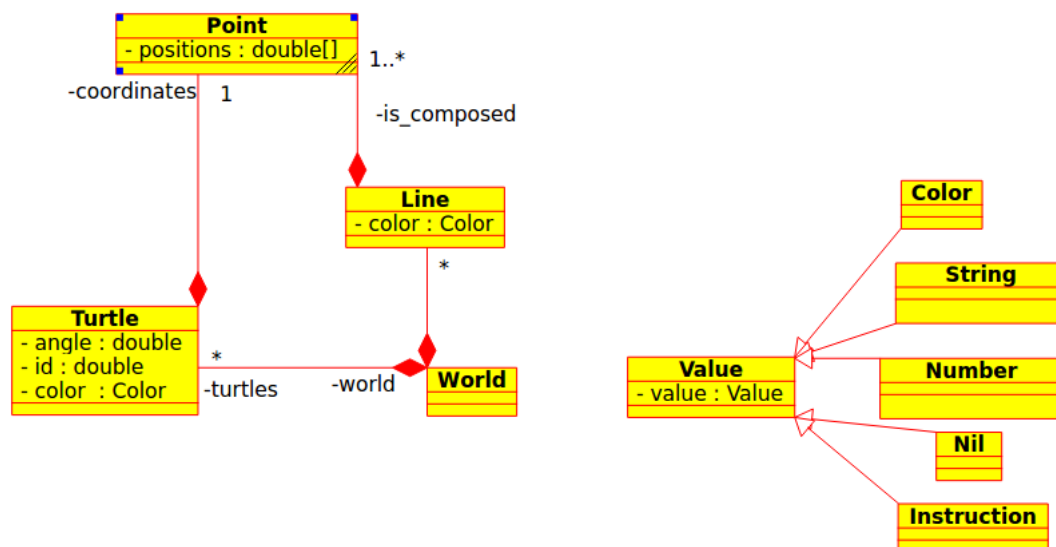


FIGURE 4.1 – UML prévisionnel de la version 0.1

La classe **World** représente le monde, contenant la liste des tortues, et les lignes tracées par celles-ci, et communique avec l'interface pour l'affichage. À la fin du sprint 1, on pouvait voir une tortue avancer et tracer une ligne sur son passage. Nous avons réalisé le schéma 4.2. Il respecte la version prévue, en dehors de la classe **Instruction**, qui n'était finalement pas nécessaire, les instructions étant des méthodes de **Turtle**.

Lors du second sprint, nous avons ajouté une classe **Agent**, super-classe de **World**, **Turtle** et **Zone** ; en effet, elles sont toutes les trois des agents, et ont donc des comportements similaires

(cf. Figure 4.3).

Ces classes contiennent chacune un parent, une liste d'enfants, et des propriétés. Les propriétés sont des variables définies par l'utilisateur lors de la définition du code de l'agent, donc surtout utile pour les tortues.

De plus, le monde a une taille et deux listes d'espèce de tortues (**Breed**) : les espèces nommées et les anonymes.

Comme le montre le listing 4.1, on peut créer des tortues nommées ou pas. Une tortue anonyme a son corps défini à la création de l'agent (lors du **new agent**) tandis que les tortues nommées on leur corps défini lors de la création du type d'agent (instruction **agent <nom>**).

```
agent listener () {  
    fd 2  
}  
  
new listener ()  
  
new agent {  
    lt 30  
    fd 5  
}
```

Listing 4.1 – Nommage lors de la création d'une tortue

Sprints 3 et 4

Lors du troisième sprint, nous avons mis en place des pointeurs intelligents dans toutes nos classes (cf. Figure 4.4). Le but de ce sprint était la mise en place de la communication entre les agents : les tortues devaient pouvoir communiquer via les zones par modification de leurs propriétés, et elles devaient aussi pouvoir communiquer entre elles grâce à des instructions comme **send** et **recv**.

L'étape suivante était d'ajouter des fonctionnalités telles que la maîtrise du temps et l'exportation du modèle. Ces ajouts ne provoquent pas de changements majeurs du côté du modèle, si ce n'est quelques méthodes dans les classes **World**, **Turtle** et **Zone** pour l'export du modèle. Celui-ci consiste à créer une sauvegarde de l'état du modèle à un instant **t** dans un fichier JSON grâce à la bibliothèque JSON Spirit (cf. Listing I.4). Cela permettra ensuite, par exemple en passant par une transformation en CSV, d'avoir des tableaux avec toutes les données, ce qui offre la possibilité d'avoir des diagrammes de l'évolution du monde.

La maîtrise du temps se fait grâce à un bouton pause, qui arrête les threads s'exécutant, ou par un slider qui permet de ralentir ou de diminuer la vitesse.

4.2 Les types

Sprints 1 et 2

Lors du premier sprint, nous avons mis en place un certains nombre de types (cf. Figure 4.2), comme **Color**, représentant les couleurs, ou encore **Nil**, représentant la valeur nulle. Ils héritent de **Value**, une classe abstraite qui contient une valeur et ses accesseurs.

Lors du second sprint, les types Stibbons sont les mêmes mais leurs définitions se sont un peu complexifiées, en passant par une classe **SimpleValue** pour la mise en place des mutex. Une énumération des types Stibbons existe, elle est utilisée avec la méthode **getType()** pour pouvoir connaître le type d'une valeur. Pour que l'utilisateur puisse écrire des fonctions dans le code, nous avons ajouté une classe **Function**, qui stocke un arbre abstrait, contenant le code de la

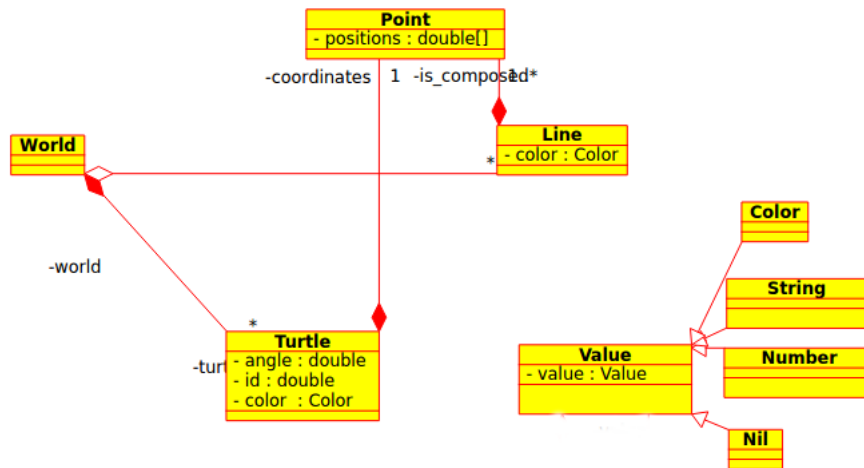


FIGURE 4.2 – UML de la version 0.1 réalisée

fonction déjà analysé. Des mutex ont également été ajoutés dans toutes les classes pour assurer que les objets soient thread-safe (cf. 4.3).

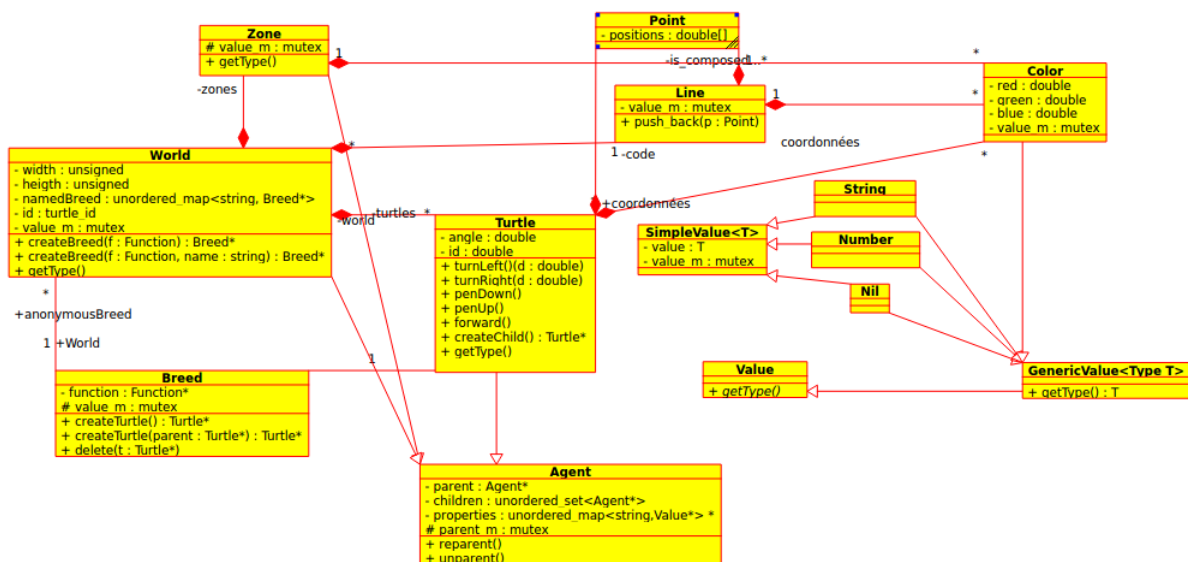


FIGURE 4.3 – UML version 0.2

Sprints 3 et 4

Lors du troisième sprint, nous avons ajouté une sous-classe de `Function`, `userFunction`, qui représente les fonctions créées par l'utilisateur (cf. Figure 4.4). Nous avons également créé des fonctions standard comme `ask_zone`, qui permettent de donner des ordres aux zones.

L'ajout de `Table` représentant l'unique type de conteneurs, les tableaux, fait aussi parti de ce sprint. Nous les écrivons à la façon de PHP (cf. Listing 4.2).

```
a = 12

t = {18, red, "bla"}
v = { "bla" : "blou", "blue" : blue, a : 29 }
```

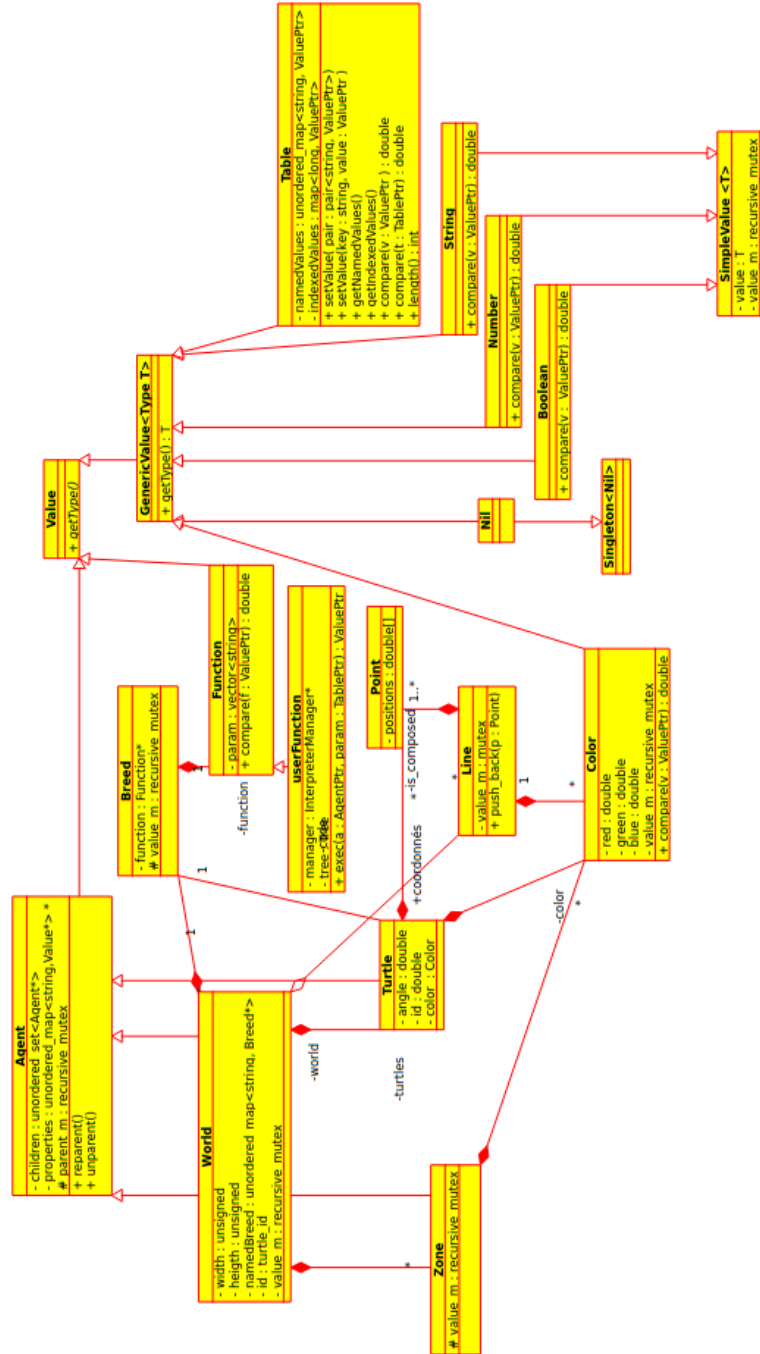



FIGURE 4.5 – UML version 0.4

Chapitre 5

Interprète

L'application Stibbons fournit un interprète (ou interpréteur) pour le langage du même nom. Cette interprétation du code se déroule en deux phases : une de compilation lors du chargement du code, et une d'interprétation de l'arbre abstrait (généré durant la première phase) lors de l'exécution.

La première phase peut elle-même être découpée en deux parties :

- l'analyseur lexical qui « lit le flot de caractères qui constituent le programme source et les regroupe en séquences de caractères significatives appelées *lexèmes*. » [Aho et al., 2007] ;
- l'analyseur syntaxique qui, à partir des lexèmes, génère un arbre abstrait qui pourra par la suite être analysé pour être interprété.

L'interprétation quant à elle se déroule lors de l'analyse sémantique de l'arbre abstrait, qui exécute les opérations contenues dans les nœuds.

5.1 Analyseur lexical

L'analyse lexicale vise à produire un flot de jetons qui pourront être analysés par l'analyseur syntaxique. Ces jetons sont des paires composées d'un type de jeton et de la valeur du lexème (par exemple, l'analyse du lexème 12 va générer le jeton `<NUMBER,12>` dans notre cas). Certains lexèmes peuvent générer des jetons qui n'ont pas de valeur (par exemple, le lexème `(` va entraîner la génération du jeton `<(>`).

Nous avons fait le choix d'utiliser l'outil Flex pour notre projet (cf. 3.5.1).

5.1.1 Jetons

Notre analyseur lexical a dans un premier temps généré un nombre limité de jetons. Ainsi, en version 0.1, nous générions seulement 26 jetons différents (dont 5 jetons de littéraux), contre 40 jetons en version 1.0 (dont 7 jetons de littéraux). Ces jetons sont définis dans le code source Bison (cf. Listing I.2) et leurs valeurs ont un type C++ défini par la structure listing 5.1. Dans cette structure, l'attribut `tok` correspond au type du jeton (défini par une énumération plus tard) tandis que l'attribut `v` correspond à une valeur. En effet, notre langage ayant un typage dynamique, toutes les valeurs ont un type statique de type `Value`. Ainsi, grâce au polymorphisme, le jeton aura une valeur qui pourra être de type dynamique `Number`, `String`, `Boolean`, etc. tout en gardant un type statique de `Value`. C'est lors de l'analyse sémantique (cf. 5.3) que le type réel du jeton est analysé.

```
struct {  
    stibbons::ValuePtr v;  
    stibbons::TreePtr tr;  
    int tok;
```



```
| };
```

Listing 5.1 – Type des valeurs des jetons

5.1.2 Fonctionnement

Par défaut, l’analyseur lexical généré par Flex utilise des variables globales et n’est pas réentrant. Dans l’optique d’obtenir un analyseur réentrant, nous avons utilisé l’option `%option c++` et établi le modèle 5.1.

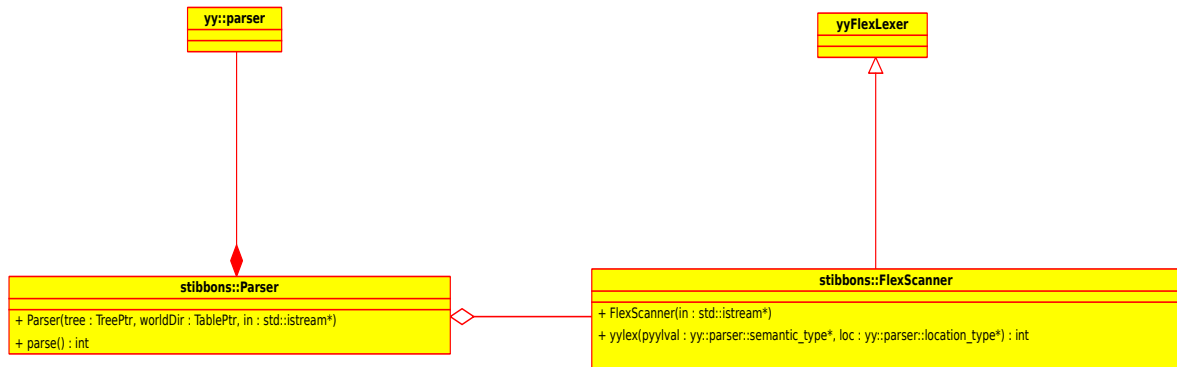


FIGURE 5.1 – UML des analyseurs lexical et syntaxique

Chaque appel à la méthode `yylex()` de `FlexScanner` génère un nouveau jeton à partir du flux d’entrée, passé en paramètre lors de la construction de cet objet. Cette méthode analyse le flux à partir des règles définies dans le fichier `lexer.1+` (cf. Listing I.1); ainsi, les instructions définies pour chaque règle sont effectuées quand une chaîne de caractères correspondante à la règle est détectée. Dans l’extrait 5.2, lorsque l’analyseur détecte le caractère `#` suivi d’une séquence de 3 ou 6 nombres hexadécimaux, un appel au constructeur de `Color(string)` est effectué. Ce dernier crée une couleur à partir d’une chaîne de caractère respectant les codes de couleur html.

```
#([a-f0-9]{6}|[a-f0-9]{3}) {
    pyylval->v=make_shared<stibbons::Color>(
        yytext);
    return yy::parser::token::COLOR;
}
```

Listing 5.2 – Exemple de séquence d’instructions lors de la détection d’une couleur

De plus, un appel à la méthode `step()` du paramètre `loc` (cf. Figure 5.1) est effectué au début de chaque appel à `yylex()`, et permet de faire avancer la position actuelle de la longueur du lexème détecté, et un appel à sa méthode `lines()` est effectué lors de la détection d’un retour à la ligne, afin de faire avancer de `n` lignes la position actuelle (avec `n` le nombre de retour à la ligne détecté). Les différentes règles Flex peuvent être consultées au listing I.1, ou de façon plus lisible dans l’annexe A.1.1.

5.2 Analyseur syntaxique

L’analyse syntaxique permet de vérifier que la structure d’un programme est bien en accord avec les règles de grammaire du langage. Par exemple, la grammaire de notre langage comporte une règle, qui indique qu’un appel de fonction sans paramètre, est constituée du jeton `<ID>` suivi

des jetons $\langle \rangle$ et $\langle \rangle$. Le but de notre analyse ici est double : vérifier que le programme est bien un programme de notre langage valide, mais également générer un arbre abstrait qui pourra facilement être analysé par notre analyseur sémantique.

Nous avons fait le choix d'utiliser GNU Bison (cf. 3.5.2) dans notre projet.

5.2.1 Arbre abstrait

Un arbre abstrait est une structure d'arbre dont chaque nœud feuille représente les opérandes des opérations contenues sur les autres nœuds. Cet arbre peut être considéré comme une forme de code intermédiaire, et peut être interprété très facilement en parcourant, pour chaque nœud, les sous-arbres et en appliquant l'opération du nœud courant.

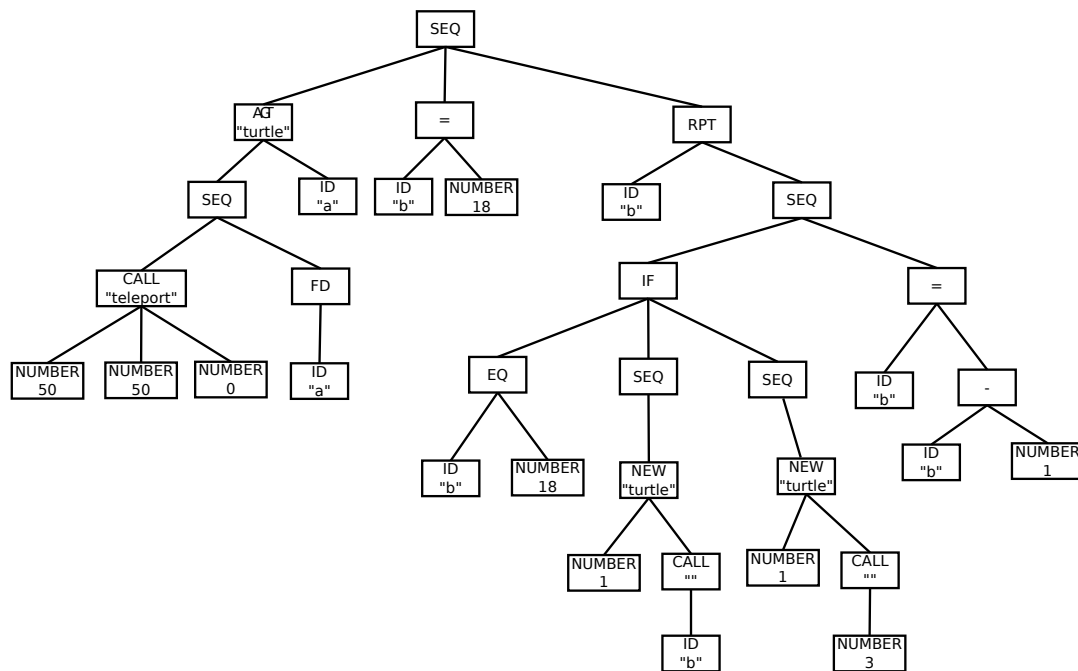


FIGURE 5.2 – Arbre abstrait généré lors de l'analyse du code 5.3

```

agent turtle (a) {
  teleport(50,50,0)
  fd a
}

b = 18
repeat b {
  if(b == 18) {
    new turtle(b)
  }
  else {
    new turtle(3)
  }
  b = b - 1
}

```

Listing 5.3 – Exemple de code Stibbons

L'avantage de générer un tel arbre est qu'il est bien plus rapide d'effectuer un parcours d'arbre à chaque interprétation plutôt que de refaire une analyse syntaxique à chaque fois.

5.2.2 Fonctionnement

L'écriture du fichier Bison est assez semblable au fichier Flex. En effet, on peut associer à chaque règle une liste d'instructions qui vont être exécutées lors de la détection de la règle. Ces instructions peuvent en outre être des affectations de valeur sémantique à une expression. Ainsi, dans l'extrait de code 5.4, la valeur sémantique d'une sélection est un arbre dont les enfants sont dans l'ordre : l'expression de test, la séquence d'instructions à exécuter si la condition est vraie et, optionnellement, la séquence d'instructions à exécuter si la condition est fausse.

```
selection : IF expr statement
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::IF, nullptr);
    t1->addChild($2);
    t1->addChild($3);
    t1->setPosition({@1.begin.line, @1.begin.column});
    $$ = t1;
}
| IF expr statement ELSE statement
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::IF, nullptr);
    t1->addChild($2);
    t1->addChild($3);
    t1->addChild($5);
    t1->setPosition({@1.begin.line, @1.begin.column});
    $$ = t1;
};
```

Listing 5.4 – Cas du IF en Bison

Bison génère à partir de ces règles un analyseur syntaxique LALR (*look-ahead left-to-right rightmost derivation*). L'analyse du code et la génération de l'arbre sont lancées par un appel à la méthode `stibbons::Parser::parse()` qui va elle-même faire appel à la méthode `yy::parser::parse()`. L'ensemble du code va être analysé, cette méthode se chargeant de faire appel à la méthode `stibbons::FlexScanner::yylex()`, générant et analysant les jetons en une seule passe.

5.3 Analyseur sémantique

L'objectif de l'analyseur sémantique est l'interprétation pure et simple de l'arbre abstrait précédemment généré. En effet, le but ici est l'exécution du code écrit en Stibbons, et l'interprète prend ainsi en paramètre un arbre ainsi qu'un agent sur lequel exécuter cet arbre. Il effectue un parcours en profondeur de l'arbre, se rappelant récursivement sur chaque nœud fils de l'arbre.

5.3.1 Fonctionnement

Lors de l'analyse d'un jeton, qui correspond à un nœud de l'arbre syntaxique, on analyse d'abord ses enfants (s'il y en a) puis ensuite le contenu du nœud courant. Par exemple, pour le test d'égalité `1 == 2`, le nœud `<EQ>` aura pour enfants les sous-arbres `<NUMBER,1>` et `<NUMBER,2>`. Ces deux derniers seront interprétés en premier et renverront leurs valeurs sémantiques (en

l'occurrence, des `stibbons::Number` ayant respectivement pour valeur 1 et 2), puis le test `==` sera appliqué sur ces résultats.

Nous avons dû faire un certain nombre de choix sémantiques lors de l'élaboration de notre langage, et en voici quelques uns particuliers :

affectation Lors d'une affectation, on modifie en priorité les paramètres de l'agent ou de la fonction courante, si l'identifiant existe dans la table des paramètres, sinon, on modifie la propriété correspondante de l'agent.

identifiant Lorsque l'on accède à la valeur d'un identifiant, on recherche tout d'abord cet identifiant dans les paramètres de l'agent courant. S'il n'existe pas, on le recherche dans les propriétés de l'agent courant. Ainsi, la portée des « variables » est exclusivement locale à l'agent. On peut explicitement accéder à une propriété d'un autre agent, via l'opérateur « `.` ».

fonction Les fonctions sont des valeurs. Ainsi, tout comme les valeurs, leurs créations ont un comportement similaire à une affectation et leurs accès également (espace de noms des fonctions et des variables communs).

agent La création d'un nouveau type d'agent, où qu'elle soit faite, est liée au monde et est, par conséquent, accessible depuis tous les autres agents.

5.3.2 Fonctionnalités

Sprints 1 et 2

Les deux premiers sprints ont été assez conséquents au niveau du nombre de fonctionnalités ajoutées. En effet lors du sprint 1, on pouvait déjà effectuer les opérations élémentaires sur une tortue, telles que avancer, tourner, écrire, etc. De plus, les opérations arithmétiques ainsi que le support des nombres ont été faits. En effet, il était nécessaire de gérer les nombres de manière à pouvoir indiquer à la tortue la distance de laquelle elle devait avancer.

Lors du deuxième sprint sont apparus les conditionnelles, ainsi que les boucles, les comparaisons d'ordre, les booléens et de nouveaux types (couleurs, chaînes de caractères, etc.). Sont également apparues la création de nouveaux agents dans le code et les fonctions sans paramètres. Ce sprint fut déjà une version bien avancée de notre programme.

Sprints 3 à 5

Les sprints suivants furent plus légers. Non pas parce qu'il y avait moins de travail, car la gestion de l'interpréteur fut remaniée à ce moment là, mais car il y avait moins de fonctionnalités à ajouter. En effet, à partir du sprint 3, les fonctionnalités suivantes ont été rajoutées :

- l'accès à la parenté et aux zones ;
- l'ajout des tables et de boucles dédiées (`for`) ;
- la gestion de la vitesse et de la pause ;
- l'ajout des communications avec le `send` et le `recv`.

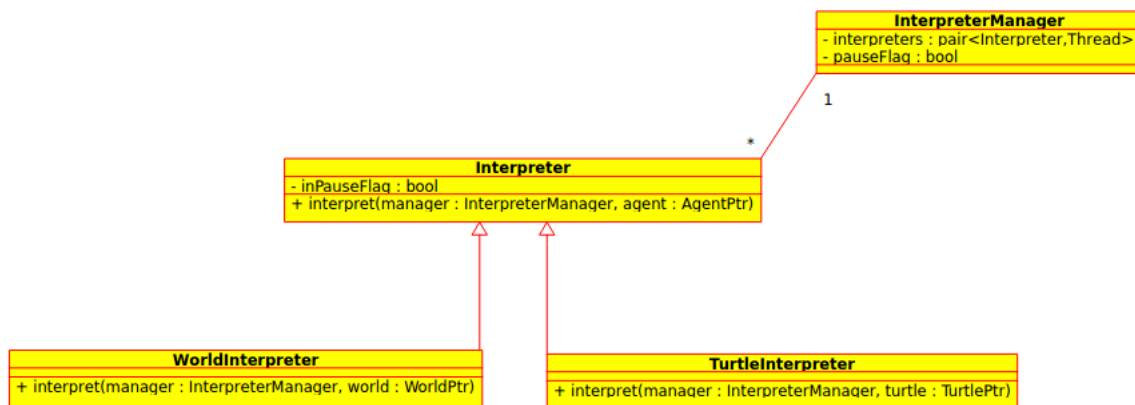
5.3.3 Organisation

L'analyseur sémantique a d'abord été implémenté par l'unique classe `Interpreter`. Cette dernière implémentait tous types d'actions à effectuer pour n'importe quel type d'agent. Cependant, lors de l'arrivée de la fonctionnalité d'ajout d'un nouvel agent (`new agent`), nous nous sommes aperçus qu'il était plus intéressant d'avoir un interpréteur par type d'agent, ou plus précisément un interpréteur pour le monde, un pour les tortues et un pour les actions communes aux deux types (cf. UML 5.3) ; nous avons alors créé les classes `WorldInterpreter` et `TurtleInterpreter`.

Ainsi, par exemple, si on demande à une tortue d'avancer en Stibbons (`fd 10`), alors c'est le `TurtleInterpreter` qui gèrera cette action. Par contre, si on effectue une affectation (`a = 10`) alors l'`Interpreter` affectera la valeur 10 à la propriété `a` de l'agent courant.

De manière parallèle, la création du monde s'effectuait dans l'`Interpreter`, puis dans le `WorldInterpreter`, il fut alors nécessaire de créer une classe qui gèrerait à la fois la création du monde et tout ce qui concernait l'application (la pause, le temps, les interpréteurs eux-mêmes). Nous avons alors décidé de créer la classe `InterpreterManager`. Cette classe connaît tous les interpréteurs, et permet de les prévenir d'une éventuelle pause du programme, de stocker les threads correspondants à chaque interpréteur, de créer un monde avec les directives choisies ; c'est une sorte de gestionnaire d'interpréteurs.

FIGURE 5.3 – UML de l'analyseur sémantique



Chapitre 6

Applications

6.1 Application graphique

6.1.1 Version 0.1

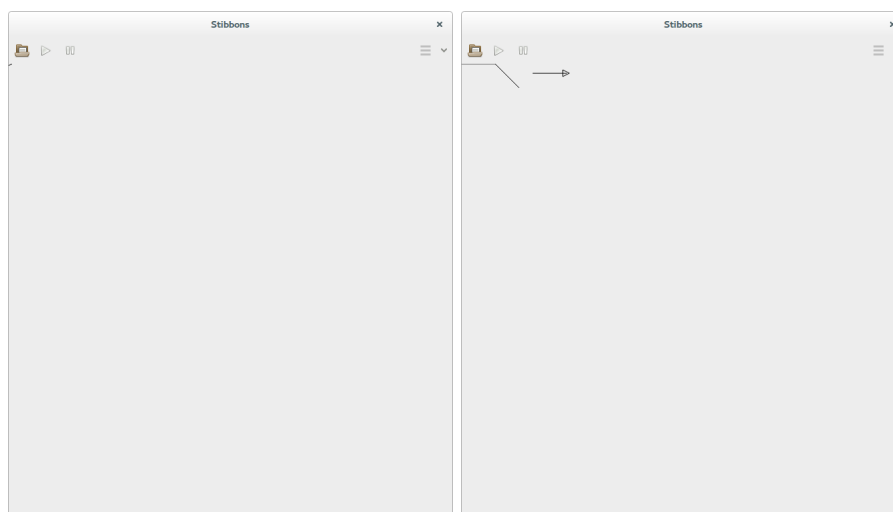


FIGURE 6.1 – Capture d’écran de la version 0.1

L’interface de la version 0.1 (cf. Figure 6.1) de Stibbons est composée d’une fenêtre séparée en une barre d’outil et une vue du monde. La barre d’outil contient un bouton permettant d’ouvrir un fichier contenant un programme Stibbons (Ctrl+O), un bouton permettant d’exécuter le programme ouvert, un bouton de pause non fonctionnel, un bouton ouvrant un menu proposant la boîte de dialogues « À propos » et de quitter l’application (Ctrl+Q).

Si la boîte de dialogue permettant d’ouvrir un programme est fermée sans avoir choisi de fichier, un message d’erreur l’indiquant est alors affiché.

Une tortue existe par défaut dans le monde, elle est située dans le coin supérieur gauche de la vue et est orientée vers la droite. La tortue est alors dessinée comme un triangle noir creux et elle peut dessiner des lignes noires.

L’analyseur syntaxique n’étant à ce moment pas réentrant, ouvrir plusieurs programmes les uns après les autres implique alors de redémarrer l’application.

6.1.2 Version 0.2

La version 0.2 (cf. Figure 6.2) apporte peu de modifications à l’application Stibbons elle-même, le changement le plus visible étant le placement temporaire de l’origine au centre de la vue, afin d’avoir une meilleure vue du programme s’exécutant.

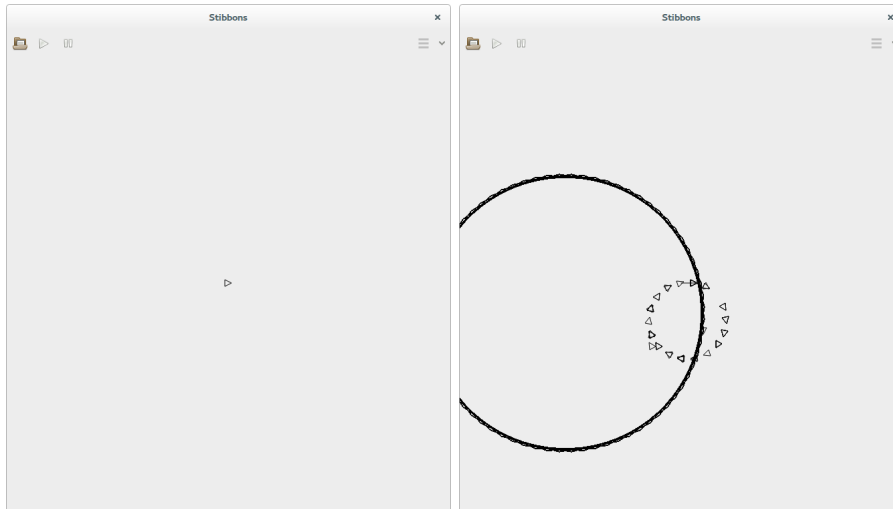


FIGURE 6.2 – Capture d’écran de la version 0.2

6.1.3 Version 0.3

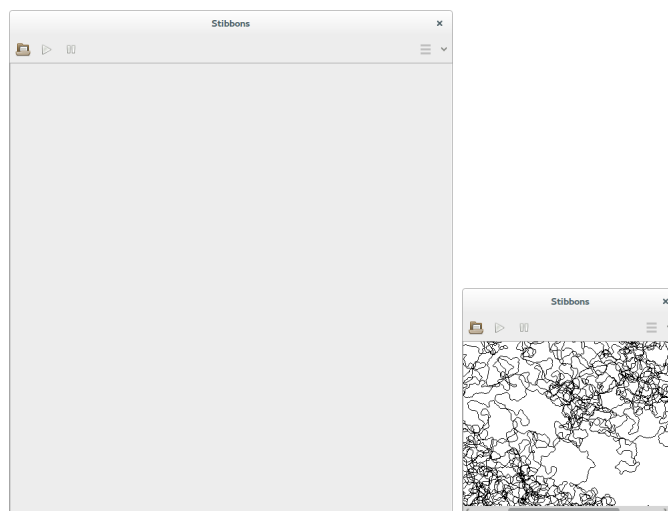


FIGURE 6.3 – Capture d’écran de la version 0.3

À partir de la version 0.3 (cf. Figure 6.3), les couleurs des tortues peuvent être modifiées et, afin de les rendre plus visibles, elles sont désormais dessinées comme des triangles pleins. La ligne dessinée par une tortue prend la couleur de cette dernière au moment de l’abaissement du stylo.

Le monde est désormais borné, c’est à dire que les tortues ne peuvent plus sortir du monde visible pour l’utilisateur, et est centré dans la vue, les zones le constituant sont désormais affichées, et leurs couleurs peuvent être modifiées. Si la vue est plus petite que le monde, des ascenseurs seront affichés.

Redessiner le monde prenait de plus en plus de temps au fur et à mesure que le nombre de lignes le constituant augmentait. Afin d’améliorer drastiquement ces performances de rendu, les lignes seront désormais dessinées sur un tampon et seules les nouvelles lignes auront donc besoin d’être dessinées sur ce dernier, avant qu’il soit reporté sur la vue.

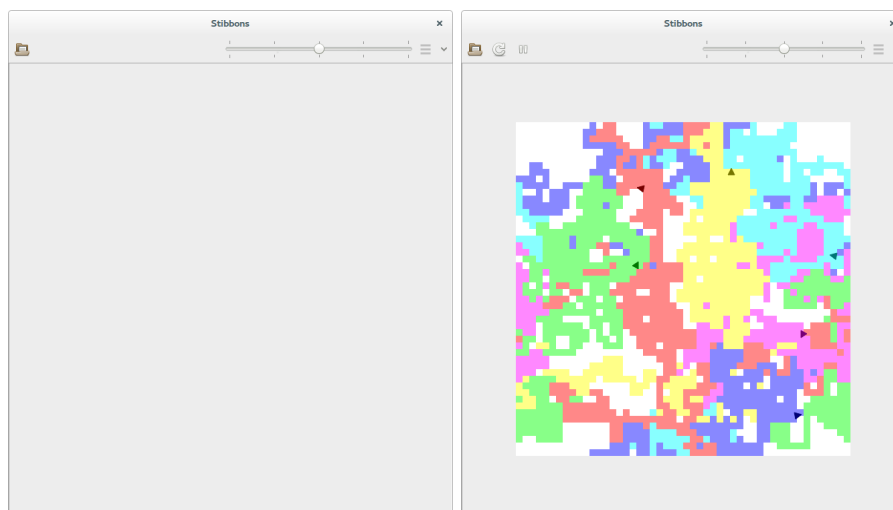


FIGURE 6.4 – Capture d’écran de la version 0.4

6.1.4 Version 0.4

À partir de la version 0.4 (cf. Figure 6.4), il est possible de modifier la vitesse d’exécution du programme, de le mettre en pause, de reprendre son exécution et de le redémarrer depuis le début (F5). De plus, l’analyseur syntaxique étant désormais réentrant, il est possible d’ouvrir plusieurs programmes les uns à la suite des autres.

Les boutons de démarrage et de mise en pause sont désormais confondus, et sont cachés, de même que le bouton de redémarrage, tant qu’aucun programme n’a été ouvert.

Une entrée a été ajoutée au menu, permettant d’exporter le modèle en cours d’exécution.

Si une tortue change de couleur alors qu’elle trace une ligne, le stylo avec lequel la tortue trace ses lignes changera de couleur en même temps.

Il est désormais possible de faire reboucler le monde sur lui même, cette option étant paramétrable pour chaque axe. Cela permet aux tortues, lorsqu’elles atteignent un bord, d’apparaître du côté opposé, comme si le monde était circulaire.

Le monde ne comporte plus de tortue par défaut et c’est désormais celui-ci qui exécute le corps principal du programme.

6.1.5 Version 1.0

La version 1.0 (cf. Figure 6.5) marque l’apparition d’un éditeur de programme Stibbons dans l’application elle-même. L’éditeur propose également une coloration syntaxique. L’ajout de cet éditeur implique que les boutons de démarrage, de pause et de redémarrage ne soient plus cachés par défaut car il est désormais possible d’exécuter un programme écrit sans en avoir chargé un avant.

De nombreux raccourcis clavier sont ajoutés, afin d’enregistrer le programme dans son fichier (Ctrl+S) ou dans un nouveau fichier (Ctrl+Shift+S), d’exporter le modèle (Ctrl+E) et de démarrer ou mettre en pause l’exécution du programme (Ctrl+Espace).

Si la version 0.4 a ajouté une option permettant au monde de reboucler selon chaque axe, la version 1.0 permet d’avoir des bords solides, faisant rebondir toute tortue les croisant.

6.2 Application en ligne de commande

Une application en ligne de commande a été ajoutée à la version 0.4. Elle permet d’exécuter des programmes Stibbons sans serveur graphique et à pleine vitesse.

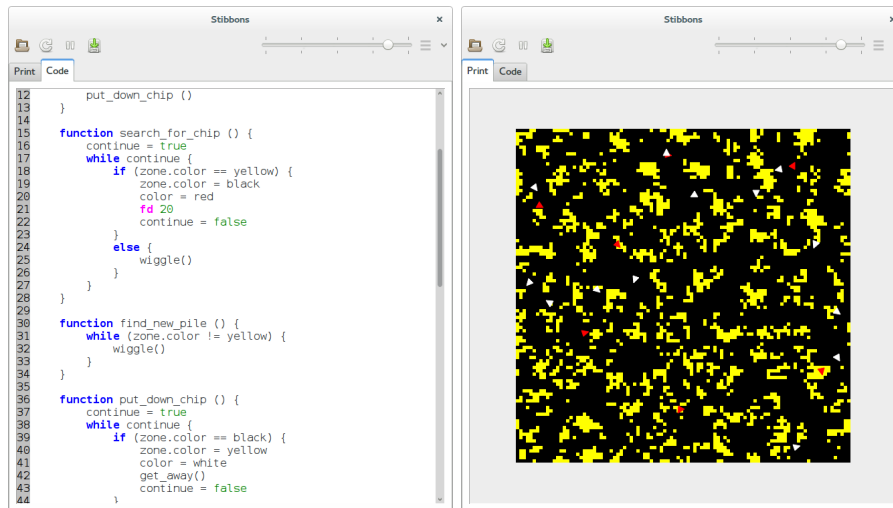


FIGURE 6.5 – Capture d’écran de la version 1.0

6.2.1 Utilisation

Utilisation : `stibbons-cli [options] fichier`

Options :

- h, --help** Affiche l’aide.
- v, --version** Affiche l’information de version.
- e, --export <secondes>** Exporte le modèle toutes les <secondes> secondes.
- p, --prefix <prefixe>** Préfixe les fichiers exportés avec <prefixe>.
- png** Génère une image PNG pour chaque export.
- no-json** N’exporte pas le modèle dans un fichier JSON.

Arguments :

fichier Le fichier de programme Stibbons à exécuter.

6.2.2 Fonctionnement

Le programme analyse la ligne de commande à l’aide de la classe `QCommandLineParser`, puis crée et exécute un objet représentant l’application selon les paramètres passés par l’utilisateur.

Lors de l’exécution de l’application, il est possible d’exporter le modèle à intervalle régulier grâce à l’option **--export**. Par défaut, le modèle est exporté dans un fichier JSON, mais il est également possible d’exporter un rendu du monde en PNG, ou encore de ne rien exporter.

Afin de pouvoir dessiner le monde dans un fichier image, la classe `WorldView`, widget de l’application graphique jusque là responsable de dessiner le monde sur lui même, a vu ses fonctionnalités de dessin d’un monde migrer vers la nouvelle classe `WorldPainter`, capable de dessiner un monde sur une surface. Cette nouvelle classe est désormais utilisée dans l’application graphique par `WorldView` pour se dessiner, et par l’application en ligne de commande pour dessiner sur une `QImage`.

Chapitre 7

Conclusion

Durant quatre mois, nous avons réalisé ce projet que nous avions imaginé. Ce projet a été positif à bien des égards. En effet, il nous a tout d'abord permis de pratiquer une méthode agile, ce qui était une bonne expérience et, avec le recul, un bon choix, car nous avons un projet fonctionnel toutes les deux semaines. Nous avons pu en outre grâce à cette méthode ajouter facilement de nouvelles fonctionnalités en cours de route, avec notamment l'application en ligne de commande, complémentaire à notre programme principal.

Ce fut également une expérience profitable car nous avons pu voir l'importance de la communication sur ce type de développement. En effet, les cycles étant très courts, il était important de communiquer rapidement et clairement pour éviter de prendre du retard. Si nous avons parfois dépassé légèrement le délai (quelques heures de retard sur le premier sprint), nous avons tout de même su gérer notre temps, ce qui nous a permis d'être dans les temps sur l'ensemble du projet.

Nous avons également pu grâce à ce projet travailler avec des outils nouveaux (Qt, JsonSpirit, `std::thread`) et nous améliorer sur d'autres outils (Flex, Bison, Pointeurs intelligents).

Bien que la 1.0 soit une version fonctionnelle, il y a cependant des choses à améliorer. La première, et sans doute la plus importante, est le fait que notre parcours d'arbre se fait de façon récursive. Par conséquent, les appels récurifs en Stibbons vont impacter directement la pile d'exécution de l'interprète, et nous limite beaucoup. De plus, les tortues évoluant chacune dans un thread séparé, on est relativement vite limité par le nombre de tortues pouvant évoluer dans notre programme (changement de contexte lourd). Il y a également bon nombre de fonctionnalités que nous pourrions ajouter, comme les entrées dans l'interface, un peu à la manière de NetLogo, qui permettraient de modifier des propriétés de façon interactive avec l'exécution, ou encore l'affichage de la sortie standard dans un cadre du programme, voire dans des petites bulles au dessus des tortues.

Nous retiendrons néanmoins de notre projet un bilan positif. En effet, nous avons créé un langage, le Stibbons, ainsi que son interprète et une application graphique permettant de suivre l'évolution du modèle.

Table des figures

3.1	Système de suivi de bugs de Gitlab	18
3.2	Vue des branches dans Gitg	19
4.1	UML prévisionnel de la version 0.1	25
4.2	UML de la version 0.1 réalisée	27
4.3	UML version 0.2	27
4.4	UML version 0.3	28
4.5	UML version 0.4	29
5.1	UML des analyseurs lexical et syntaxique	31
5.2	Arbre abstrait généré lors de l'analyse du code 5.3	32
5.3	UML de l'analyseur sémantique	35
6.1	Capture d'écran de la version 0.1	36
6.2	Capture d'écran de la version 0.2	37
6.3	Capture d'écran de la version 0.3	37
6.4	Capture d'écran de la version 0.4	38
6.5	Capture d'écran de la version 1.0	39

Liste des tableaux

3.1	Le backlog au début du sprint 2	16
3.2	Le backlog à la fin du sprint 5	17

Listings

2.1	Procédure en Logo	12
2.2	Boucle en Logo	12
2.3	Conditionnelles en Logo	12
3.1	Partie définition d'un fichier Flex	20
3.2	Options de Flex	20
3.3	Partie règles de Flex	20
3.4	Partie fonctions de Flex	20
3.5	Définition C++ en bison	20
3.6	Règles de grammaire en bison	21
3.7	Exemple du type des valeurs des jetons avant la version 0.3	21
3.8	Exemple du type des valeurs des jetons en version 1.0	22
4.1	Nommage lors de la création d'une tortue	26
4.2	Syntaxe des tables en Stibbons	27
5.1	Type des valeurs des jetons	30
5.2	Exemple de séquence d'instructions lors de la détection d'une couleur	31
5.3	Exemple de code Stibbons	32
5.4	Cas du IF en Bison	33
	src/interpreter/lexer.l+	183
	src/interpreter/parser.y+	188
	src/tests/test-agent.cpp	203
	doc/report/listings/sauvegarde.json	205

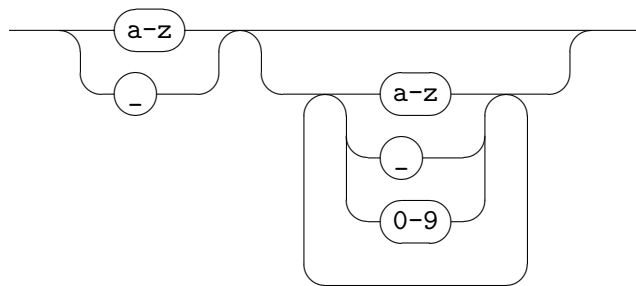
Annexe A

Documentation

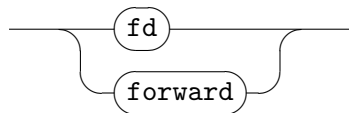
A.1 Syntaxe

A.1.1 Flex

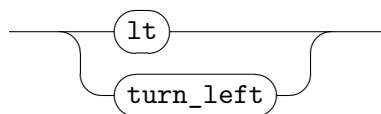
ID



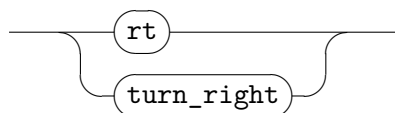
FD



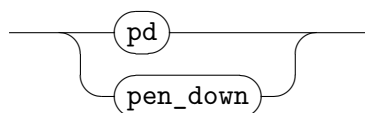
LT



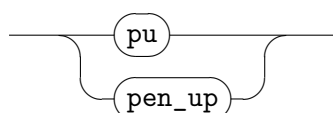
RT



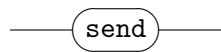
PD



PU



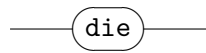
SEND



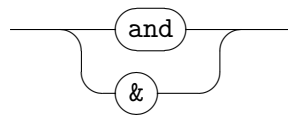
RECV



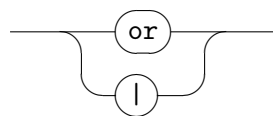
DIE



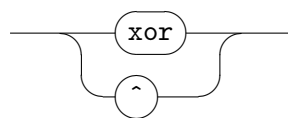
AND



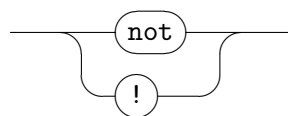
OR



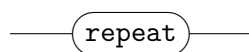
XOR



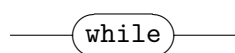
NOT



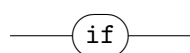
RPT



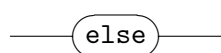
WHL



IF



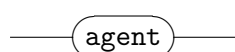
ELSE



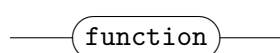
NEW



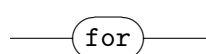
AGT



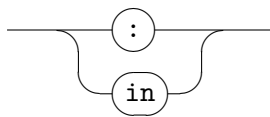
FCT



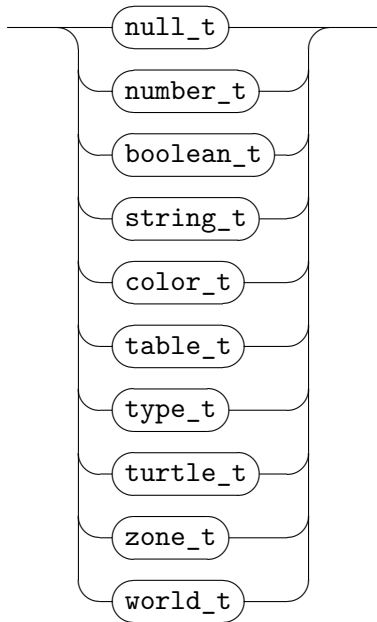
FOR



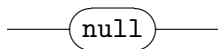
IN



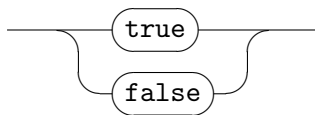
TYPE



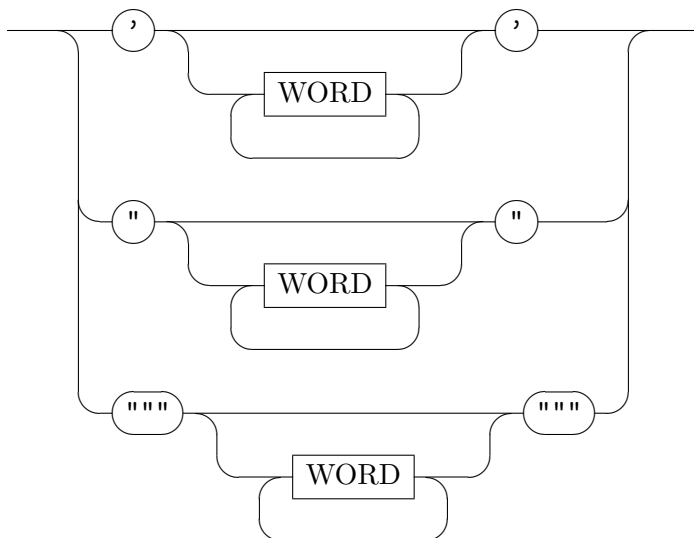
NIL



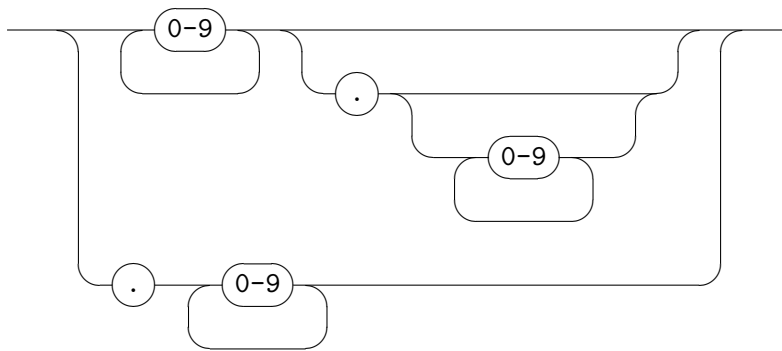
BOOLEAN



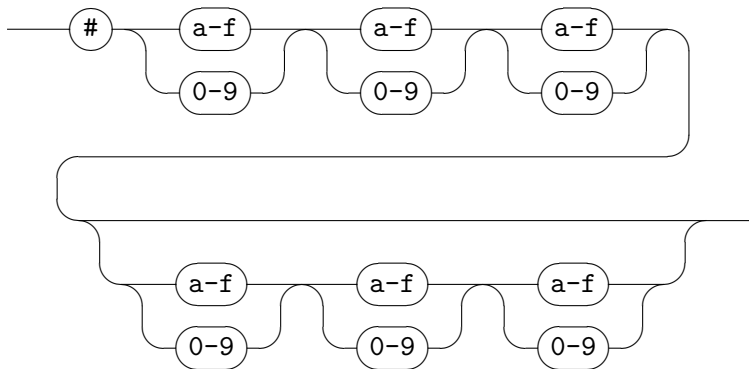
STRING



NUMBER

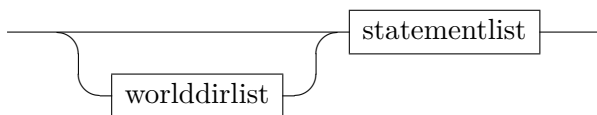


COLOR

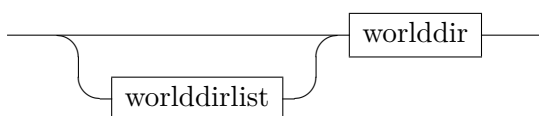


A.1.2 Bison

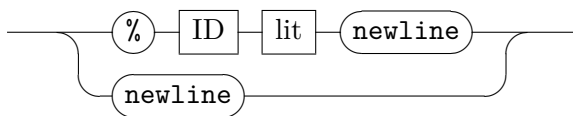
code



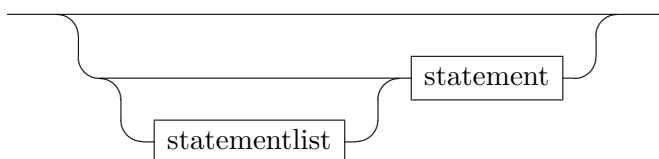
worlddirlist



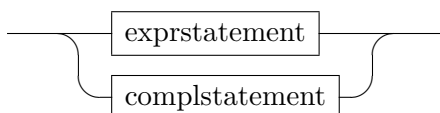
worlddir



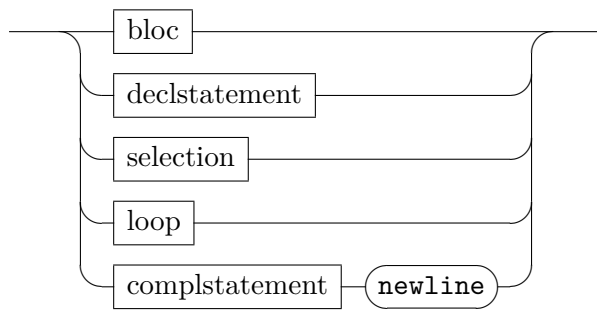
statementlist



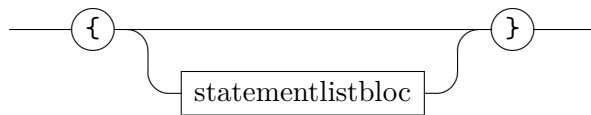
statement



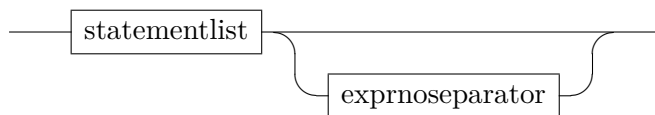
complstatement



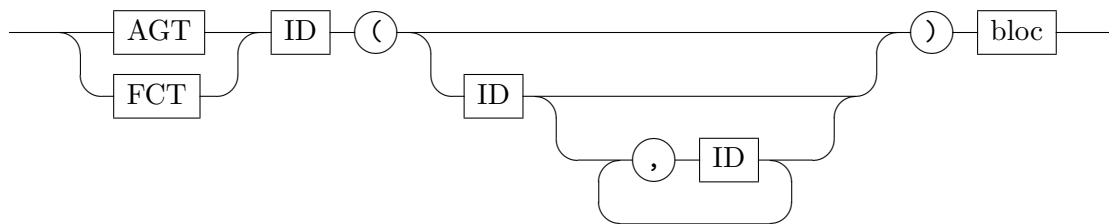
bloc



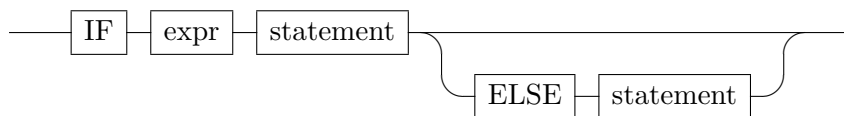
statementlistbloc



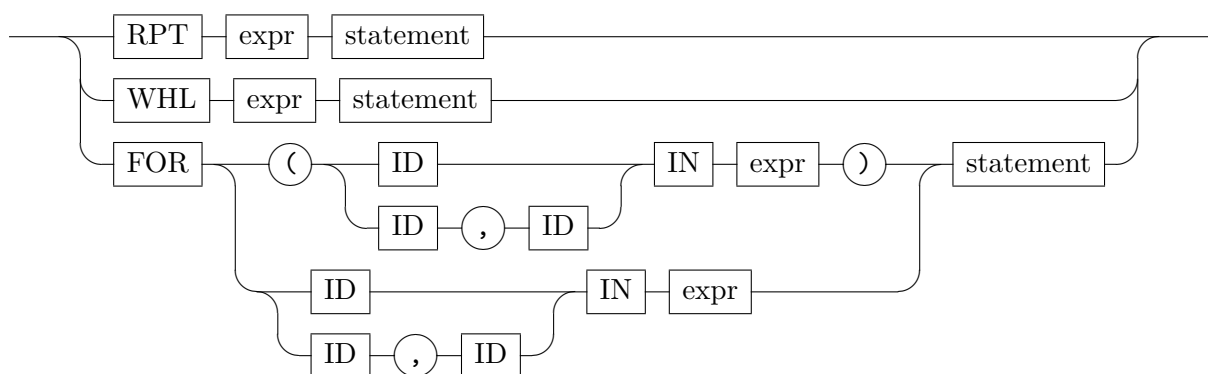
declstatement



selection



loop



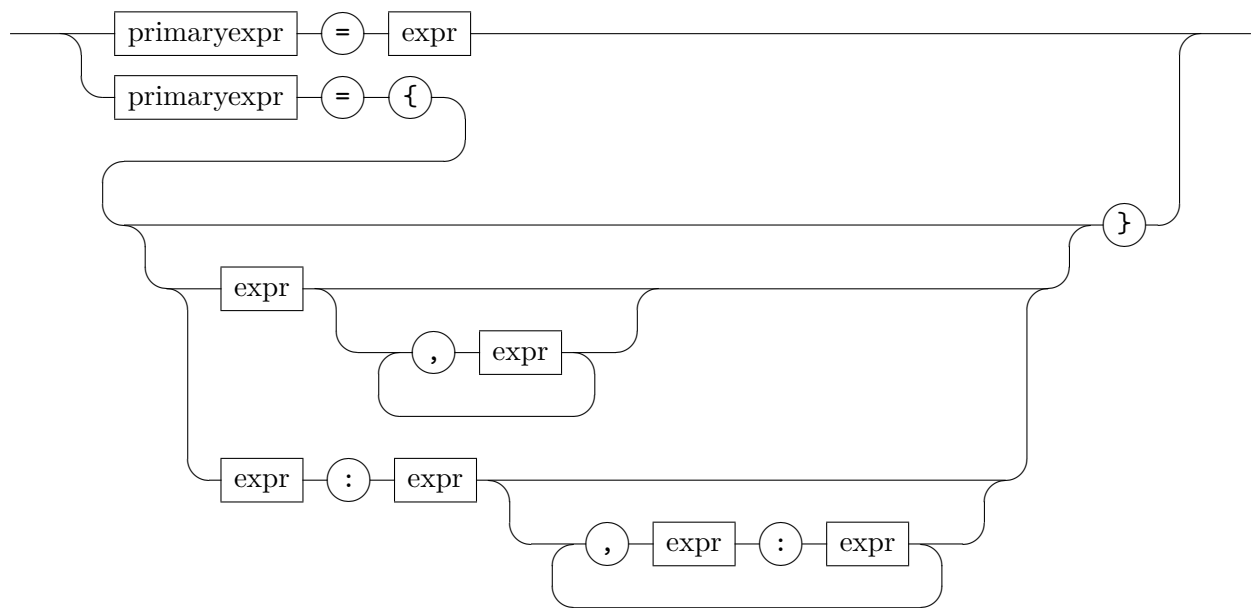
```

graph LR
    block([block]) --- newline1([newline])
    block --- expr_newline([expr newline])
    block --- instrturtle_newline([instrturtle newline])
    block --- creatstatement_newline([creatstatement newline])
  
```

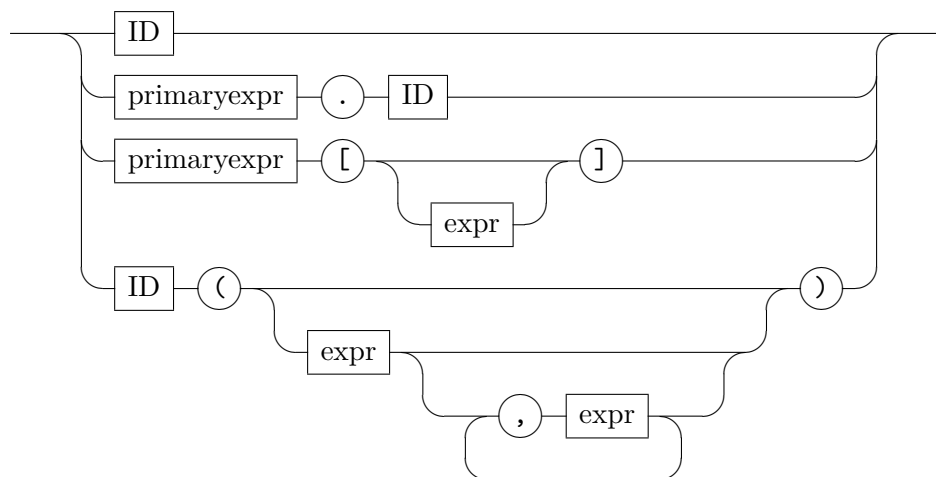
```

graph TD
    assignmentexpression --> LParen["("]
    assignmentexpression --> expr1["expr"]
    assignmentexpression --> RParen[")"]
    expr1 --> expr2["expr"]
    expr1 --> Plus["+"]
    expr1 --> expr3["expr"]
    expr1 --> Minus["-"]
    expr1 --> expr4["expr"]
    expr1 --> Slash["/"]
    expr1 --> expr5["expr"]
    expr1 --> Star["*"]
    expr1 --> expr6["expr"]
    expr1 --> Percent["%"]
    expr1 --> expr7["expr"]
    expr1 --> AND["AND"]
    expr1 --> expr8["expr"]
    expr1 --> OR["OR"]
    expr1 --> expr9["expr"]
    expr1 --> XOR["XOR"]
    expr1 --> expr10["expr"]
    expr1 --> Eq["=="]
    expr1 --> expr11["expr"]
    expr1 --> NotEq["!="]
    expr1 --> expr12["expr"]
    expr1 --> GT[">"]
    expr1 --> expr13["expr"]
    expr1 --> GTE[">="]
    expr1 --> expr14["expr"]
    expr1 --> LT["<"]
    expr1 --> expr15["expr"]
    expr1 --> LTE["<="]
    expr1 --> Minus2["-"]
    expr1 --> expr16["expr"]
    primaryexpr --> NOT["NOT"]
    primaryexpr --> expr17["expr"]
    lit[lit]
  
```

assignmentexpression



primaryexpr



```

graph LR
    subgraph Expr_1 [ ]
        direction TB
        P1[primaryexpr] --- E1((+))
    end
    subgraph Expr_2 [ ]
        direction TB
        N1[NUMBER] --- P2[primaryexpr]
    end
    subgraph Expr_3 [ ]
        direction TB
        E2[expr] --- C1((,))
        E2 --- E3[expr]
    end
    subgraph Expr_4 [ ]
        direction TB
        N2[NEW] --- ID[ID] --- P3[primaryexpr]
    end
    subgraph Expr_5 [ ]
        direction TB
        N3[NEW] --- AGT[AGT] --- B[bloc]
    end
    E1 --- E2
    E1 --- E3
    E1 --- E4[ ]
    E2 --- E5[ ) ]
    E3 --- E5
    E4 --- N1
    E4 --- N2
    E4 --- E6[ ]
    E6 --- E7[ ]
    E6 --- E8[ ]
    E7 --- E9[ ]
    E8 --- E9
    E9 --- E10[ ]
    E10 --- E11[ ]
    E10 --- E12[ ]
    E11 --- E13[ ]
    E12 --- E13
    E13 --- E14[ ]
    E14 --- E15[ ]
    E14 --- E16[ ]
    E15 --- E17[ ]
    E16 --- E17
    E17 --- E18[ ]
    E18 --- E19[ ]
    E18 --- E20[ ]
    E19 --- E21[ ]
    E20 --- E21
    E21 --- E22[ ]
    E22 --- E23[ ]
    E22 --- E24[ ]
    E23 --- E25[ ]
    E24 --- E25
    E25 --- E26[ ]
    E26 --- E27[ ]
    E26 --- E28[ ]
    E27 --- E29[ ]
    E28 --- E29
    E29 --- E30[ ]
    E30 --- E31[ ]
    E30 --- E32[ ]
    E31 --- E33[ ]
    E32 --- E33
    E33 --- E34[ ]
    E34 --- E35[ ]
    E34 --- E36[ ]
    E35 --- E37[ ]
    E36 --- E37
    E37 --- E38[ ]
    E38 --- E39[ ]
    E38 --- E40[ ]
    E39 --- E41[ ]
    E40 --- E41
    E41 --- E42[ ]
    E42 --- E43[ ]
    E42 --- E44[ ]
    E43 --- E45[ ]
    E44 --- E45
    E45 --- E46[ ]
    E46 --- E47[ ]
    E46 --- E48[ ]
    E47 --- E49[ ]
    E48 --- E49
    E49 --- E50[ ]
    E50 --- E51[ ]
    E50 --- E52[ ]
    E51 --- E53[ ]
    E52 --- E53
    E53 --- E54[ ]
    E54 --- E55[ ]
    E54 --- E56[ ]
    E55 --- E57[ ]
    E56 --- E57
    E57 --- E58[ ]
    E58 --- E59[ ]
    E58 --- E60[ ]
    E59 --- E61[ ]
    E60 --- E61
    E61 --- E62[ ]
    E62 --- E63[ ]
    E62 --- E64[ ]
    E63 --- E65[ ]
    E64 --- E65
    E65 --- E66[ ]
    E66 --- E67[ ]
    E66 --- E68[ ]
    E67 --- E69[ ]
    E68 --- E69
    E69 --- E70[ ]
    E70 --- E71[ ]
    E70 --- E72[ ]
    E71 --- E73[ ]
    E72 --- E73
    E73 --- E74[ ]
    E74 --- E75[ ]
    E74 --- E76[ ]
    E75 --- E77[ ]
    E76 --- E77
    E77 --- E78[ ]
    E78 --- E79[ ]
    E78 --- E80[ ]
    E79 --- E81[ ]
    E80 --- E81
    E81 --- E82[ ]
    E82 --- E83[ ]
    E82 --- E84[ ]
    E83 --- E85[ ]
    E84 --- E85
    E85 --- E86[ ]
    E86 --- E87[ ]
    E86 --- E88[ ]
    E87 --- E89[ ]
    E88 --- E89
    E89 --- E90[ ]
    E90 --- E91[ ]
    E90 --- E92[ ]
    E91 --- E93[ ]
    E92 --- E93
    E93 --- E94[ ]
    E94 --- E95[ ]
    E94 --- E96[ ]
    E95 --- E97[ ]
    E96 --- E97
    E97 --- E98[ ]
    E98 --- E99[ ]
    E98 --- E100[ ]
    E99 --- E101[ ]
    E100 --- E101
    E101 --- E102[ ]
    E102 --- E103[ ]
    E102 --- E104[ ]
    E103 --- E105[ ]
    E104 --- E105
    E105 --- E106[ ]
    E106 --- E107[ ]
    E106 --- E108[ ]
    E107 --- E109[ ]
    E108 --- E109
    E109 --- E110[ ]
    E110 --- E111[ ]
    E110 --- E112[ ]
    E111 --- E113[ ]
    E112 --- E113
    E113 --- E114[ ]
    E114 --- E115[ ]
    E114 --- E116[ ]
    E115 --- E117[ ]
    E116 --- E117
    E117 --- E118[ ]
    E118 --- E119[ ]
    E118 --- E120[ ]
    E119 --- E121[ ]
    E120 --- E121
    E121 --- E122[ ]
    E122 --- E123[ ]
    E122 --- E124[ ]
    E123 --- E125[ ]
    E124 --- E125
    E125 --- E126[ ]
    E126 --- E127[ ]
    E126 --- E128[ ]
    E127 --- E129[ ]
    E128 --- E129
    E129 --- E130[ ]
    E130 --- E131[ ]
    E130 --- E132[ ]
    E131 --- E133[ ]
    E132 --- E133
    E133 --- E134[ ]
    E134 --- E135[ ]
    E134 --- E136[ ]
    E135 --- E137[ ]
    E136 --- E137
    E137 --- E138[ ]
    E138 --- E139[ ]
    E138 --- E140[ ]
    E139 --- E141[ ]
    E140 --- E141
    E141 --- E142[ ]
    E142 --- E143[ ]
    E142 --- E144[ ]
    E143 --- E145[ ]
    E144 --- E145
    E145 --- E146[ ]
    E146 --- E147[ ]
    E146 --- E148[ ]
    E147 --- E149[ ]
    E148 --- E149
    E149 --- E150[ ]
    E150 --- E151[ ]
    E150 --- E152[ ]
    E151 --- E153[ ]
    E152 --- E153
    E153 --- E154[ ]
    E154 --- E155[ ]
    E154 --- E156[ ]
    E155 --- E157[ ]
    E156 --- E157
    E157 --- E158[ ]
    E158 --- E159[ ]
    E158 --- E160[ ]
    E159 --- E161[ ]
    E160 --- E161
    E161 --- E162[ ]
    E162 --- E163[ ]
    E162 --- E164[ ]
    E163 --- E165[ ]
    E164 --- E165
    E165 --- E166[ ]
    E166 --- E167[ ]
    E166 --- E168[ ]
    E167 --- E169[ ]
    E168 --- E169
    E169 --- E170[ ]
    E170 --- E171[ ]
    E170 --- E172[ ]
    E171 --- E173[ ]
    E172 --- E173
    E173 --- E174[ ]
    E174 --- E175[ ]
    E174 --- E176[ ]
    E175 --- E177[ ]
    E176 --- E177
    E177 --- E178[ ]
    E178 --- E179[ ]
    E178 --- E180[ ]
    E179 --- E181[ ]
    E180 --- E181
    E181 --- E182[ ]
    E182 --- E183[ ]
    E182 --- E184[ ]
    E183 --- E185[ ]
    E184 --- E185
    E185 --- E186[ ]
    E186 --- E187[ ]
    E186 --- E188[ ]
    E187 --- E189[ ]
    E188 --- E189
    E189 --- E190[ ]
    E190 --- E191[ ]
    E190 --- E192[ ]
    E191 --- E193[ ]
    E192 --- E193
    E193 --- E194[ ]
    E194 --- E195[ ]
    E194 --- E196[ ]
    E195 --- E197[ ]
    E196 --- E197
    E197 --- E198[ ]
    E198 --- E199[ ]
    E198 --- E200[ ]
    E199 --- E201[ ]
    E200 --- E201
    E201 --- E202[ ]
    E202 --- E203[ ]
    E202 --- E204[ ]
    E203 --- E205[ ]
    E204 --- E205
    E205 --- E206[ ]
    E206 --- E207[ ]
    E206 --- E208[ ]
    E207 --- E209[ ]
    E208 --- E209
    E209 --- E210[ ]
    E210 --- E211[ ]
    E210 --- E212[ ]
    E211 --- E213[ ]
    E212 --- E213
    E213 --- E214[ ]
    E214 --- E215[ ]
    E214 --- E216[ ]
    E215 --- E217[ ]
    E216 --- E217
    E217 --- E218[ ]
    E218 --- E219[ ]
    E218 --- E220[ ]
    E219 --- E221[ ]
    E220 --- E221
    E221 --- E222[ ]
    E222 --- E223[ ]
    E222 --- E224[ ]
    E223 --- E225[ ]
    E224 --- E225
    E225 --- E226[ ]
    E226 --- E227[ ]
    E226 --- E228[ ]
    E227 --- E229[ ]
    E228 --- E229
    E229 --- E230[ ]
    E230 --- E231[ ]
    E230 --- E232[ ]
    E231 --- E233[ ]
    E232 --- E233
    E233 --- E234[ ]
    E234 --- E235[ ]
    E234 --- E236[ ]
    E235 --- E237[ ]
    E236 --- E237
    E237 --- E238[ ]
    E238 --- E239[ ]
    E238 --- E240[ ]
    E239 --- E241[ ]
    E240 --- E241
    E241 --- E242[ ]
    E242 --- E243[ ]
    E242 --- E244[ ]
    E243 --- E245[ ]
    E244 --- E245
    E245 --- E246[ ]
    E246 --- E247[ ]
    E246 --- E248[ ]
    E247 --- E249[ ]
    E248 --- E249
    E249 --- E250[ ]
    E250 --- E251[ ]
    E250 --- E252[ ]
    E251 --- E253[ ]
    E252 --- E253
    E253 --- E254[ ]
    E254 --- E255[ ]
    E254 --- E256[ ]
    E255 --- E257[ ]
    E256 --- E257
    E257 --- E258[ ]
    E258 --- E259[ ]
    E258 --- E260[ ]
    E259 --- E261[ ]
    E260 --- E261
    E261
```

FD expr

LT expr

RT expr

PU

PD

SEND expr expr

SEND expr

RECV primaryexpr primaryexpr

RECV primaryexpr

DIE

A diagram showing a vertical list of data types. The types are arranged in a column, each in its own rectangular box. From top to bottom, the types are: NUMBER, STRING, BOOLEAN, COLOR, NIL, and TYPE. The boxes are connected by a vertical line on the left and a vertical line on the right, with horizontal lines connecting each box to these vertical lines.

A.2 Propriétés standard

Les agents possèdent certaines propriétés par défaut. Ces propriétés varient selon leur type, et peuvent être de simples valeurs en lecture seule, des fonctions en lecture seule, ou encore des propriétés spéciales ayant une sémantique particulière et requérant un type précis.

A.2.1 Attributs communs à tous les agents

black → #000000

Le noir.

Attribut en lecture seule.

white → #ffffff

Le blanc.

Attribut en lecture seule.

grey → #7f7f7f

Un gris moyen.

Attribut en lecture seule.

red → #ff0000

Le rouge.

Attribut en lecture seule.

green → #00ff00

Le vert.

Attribut en lecture seule.

blue → #0000ff

Le bleu.

Attribut en lecture seule.

yellow → #ffff00

Le jaune.

Attribut en lecture seule.

cyan → #00ffff

Le cyan.

Attribut en lecture seule.

magenta → #ff00ff

Le magenta.

Attribut en lecture seule.

A.2.2 Fonctions communes à tous les agents

print (value) → null

Imprime une valeur sur la sortie standard.

value La valeur à imprimer

Retourne null

println (value) → null

Imprime une valeur dans une nouvelle ligne sur la sortie standard.

value La valeur à imprimer

Retourne null

rand () → number

Retourne un entier positif au hasard.

Retourne un entier positif au hazard

random (min, max) → number

Retourne un entier positif au hasard compris entre les bornes indiquées.

min La valeur borne minimale inclusive

max La valeur borne maximale exclusive

Retourne un entier positif au hazard compris entre les bornes indiquées

type_of (value) → type

Retourne le type d'une valeur.

value La valeur dont on veut obtenir le type

Retourne le type de la valeur

size (table) → number

Retourne le nombre d'éléments contenus dans une table.

table La table dont on veut connaître le nombre d'éléments

Retourne le nombre d'éléments contenus dans la table

A.2.3 Attributs du monde

max_x → number

L'abscisse maximale du monde.

Attribut en lecture seule.

max_y → number

L'ordonnée maximale du monde.

Attribut en lecture seule.

A.2.4 Fonctions du monde

ask_zones (function) → null

Exécute une fonction sur chaque zone.

function La fonction à exécuter sur chaque zone

Retourne null

A.2.5 Attributs des tortues

color → #000000

La couleur de la tortue.

Requiert une valeur de type color.

parent → agent

L'agent qui a créé la tortue.

Attribut en lecture seule.

pos_x → number

L'abscisse de la position de la tortue.

Requiert une valeur de type number.

pos_y → number

L'ordonnée de la position de la tortue.

Requiert une valeur de type number.

pos_angle → number

L'angle de la tortue.

Requiert une valeur de type number.

world → world

Le monde.

Attribut en lecture seule.

zone → zone

La zone dans laquelle est la tortue.

Attribut en lecture seule.

A.2.6 Fonctions des tortues

distance_to (turtle) → number

Retourne la distance la plus courte vers une autre tortue.

turtle La tortue vers laquelle obtenir la distance

Retourne null

face (turtle) → null

Tourne la tortue pour qu'elle fasse face à une autre par le chemin le plus court.

turtle La tortue à laquelle faire face

Retourne null

in_radius (distance) → table

Retourne l'ensemble de tortues dans le rayon donné autour de la tortue.

distance Le rayon autour de la tortue à sonder

Retourne Une table contenant les tortues dans le rayon

inbox () → number

Retourne le nombre de message non lus.

Retourne Le nombre de messages non lus

teleport (x, y, angle) → null

Téléporte une tortue à une certaine coordonnée et avec un certain angle.

x L'abscisse où téléporter la tortue

y L'ordonnée où téléporter la tortue

angle L'angle à donner à la tortue

Retourne null

A.2.7 Attributs des zones

color → #fffff

La couleur de la zone.

Requiert une valeur de type color.

parent → agent

L'agent qui a créé la zone.

Attribut en lecture seule.

world → world

Le monde.

Attribut en lecture seule.

Annexe B

Tutoriel

B.1 Tutoriel

B.1.1 Salut, monde !

Commençons par imprimer du texte.

```
| println("Salut , monde !")
```

Ici l'agent par défaut, le monde, appelle la fonction `println` avec pour paramètre la chaîne de caractères "Salut, monde!", ce qui a pour effet d'imprimer ce texte dans une nouvelle ligne sur la sortie standard.

B.1.2 Les premiers agents

Créons maintenant des agents.

```
| new agent {  
    println("Salut , humain !")  
}
```

Le monde crée un nouvel agent mobile, une tortue, qui apparaîtra alors dans le monde et exécutera le code passé entre accolades.

Il est possible de créer plusieurs tortues exécutant le même code en spécifiant leur nombre.

```
| 5 new agent {  
    println("Salut , humain !")  
}
```

Ainsi, cinq tortues sont créées et chacune d'elles imprime "Salut, humain!".

B.1.3 Dessiner un carré

Les tortues peuvent se déplacer sur le monde en avançant et en tournant à gauche ou à droite. Elles ont également un stylo qu'elles peuvent abaisser ou relever afin de tracer des lignes sur le monde.

```
| new agent {  
    pd  
    fd 50  
    rt 90  
    fd 50  
    rt 90  
    fd 50
```

```

    rt 90
    fd 50
    println("Voici un beau carré !")
}

```

Ici, `pd` demande à la tortue d'abaisser son stylo (pen down), `fd` demande à la tortue d'avancer (forward) d'une certaine distance, et `rt` demande à la tortue de tourner d'un certain nombre de degrés.

B.1.4 Répéter

Afin d'éviter de se répéter, on peut demander à l'interprète de le faire un certain nombre de fois pour nous.

```

new agent {
    pd
    repeat 4 {
        fd 50
        rt 90
    }
    println("Voici qui est mieux. =)")
}

```

B.1.5 Boucler

Il est également possible de boucler tant qu'une condition est vraie.

```

new agent {
    println("Je vais faire ma ronde.")
    while true {
        fd 50
        rt 90
    }
}

```

B.1.6 Agents typés

Il est possible de définir un type d'agent sans en créer, afin d'en créer plus tard.

```

agent personne (nom) {
    println("Je m'appelle " + nom + ".")
}

new personne("Mathieu")
new personne("Michel")

```

Ici, le type d'agent `personne` a été défini. Un type d'agent peut prendre des paramètres exactement de la même manière qu'une fonction.

Ainsi on a pu créer deux tortues de type `personne`, chacune ayant son propre nom.

B.1.7 Fonctions

Il est possible de définir des fonctions. Les fonctions sont définies dans l'espace de nom des propriétés de l'agent.

```

%x_border bounce
%y_border bounce

agent fourmi () {
    function gigoter () {
        rt rand() % 60
        lt rand() % 60
        fd 1
    }

    while true {
        gigoter()
    }
}

new fourmi()

```

Les deux premières lignes seront expliquées un peu plus tard.

On définit ici la fonction `gigoter` pour les agents de type `fourmi`, qui est utilisée un peu plus bas dans le code.

B.1.8 Couleurs

Les tortues ont une couleur qui peut être modifiée. C'est également cette couleur qu'elles utilisent pour dessiner sur le monde.

```

%x_border bounce
%y_border bounce

agent fourmi (couleur) {
    function gigoter () {
        rt rand() % 60
        lt rand() % 60
        fd 1
    }

    color = couleur
    pd

    while true {
        gigoter()
    }
}

new fourmi(red)
new fourmi(blue)

```

B.1.9 Propriétés d'autres agents et parent

Il est possible d'accéder aux propriétés d'autres agents via l'opérateur « `.` ». De plus, il est possible d'accéder à certains agents via des propriétés spéciales, telles que `parent` pour obtenir le parent de l'agent actuel, `world` pour obtenir le monde et `zone` pour obtenir la zone sur laquelle se trouve une tortue.

Ces deux derniers types d'agents seront présentés un peu plus tard.

```
%x_border bounce
%y_border bounce

color = black

agent fourmi (enfants) {
    color = parent.color + #444

    if (enfants > 0) {
        2 new fourmi (enfants - 1)
    }

    function gigoter () {
        rt rand() % 60
        lt rand() % 60
        fd 1
    }

    while true {
        gigoter()
    }
}

new fourmi (2)
```

Dans cet exemple, chaque fourmi définit sa couleur en fonction de celle de son parent en l'obtenant via `parent.color`.

B.1.10 Le monde

Le monde est un agent très particulier : il est unique, immobile, possède une taille, et c'est dans son contexte qu'est exécuté le corps principal d'un programme Stibbons. Ça en fait de fait l'ancêtre commun à tous les autres agents.

```
5 new agent {
    teleport(rand() % world.max_x, rand() % world.max_y, rand())
}
```

Cet exemple crée 5 nouveaux agents et, grâce aux fonctions `teleport` et `rand` et aux propriétés spéciales du monde `max_x` et `max_y`, positionne chaque agent une position et un angle au hasard à l'intérieur des bords du monde.

B.1.11 Les zones

Le monde est constitué de zones qui sont elles aussi des agents. Les zones sont immobiles, colorées, et ont le monde pour parent.

```
%x_border bounce
%y_border bounce

agent fourmi (couleur) {
    teleport(rand() % world.max_x, rand() % world.max_y, rand())
```

```

function gigoter () {
    rt rand() % 60
    lt rand() % 60
    fd 1
}

color = couleur - #444
couleur_zone = couleur + #888

while (true) {
    gigoter()
    zone.color = couleur_zone
}
}

new fourmi(red)
new fourmi(green)
new fourmi(blue)
new fourmi(yellow)
new fourmi(cyan)
new fourmi(magenta)

```

Ici, les fourmis changent la couleur des zones sur lesquelles elles passent.

B.1.12 Directives de monde

Dans cette section seront enfin expliquées les mystérieuses instructions `%x_border bounce` et `%y_border bounce`.

Le monde peut être paramétré au chargement du programme, pour cela on utilise des directives de monde qui doivent toutes être placées en tout début du programme.

Les directives `%world_width` et `%world_height` permettent de définir le nombre de zones constituant le monde en largeur et en hauteur, et doivent être suivi du nombre souhaité. Les directives `%zone_width` et `%zone_height` permettent de définir la largeur et la hauteur des zones constituant le monde, et doivent être suivi de la taille souhaitée. Les directives `%x_border` et `%y_border` permettent de définir le comportement des tortues lorsqu'elles franchissent un bord du monde. Elles doivent être suivies de `none` pour laisser les tortues sortir du monde, de `bounce` pour faire rebondir les tortues contre les bords ou de `wrap` pour reboucler le monde sur lui même par l'axe en question.

```

%world_width 100 // Le monde aura 100 zones en largeur
%world_height 100 // Le monde aura 100 zones en hauteur
%zone_width 5 // Les zones feront 5 unités de large
%zone_height 5 // Les zones feront 5 unités de haut
%x_border wrap // Le monde rebouclera sur lui même sur les bords
                verticaux
%y_border bounce // Le monde sera solide sur les bords horizontaux

```

B.1.13 Messages

Les tortues peuvent s'envoyer des messages. Les messages peuvent être envoyés à un destinataire précis, à un ensemble de destinataires ou à toutes les tortues.

```

destinataire = new agent {
    recv message
}

```

```

    println("Quelqu'un m'a dit : " + message)
}

```

```

new agent {
    send "Coucou !" world.destinataire
}

```

```

5 new agent {
    recv message

    println("Quelqu'un m'a dit : " + message)
}

```

```

new agent {
    send "Oyez, agents !"
}

```

S'il n'y a aucun message dans sa boîte à messages, une tortue demandant la lecture d'un message sera bloquée jusqu'à réception d'un message à lire. Pour éviter un blocage, il est possible de vérifier le nombre de messages présents dans la boîte.

```

new agent {
    new agent {
        send "Bonjour, parent !" parent
    }

    while true {
        if (inbox() > 0) {
            recv message

            println("Quelqu'un m'a dit : " + message)
        }

        rt 1
        fd 1
    }
}

```

Annexe C

Résumés des réunions

C.1 27 janvier 2015

Voici le résumé de la première réunion, le 27 janvier.

Tout d'abord, petite précision et chose à faire rapidement :

- trouver un article à lire sur le multi-agents
- le rapport sera fait en Latex
- il faut créer un dépôt GIT du Sif pour mettre le travail et le rapport

Résumé du projet :

Nous allons développer un système multi-agent sur une même machine, pas de réseau.

Nous avons d'abord parler de faire un ordonnanceur de tâche pour exécuter les agents un à un, comme en NetLOGO, puis nous avons plutôt opter pour l'exécution en parallélisme des tortues, car faire un ordonnanceur causerai des difficultés, (comme comment récupérer la main sur les tortues(utilisation de alarm, mais on ne sait pas qui, ect),et nous préférons utiliser les mécanismes UNIX existant. Nous utiliserons des threads pour programmer les processus tortues plutôt que les forks car ils sont plus modernes, et nous permettront l'usage de moyens de communication comme la mémoire partagée (vs les files de msg avec fork), une meilleure performance, et pour la synchronisation il y aura les mutexs (vs les sémaphores avec fork).

Chaque thread devra interpréter son code.

On aura un thread_main qui aura ses instructions, et la main. Il sera le thread pré-existant, et il y aura des thread_turtles pour les tortues. Nous ferons un MVC, et utiliserons Qt pour l'interface, et C++.

Nous allons procéder de manière incrémentale, avec la méthode AGILE. C'est à dire que nous commencerons par avoir un noyau : une tortue qui s'affiche sur l'interface graphique 2D et qui peut executer 3 instructions simples. Pour commencer, le plan aura une taille fixe. Nous améliorerons l'interpréteur au fur et à mesure et nous ferons des intégrations successives des fonctionnalités.

A faire :

- Quels sont ses moyens de communication des agents ?
- Réfléchir si 1 agent = 1 thread ? thread séquentiel ? système de jeton ?
- Les agents peuvent-ils communiquer avec les patches ? Avec qui peut-on communiquer ?

C.2 3 février 2015

Lors de cette réunion, nous avons principalement discuté du projet, et nous nous sommes mis d'accord sur quelques points.

Tout d'abord, nous utiliserons un thread par tortue, car cela ne devrait pas poser de problème de performance, et cela permettra l'exécution en parallèle des tortues, ce qui change de NetLogo.

Nous avons parlé de mettre une option pour exécuter pas à pas ces threads.

A propos du vocabulaire du projet, nous avons choisi d'appeler les patchs de NetLogo des zones, et pour la tortue d'origine nous la nommons le dieu-tortue.

Pour la communication, elle sera autorisée entre tortues et zones, et éventuellement limitée par la distance.

Pour la partie technique, le main devrait gérer les zones, et la communication se fera avec des files de messages. Nous avons également parlé d'utiliser des classes anonymes pour remplacer la fonction go dans les tortues.

C'est ce qui est ressorti de nos discussions, mais ces décisions ne sont pas encore définitive.

C.3 10 février 2015

Voici la liste des tâches à faire pour la semaine prochaine (si assignées).

C.3.1 Analyse de l'existant

- Logo → Florian
- NetLogo → Clément
- StarLogo → Clément

C.3.2 Analyse des outils

- Gestion de projet :
 - producteev
 - openproj
 - git
 - make
- Tests unitaires → Florian
- C++11 ou supérieur
 - Threads standard C++ → Florian
 - gdb
- Flex, Bison → Julia
- Qt → Adrien
 - Qt en général
 - Dessiner avec Qt
- LaTeX (et Beamer)

C.4 24 février 2015

Lors de cette réunion nous avons tout d'abord fait un point sur le backlog et les taches que nous avons choisi pour le premier sprint :

- interprète simple, qui ne comprend que des instructions basiques ;
- une seule tortue est gérée ;
- pas d'éditeur intégré dans un premier temps mais seulement lecture de code dans un fichier externe.

Nous avons également précisé que le code analysé serait lu en intégralité et interprété, et non pas interprété ligne à ligne, tout du moins dans un premier temps. Ainsi, nous passerons par une phase de pré-compilation afin d'analyser le code source stibbons. Nous aurons donc un schéma du type : code source → tokenizer → analyse et interprétation

Nous définirons dans un premier temps un langage élémentaire, contenant les affectations ainsi que les instructions de bases pour les tortues (forward, turn-right, etc.) et augmenterons

la complexité du langage dans le temps. Il faudra ainsi définir très rapidement une grammaire, l'idéal étant pour la fin de la semaine. Cette tâche est assignée à Florian et Clément.

Pendant ce temps-là, Adrien et Julia sont chargés de définir le modèle. On utilisera pour cela le langage UML, ainsi que le logiciel Umbrello qui permet la génération de code.

Nous avons également abordé la question de la gestion de la bibliographie. Nous utiliserons bibtex dans le rapport pour celle-ci.

C.5 16 mars 2015

Bilan du premier sprint, par rapport à ce qui était prévu, nous notons les différences suivantes :

- le chargement et l'importation n'est pas complet, on doit passer par le terminal ;
- l'interprétation d'un agent se fait, mais limité.

Pour ce qui est de l'estimation des temps des tâches, nous étions globalement au-dessus. Nous avons ajouté quatre nouvelles tâches au backlog :

- ajout de fonctions ;
- ajout des variables ;
- ajout des boucles ;
- ajout des conditionnelles.

Pour la version 0.2 nous avons choisis les tâches suivantes :

- l'utilisateur utilise une variable ;
- l'utilisateur utilise une fonction ;
- les tortues fonctionnent en parallèle ;
- ajout de l'utilisation de breed.

Nous avons défini les tests pour ces tâches.

Nous avons également effectué les estimations de la durée de chaque tâche en écrivant chacun le temps minimum et le temps maximum estimé, puis nous avons fait la moyenne de ces temps, en discutant des raisons des résultats. Nous estimons donc le deuxième sprint à deux semaines, à raison de cinq après-midi de cinq heures par personne (environ 96 heures au total).

C.6 17 mars 2015

Nous avons effectué une réunion avec notre encadrant suite à la fin de notre premier sprint. Certains points ont été soulevés tels que la communication inter-agents et le moyen de l'implémenter mais cela concerne la version 0.3, nous devons donc commencer à y réfléchir sans le réaliser. Le multi-agents de la version 0.2 sera réalisé à l'aide de threads, avec un thread pour chaque tortue.

Le diagramme UML a également été ré-étudié, nous avons donc remarqué certaines différences avec l'implémentation réalisée au cours de la version 0.1, notamment concernant les classes :

- Value ;
- Type ;
- Colored ;
- Boolean ;
- Color ;
- Les classes liées à l'interpréteur (Tree, Interpreter).

Certaines conceptions ont été revues :

- La classe Shape devrait être reliée à une Breed, pas à une Tortue : la Breed définira le comportement d'une tortue ;
- La classe Breed : renommage en TurtleClass ;
- Les agrégations et compositions entre les classes sont à rectifier.

- Pour terminer, deux pistes concernant la gestion de l'arbre syntaxique ont été évoqué :
- Réaliser un arbre de dérivation avec chaque frère droit qui est une instruction ;
 - Un arbre abstrait en UML.

C.7 7 avril 2015

Nous avons fini le second sprint. Nous avons surtout ajouté des choses dans le modèle et dans l'interpréteur : il s'agissait de la prise en charge du multi-threading et des variables.

Coté modèle, on a ajouté les classes `Function`, `Breed`, adapté la classe `Turtle`, ajouté des mutexs dans chaque classe qui le demandait. Nous avons également rajouté un système de "parenté", qui permet à une tortue de connaître son parent et d'avoir accès à ses propriétés.

Coté interprète, il a fallu mettre en place les threads, un à chaque "new agent", et les fonctions. Lors de cette dernière réunion, nous avons discuté de comment afficher les résultats de simulation de notre programme : diagramme, variables tracées avec un journal... Il faudra choisir une simulation de NetLogo, la tester et comparer les résultats des deux applications.

Pour le moment, nous devons redémarrer le programme pour lancer un nouveau fichier. Il faudra utiliser `yywrap`, qui chaîne les fichiers.

Pour la taille du monde, nous pensons à faire une taille par défaut si l'utilisateur n'en choisit pas, et lui donner la possibilité d'en choisir une en début de fichier grâce à une syntaxe différente du `stibbons`.

Annexe D

Hierarchical Index

D.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically :

BorderType	73
stibbons : :Breed	73
stibbons : :Changeable	75
stibbons : :World	147
enable_shared_from_this	
stibbons : :Agent	66
stibbons : :Turtle	129
stibbons : :World	147
stibbons : :Zone	156
exception	
stibbons : :exit_requested_exception	81
stibbons : :InterpreterException	91
stibbons : :SemanticException	112
stibbons : :SyntaxException	122
stibbons : :Interpreter	87
stibbons : :TurtleInterpreter	137
stibbons : :WorldInterpreter	152
stibbons : :InterpreterManager	92
stibbons : :Line	94
stibbons : :Parser	102
stibbons : :Point	102
QApplication	
stibbons : :Application	69
QCoreApplication	
stibbons : :Application	69
QMainWindow	
stibbons : :Window	146
QPlainTextEdit	
StibbonsEditor	119
QSyntaxHighlighter	
StibbonsHighlighter	120
QThread	
stibbons : :Runner	110
QWidget	

LineNumberArea	96
stibbons : :WorldView	155
stibbons : :SimpleValue< T >	114
stibbons : :SimpleValue< bool >	114
stibbons : :Boolean	72
stibbons : :SimpleValue< double >	114
stibbons : :Number	97
stibbons : :SimpleValue< std : :string >	114
stibbons : :String	121
stibbons : :SimpleValue< Type >	114
stibbons : :TypeValue	140
stibbons : :Singleton< T >	116
stibbons : :AskZonesFunction	71
stibbons : :Singleton< AskZonesFunction >	116
stibbons : :Singleton< DistanceToFunction >	116
stibbons : :DistanceToFunction	80
stibbons : :Singleton< FaceFunction >	116
stibbons : :FaceFunction	81
stibbons : :Singleton< InboxFunction >	116
stibbons : :InboxFunction	85
stibbons : :Singleton< InRadiusFunction >	116
stibbons : :InRadiusFunction	86
stibbons : :Singleton< Nil >	116
stibbons : :Nil	96
stibbons : :Singleton< PrintFunction >	116
stibbons : :PrintFunction	107
stibbons : :Singleton< PrintlnFunction >	116
stibbons : :PrintlnFunction	108
stibbons : :Singleton< RandFunction >	116
stibbons : :RandFunction	108
stibbons : :Singleton< RandomFunction >	116
stibbons : :RandomFunction	110
stibbons : :Singleton< SendFunction >	116
stibbons : :SendFunction	114
stibbons : :Singleton< SizeFunction >	116
stibbons : :SizeFunction	118
stibbons : :Singleton< TeleportFunction >	116
stibbons : :TeleportFunction	126
stibbons : :Singleton< TypeOfFunction >	116
stibbons : :TypeOfFunction	138
stibbons : :Size	117
stibbons : :Tree	127
Type	138
stibbons : :Value	142
stibbons : :GenericValue< Type : :BOOLEAN >	85
stibbons : :Boolean	72
stibbons : :GenericValue< Type : :COLOR >	85

stibbons : :Color	76
stibbons : :GenericValue< Type : :FUNCTION >	85
stibbons : :Function	83
stibbons : :AskZonesFunction	71
stibbons : :DistanceToFunction	80
stibbons : :FaceFunction	81
stibbons : :InboxFunction	85
stibbons : :InRadiusFunction	86
stibbons : :PrintFunction	107
stibbons : :PrintlnFunction	108
stibbons : :RandFunction	108
stibbons : :RandomFunction	110
stibbons : :SendFunction	114
stibbons : :SizeFunction	118
stibbons : :TeleportFunction	126
stibbons : :TypeOfFunction	138
stibbons : :UserFunction	141
stibbons : :GenericValue< Type : :NIL >	85
stibbons : :Nil	96
stibbons : :GenericValue< Type : :NUMBER >	85
stibbons : :Number	97
stibbons : :GenericValue< Type : :STRING >	85
stibbons : :String	121
stibbons : :GenericValue< Type : :TABLE >	85
stibbons : :Table	123
stibbons : :GenericValue< Type : :TYPE >	85
stibbons : :TypeValue	140
stibbons : :Agent	66
stibbons : :GenericValue< T >	85
stibbons : :WorldPainter	154
yyFlexLexer	
stibbons : :FlexScanner	82

Annexe E

Class Index

E.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions :

stibbons : :Agent	
Class Agent in stibbons	66
stibbons : :Application	
The headless application	69
stibbons : :AskZonesFunction	
A class applying a function to every zone	71
stibbons : :Boolean	
A class representing a boolean	72
BorderType	
The type of borders	73
stibbons : :Breed	
A class representing a breed	73
stibbons : :Changeable	
A class which can trigger a callback when its object has changed	75
stibbons : :Color	
A color	76
stibbons : :DistanceToFunction	
A class returning the distance to another turtle	80
stibbons : :exit_requested_exception	81
stibbons : :FaceFunction	
A class making a turtle face another	81
stibbons : :FlexScanner	
Class for lexical analysis	82
stibbons : :Function	
A class representing a function	83
stibbons : :GenericValue< T >	
An template class easing the implementation of a Value class	85
stibbons : :InboxFunction	
A class returning the number of unread messages	85
stibbons : :InRadiusFunction	
A class making a turtle face another	86
stibbons : :Interpreter	
Class that will interpret stibbons language	87
stibbons : :InterpreterException	
Abstraction of interpreter exceptions	91

stibbons : :InterpreterManager	
Class that will interpret stibbons language	92
stibbons : :Line	
A colored polyline	94
LineNumberArea	
Widget that contain the line number area	96
stibbons : :Nil	
A class to represent the null value	96
stibbons : :Number	
A class representing a real number	97
stibbons : :Parser	
Class for syntactic analysis	102
stibbons : :Point	
A point with a parametrable number of dimensions	102
stibbons : :PrintFunction	
A class printing on the standard output	107
stibbons : :PrintlnFunction	
A class printing a new line on the standard output	108
stibbons : :RandFunction	
A class returning a random number	108
stibbons : :RandomFunction	
A class returning a random number in a range	110
stibbons : :Runner	
Bridge between the interpreter and Qt application	110
stibbons : :SemanticException	
Exception thrown when a semantic error occurred	112
stibbons : :SendFunction	
A class sending a message from a turtle to another	114
stibbons : :SimpleValue< T >	
A class representing a simple value	114
stibbons : :Singleton< T >	
A helper class to implementing a singleton	116
stibbons : :Size	
A size with a parametrable number of dimensions	117
stibbons : :SizeFunction	
A class that return an array size	118
StibbonsEditor	
The editor class in Qt application	119
StibbonsHighlighter	
The highlighter class that highlight code in Qt application	120
stibbons : :String	
A class representing a string	121
stibbons : :SyntaxException	
Exception thrown when a syntax error occurred	122
stibbons : :Table	
Class Table in stibbons	123
stibbons : :TeleportFunction	
A class teleporting a turtle to another location	126
stibbons : :Tree	
Class that will represent a syntactic tree	127

stibbons : :Turtle	
Class Turtle stibbons	129
stibbons : :TurtleInterpreter	137
Type	
The available value types	138
stibbons : :TypeOfFunction	
A class returning the type of a value	138
stibbons : :TypeValue	
A class representing a type	140
stibbons : :UserFunction	
A class representing a user defined function	141
stibbons : :Value	
An abstract class who represent a values of a given type	142
stibbons : :Window	
The window of Qt application	146
stibbons : :World	
A world consisting of zones, breeds, turtles and lines	147
stibbons : :WorldInterpreter	
Class that will interpret stibbons language	152
stibbons : :WorldPainter	
The painter of the world in Qt application	154
stibbons : :WorldView	
The view of the world in Qt application	155
stibbons : :Zone	
Class Zone	156

Annexe F

File Index

F.1 File List

Here is a list of all documented files with brief descriptions :

src/cli/ application.h	??
src/interpreter/flex-scanner.h	
FlexScanner class header	160
src/interpreter/interpreter-exception.h	
InterpreterException class header	160
src/interpreter/interpreter-manager.h	
InterpreterManager class header	161
src/interpreter/ interpreter.h	??
src/interpreter/parser.h	
Parser class header	162
src/interpreter/semantic-exception.h	
SemanticException class header	162
src/interpreter/syntax-exception.h	
SyntaxException class header	163
src/interpreter/tree.h	
Interpreter class header	163
src/interpreter/ turtle-interpreter.h	??
src/interpreter/ world-interpreter.h	??
src/model/agent.h	
A class representing an agent	165
src/model/boolean.h	
A class representing a boolean	165
src/model/border-type.h	
The BorderType header class	166
src/model/breed.h	
A class representing a breed	166
src/model/changeable.h	
The Changeable header class	167
src/model/color.h	
A color	167
src/model/function.h	
A class representing a function	168
src/model/line.h	
A colored polyline	168

src/model/nil.h	
A class to represent the null value	169
src/model/number.h	
A class representing a real number	170
src/model/point.h	
A point with a parametrable number of dimensions	170
src/model/simple-value.h	
An abstract class who represent a values of a given type	171
src/model/singleton.h	
A class representing a singleton	171
src/model/size.h	
A size with a parametrable number of dimensions	172
src/model/standard-function.h	
Classes implementing standard functions	172
src/model/string.h	
A class representing a string	173
src/model/ table.h	??
src/model/turtle.h	
The Turtle class header	174
src/model/type-value.h	
A class representing a type	175
src/model/ type.h	??
src/model/user-function.h	
A class representing a user defined function	175
src/model/value.h	
An abstract class who represent a values of a given type	176
src/model/world.h	
The World class header	176
src/model/zone.h	
The Zone header class	177
src/qt/ application.h	??
src/qt/line-number-area.h	
The LineNumberArea class header	178
src/qt/runner.h	
The Runner class header	179
src/qt/ stibbons-editor.h	??
src/qt/stibbons-highlighter.h	
The Highlighter header class	179
src/qt/window.h	
The Stibbons main window	180
src/qt/world-painter.h	
The Stibbons world painter	181
src/qt/world-view.h	
The Stibbons world view	181

Annexe G

Class Documentation

G.1 stibbons : :Agent Class Reference

```
class Agent in stibbons
#include <agent.h>
Inheritance diagram for stibbons : :Agent :
```

Public Member Functions

- virtual ~Agent ()
- AgentPtr getParent ()
- virtual WorldPtr getWorld ()=0
- void reparent ()
- void unparent ()
- virtual void setProperty (string key, ValuePtr value)
- virtual ValuePtr getProperty (string p)
- virtual unordered_map< string, ValuePtr > * getProperty ()
- virtual double compare (ValuePtr other)
- virtual double compare (AgentPtr other)
- virtual void exportProperties (Object *o)
- virtual void destroy ()

Protected Member Functions

- Agent (AgentPtr parent=nullptr)
- void init ()

Protected Attributes

- recursive_mutex **value_m**
- recursive_mutex **parent_m**

G.1.1 Detailed Description

```
class Agent in stibbons
```

G.1.2 Constructor & Destructor Documentation

```
virtual stibbons : :Agent : :~Agent ( ) [virtual]
```

Empty Destructor

```
stibbons : :Agent : :Agent (  AgentPtr parent = nullptr  )  [protected]
```

Empty Constructor

Parameters

<i>parent</i>	an unowned reference to the agent's parent
---------------	--

G.1.3 Member Function Documentation

virtual double stibbons : :Agent : :compare (ValuePtr *other*) [virtual]

Return whether the comparison value of two values

Return a negative number if this value is lower than the other, return a positive number if this value is greater than the other, return 0 if they are equal.

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the comparison value of this and the other value

Reimplemented from stibbons : :Value.

virtual double stibbons : :Agent : :compare (AgentPtr *other*) [virtual]

Return whether the comparison value of two values

Return a negative number if this value is lower than the other, return a positive number if this value is greater than the other, return 0 if they are equal.

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the comparison value of this and the other value

virtual void stibbons : :Agent : :destroy () [virtual]

Remove as much references to this agent as possible

Reimplemented in stibbons : :Turtle.

virtual void stibbons : :Agent : :exportProperties (Object * *o*) [virtual]

Export the properties in the object *o*

Parameters

<i>a</i>	reference on an object (jsan spirit type)
----------	---

AgentPtr stibbons : :Agent : :getParent ()

Get the parent of the agent

Returns

the parent of the agent

virtual ValuePtr stibbons : :Agent : :getProperty (string *p*) [virtual]

Get the value of the propertie *p*

Returns

the value of propertie p

Reimplemented in `stibbons : :Zone`, `stibbons : :Turtle`, and `stibbons : :World`.

virtual unordered_map<string,ValuePtr>* stibbons : :Agent : :getProperty ()
[virtual]

Get the map of the properties

Returns

the map of properties

virtual WorldPtr stibbons : :Agent : :getWorld () [pure virtual]

Get the value of world

Returns

the value of world

Implemented in `stibbons : :Turtle`, `stibbons : :Zone`, and `stibbons : :World`.

void stibbons : :Agent : :init () [protected]

Initialize the agent

void stibbons : :Agent : :reparent ()

Reparent to the grand parent agent

virtual void stibbons : :Agent : :setProperty (string *key*, ValuePtr *value*)
[virtual]

Add a property

Parameters

<i>key</i>	the key of the property
<i>value</i>	the value of the property

Reimplemented in `stibbons : :Zone`, `stibbons : :Turtle`, and `stibbons : :World`.

void stibbons : :Agent : :unparent ()

Remove the parent

The documentation for this class was generated from the following file :

— `src/model/agent.h`

G.2 stibbons : :Application Class Reference

The headless application.

#include <application.h>

Inheritance diagram for `stibbons : :Application` :

Public Slots

— `void exportModel ()`

Public Member Functions

- Application (int &argc, char **argv)
- ~Application ()
- int exec ()
- void setProgram (std::string program)
- void setExportInterval (size_t seconds)
- void setExportPrefix (std::string prefix)
- void setRenderPNG (bool value)
- void setExportJSON (bool value)
- Application (int &argc, char **argv)
- int exec ()

G.2.1 Detailed Description

The headless application.

The application itself.

Author

Adrien Plazas

G.2.2 Constructor & Destructor Documentation

stibbons : :Application : :Application (int & argc, char ** argv)

Constructor

Parameters

<i>argc</i>	the arguments' number
<i>argv</i>	the arguments' list

stibbons : :Application : :~Application ()

Destructor

stibbons : :Application : :Application (int & argc, char ** argv)

Constructor

Parameters

<i>argc</i>	the arguments' number
<i>argv</i>	the arguments' list

G.2.3 Member Function Documentation

int stibbons : :Application : :exec ()

Run the application

Returns

the return state of the application

int stibbons : :Application : :exec ()

Run the application

Returns

the return state of the application

void stibbons : :Application : :exportModel () [slot]

Method called to export

void stibbons : :Application : :setExportInterval (size_t *seconds*)

Set the export interval

Parameters

<i>second</i>	the interval between two exports
---------------	----------------------------------

void stibbons : :Application : :setExportJSON (bool *value*)

Set if the JSON text is exported

Parameters

<i>value</i>	a boolean to determine the export or not
--------------	--

void stibbons : :Application : :setExportPrefix (std : :string *prefix*)

Set the export prefix

Parameters

<i>prefix</i>	the prefix to determine what need to be exported
---------------	--

void stibbons : :Application : :setProgram (std : :string *program*)

Set the program to execute

Parameters

<i>program</i>	the program to execute
----------------	------------------------

void stibbons : :Application : :setRenderPNG (bool *value*)

Set if the png image is rendered

Parameters

<i>value</i>	a boolean to determine the export or not
--------------	--

The documentation for this class was generated from the following file :

— src/cli/application.h

G.3 stibbons : :AskZonesFunction Class Reference

A class applying a function to every zone.

#include <standard-function.h>

Inheritance diagram for stibbons : :AskZonesFunction :

Public Member Functions

— virtual ValuePtr exec (AgentPtr agent, TablePtr params)

Additional Inherited Members

G.3.1 Detailed Description

A class applying a function to every zone.

class AskZonesFunction

Author

Adrien Plazas

G.3.2 Member Function Documentation

virtual ValuePtr stibbons : :AskZonesFunction : :exec (AgentPtr *agent*, TablePtr *params*) [virtual]

Execute the function

Parameters

<i>an</i>	Agent pointer to execute on
<i>a</i>	Table pointer with the parameters

Implements stibbons : :Function.

The documentation for this class was generated from the following file :

— src/model/standard-function.h

G.4 stibbons : :Boolean Class Reference

A class representing a boolean.

#include <boolean.h>

Inheritance diagram for stibbons : :Boolean :

Public Member Functions

- virtual void reset ()
- virtual double compare (ValuePtr other)
- virtual double compare (BooleanPtr other)
- virtual string toString ()

Additional Inherited Members

G.4.1 Detailed Description

A class representing a boolean.

class Boolean

Author

Julia Bassoumi, Adrien Plazas

G.4.2 Member Function Documentation

virtual double stibbons : :Boolean : :compare (ValuePtr *other*) [virtual]

Return whether the comparison value of two values

Return a negative number if this value is lower than the other, return a negative number if this value is greater than the other, return 0 if they are equal.

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the comparison value of this and the other value

Reimplemented from `stibbons : :Value`.

virtual double stibbons : :Boolean : :compare (BooleanPtr *other*) [virtual]

Return whether the comparison value of two values

Return a negative number if this value is lower than the other, return a positive number if this value is greater than the other, return 0 if they are equal.

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the comparison value of this and the other value

virtual void stibbons : :Boolean : :reset () [virtual]

Reset to the default value

Implements `stibbons : :SimpleValue< bool >`.

virtual string stibbons : :Boolean : :toString () [virtual]

Return a string corresponding to the value

Returns

a string corresponding to the value

Implements `stibbons : :Value`.

The documentation for this class was generated from the following file :

— `src/model/boolean.h`

G.5 BorderType Class Reference

The type of borders.

`#include <border-type.h>`

G.5.1 Detailed Description

The type of borders.

Author

Adrien Plazas

The documentation for this class was generated from the following file :

— `src/model/border-type.h`

G.6 stibbons : :Breed Class Reference

A class representing a breed.

`#include <breed.h>`

Public Member Functions

- Breed (FunctionPtr function)
- TurtlePtr createTurtle ()
- TurtlePtr createTurtle (AgentPtr parent)
- unordered_set< TurtlePtr > getTurtles ()
- WorldPtr getWorld ()
- FunctionPtr getFunction ()

G.6.1 Detailed Description

A class representing a breed.

class Breed

Author

Julia Bassoumi, Adrien Plazas

G.6.2 Constructor & Destructor Documentation

stibbons : :Breed : :Breed (FunctionPtr *function*)

Create a breed from a function.

The breed will own the function and destroy it when it is deleted.

Parameters

<i>function</i>	the function of the breed
-----------------	---------------------------

G.6.3 Member Function Documentation

TurtlePtr stibbons : :Breed : :createTurtle ()

Create and add a new turtle to the breed

The parent of the turtle will be the breed's world.

Returns

a reference to the new turtle

TurtlePtr stibbons : :Breed : :createTurtle (AgentPtr *parent*)

Create and add a new turtle to the breed

Parameters

<i>parent</i>	the parent of the turtle
---------------	--------------------------

Returns

a reference to the new turtle

FunctionPtr stibbons : :Breed : :getFunction ()

Get the function

Returns

the function of the breed

unordered_set<TurtlePtr> stibbons : :Breed : :getTurtles ()

Get the turtles

Returns

a set containing the turtles of the breed

WorldPtr stibbons : :Breed : :getWorld ()

Get the world

Returns

the world of the breed

The documentation for this class was generated from the following file :

— src/model/breed.h

G.7 stibbons : :Changeable Class Reference

A class which can trigger a callback when its object has changed.

#include <changeable.h>

Inheritance diagram for stibbons : :Changeable :

Public Member Functions

- Changeable ()
- void onChanged (std : :function< void()> callback)
- void changed ()

G.7.1 Detailed Description

A class which can trigger a callback when its object has changed.

Author

Adrien

G.7.2 Constructor & Destructor Documentation

stibbons : :Changeable : :Changeable ()

Create a changeable

G.7.3 Member Function Documentation

void stibbons : :Changeable : :changed ()

Signal that an object has changed

void stibbons : :Changeable : :onChanged (std : :function< void()> *callback*)

Set the callback function to trigger when the object has changed

Only one callback function can be set at a time

Parameters

<i>callback</i>	the callback function
-----------------	-----------------------

The documentation for this class was generated from the following file :

— src/model/changeable.h

G.8 stibbons : :Color Class Reference

A color.

```
#include <color.h>
```

Inheritance diagram for stibbons : :Color :

Public Member Functions

- Color (Color &other)
- Color (Color &&other)
- Color & operator= (Color &other)
- Color & operator= (Color &&other)
- Color (double red=0.0, double green=0.0, double blue=0.0)
- Color (std : :string color) throw (std : :domain_error)
- void r (double red)
- void g (double green)
- void b (double blue)
- double r ()
- double g ()
- double b ()
- virtual ValuePtr add (ValuePtr other) throw (std : :domain_error)
- virtual ColorPtr add (ColorPtr other)
- virtual ValuePtr subtract (ValuePtr other) throw (std : :domain_error)
- virtual ColorPtr subtract (ColorPtr other)
- virtual ValuePtr multiply (ValuePtr other) throw (std : :domain_error)
- virtual ColorPtr multiply (ColorPtr other)
- virtual ValuePtr divide (ValuePtr other) throw (std : :domain_error)
- virtual ColorPtr divide (ColorPtr other)
- virtual double compare (ValuePtr other)
- virtual double compare (ColorPtr other)
- virtual std : :string toString ()

G.8.1 Detailed Description

A color.

Author

Adrien Plazas

G.8.2 Constructor & Destructor Documentation

stibbons : :Color : :Color (Color & *other*)

Create a copy of a color

Parameters

<i>other</i>	the other color
--------------	-----------------

stibbons : :Color : :Color (Color && *other*)

Move a color

Parameters

<i>other</i>	the other color
--------------	-----------------

stibbons : :Color : :Color (double *red* = 0.0, double *green* = 0.0, double *blue* = 0.0)

Create a color from its red, green and blue values.

Parameters

<i>red</i>	the value of the red composant
<i>green</i>	the value of the green composant
<i>blue</i>	the value of the blue composant

stibbons : :Color : :Color (std : :string *color*) throw std : :domain_error)

Creates a color from an HTML style color string.

Parameters

<i>color</i>	a stirng like #FFF or #FFFFFF
--------------	-------------------------------

G.8.3 Member Function Documentation

virtual ValuePtr stibbons : :Color : :add (ValuePtr *other*) throw std : :domain_error) [virtual]

Add a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

Reimplemented from stibbons : :Value.

virtual ColorPtr stibbons : :Color : :add (ColorPtr *other*) [virtual]

Add a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

void stibbons : :Color : :b (double *blue*)

Set the value of blue

Parameters

<i>blue</i>	the new value of blue
-------------	-----------------------

double stibbons : :Color : :b ()

Return the value of blue

Returns

the value of blue

virtual double stibbons : :Color : :compare (ValuePtr *other*) [virtual]

Return whether the comparison value of two values

Return a negative number if this value is lower than the other, return a positive number if this value is greater than the other, return 0 if they are equal.

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the comparison value of this and the other value

Reimplemented from stibbons : :Value.

virtual double stibbons : :Color : :compare (ColorPtr *other*) [virtual]

Return whether the comparison value of two values

Return a negative number if this value is lower than the other, return a negative number if this value is greater than the other, return 0 if they are equal.

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the comparison value of this and the other value

virtual ValuePtr stibbons : :Color : :divide (ValuePtr *other*) throw std : :domain__error) [virtual]

Divide a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

Reimplemented from stibbons : :Value.

virtual ColorPtr stibbons : :Color : :divide (ColorPtr *other*) [virtual]

Divide a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

void stibbons : :Color : :g (double *green*)

Set the value of green

Parameters

<i>green</i>	the new value of green
--------------	------------------------

double stibbons : :Color : :g ()

Return the value of green

Returns

the value of green

**virtual ValuePtr stibbons : :Color : :multiply (ValuePtr *other*) throw
std : :domain_error) [virtual]**

Multiply a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

Reimplemented from stibbons : :Value.

virtual ColorPtr stibbons : :Color : :multiply (ColorPtr *other*) [virtual]

Multiply a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

Color& stibbons : :Color : :operator= (Color & *other*)

Copy of a color

Parameters

<i>other</i>	the other color
--------------	-----------------

Color& stibbons : :Color : :operator= (Color && *other*)

Move a color

Parameters

<i>other</i>	the other color
--------------	-----------------

void stibbons : :Color : :r (double *red*)

Set the value of red

Parameters

<i>red</i>	the new value of red
------------	----------------------

double stibbons : :Color : :r ()

Return the value of red

Returns

the value of red

**virtual ValuePtr stibbons : :Color : :substract (ValuePtr *other*) throw
std : :domain_error) [virtual]**

Subtract a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

Reimplemented from stibbons : :Value.

virtual ColorPtr stibbons : :Color : :substract (ColorPtr *other*) [virtual]

Subtract a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

virtual std : :string stibbons : :Color : :toString () [virtual]

Return a string corresponding to the value

Returns

a string corresponding to the value

Implements stibbons : :Value.

The documentation for this class was generated from the following file :

— src/model/color.h

G.9 stibbons : :DistanceToFunction Class Reference

A class returning the distance to another turtle.

#include <standard-function.h>

Inheritance diagram for stibbons : :DistanceToFunction :

Public Member Functions

— virtual ValuePtr exec (AgentPtr agent, TablePtr params)

Additional Inherited Members

G.9.1 Detailed Description

A class returning the distance to another turtle.
class DistanceToFunction

Author

Adrien Plazas

G.9.2 Member Function Documentation

virtual ValuePtr stibbons : :DistanceToFunction : :exec (AgentPtr *agent*,
TablePtr *params*) [virtual]

Execute the function

Parameters

<i>agent</i>	the agent to execute the function on
<i>params</i>	the parameters to execute the function with

Returns

the returned value

Implements stibbons : :Function.

The documentation for this class was generated from the following file :

— src/model/standard-function.h

G.10 stibbons : :exit_requested_exception Class Reference

Inheritance diagram for stibbons : :exit_requested_exception :

The documentation for this class was generated from the following file :

— src/interpreter/interpreter-manager.h

G.11 stibbons : :FaceFunction Class Reference

A class making a turtle face another.

#include <standard-function.h>

Inheritance diagram for stibbons : :FaceFunction :

Public Member Functions

— virtual ValuePtr exec (AgentPtr agent, TablePtr params)

Additional Inherited Members

G.11.1 Detailed Description

A class making a turtle face another.

class FaceFunction

Author

Adrien Plazas

G.11.2 Member Function Documentation

virtual ValuePtr stibbons : :FaceFunction : :exec (AgentPtr *agent*, TablePtr *params*) [virtual]

Execute the function

Parameters

<i>an</i>	Agent pointer to execute on
<i>a</i>	Table pointer with the parameters

Implements stibbons : :Function.

The documentation for this class was generated from the following file :

— src/model/standard-function.h

G.12 stibbons : :FlexScanner Class Reference

Class for lexical analysis.

#include <flex-scanner.h>

Inheritance diagram for stibbons : :FlexScanner :

Public Member Functions

— FlexScanner (std : :istream *in)

— int yylex (yy : :parser : :semantic_type *pyylval, yy : :parser : :location_type *loc)

G.12.1 Detailed Description

Class for lexical analysis.

This class contain an overload of yylex function with parameters used by our lexical parser.

Author

Florian Galinier

G.12.2 Constructor & Destructor Documentation

stibbons : :FlexScanner : :FlexScanner (std : :istream * *in*) [inline]

FlexScanner constructor.

Parameters

<i>in</i>	The stream that will be parsed by yylex() method.
-----------	---

G.12.3 Member Function Documentation

int stibbons : :FlexScanner : :yylex (yy : :parser : :semantic_type * *pyylval*, yy : :parser : :location_type * *loc*) [inline]

Return the next token.

Parameters

out	<i>pyylval</i>	The value of the token.
out	<i>loc</i>	Location of the token.

Returns

The last yy : :parser : :token found or 0 if end of stream.

The documentation for this class was generated from the following file :

— src/interpreter/flex-scanner.h

G.13 stibbons : :Function Class Reference

A class representing a function.

#include <function.h>

Inheritance diagram for stibbons : :Function :

Public Member Functions

- Function (vector< string > params=vector< string >())
- virtual ~Function ()=default
- virtual ValuePtr exec (AgentPtr agent, TablePtr params)=0
- virtual ValuePtr operator() (AgentPtr agent, TablePtr params)
- vector< string > getParams ()
- virtual double compare (ValuePtr other)
- virtual double compare (FunctionPtr other)
- virtual string toString ()

Protected Attributes

- vector< string > **parameters**

G.13.1 Detailed Description

A class representing a function.

class Function

Author

Adrien Plazas

G.13.2 Constructor & Destructor Documentation

stibbons : :Function : :Function (vector< string > *params* = vector< string >())

Constructor

Parameters

<i>params</i>	the parameters that the function expect to be executed with
---------------	---

virtual stibbons : :Function : :~Function () [virtual], [default]

Destructor

G.13.3 Member Function Documentation

virtual double stibbons : :Function : :compare (ValuePtr *other*) [virtual]

Return whether the comparison value of two values

Return a negative number if this value is lower than the other, return a negative number if this value is greater than the other, return 0 if they are equal.

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the comparison value of this and the other value

Reimplemented from stibbons : :Value.

virtual double stibbons : :Function : :compare (FunctionPtr *other*) [virtual]

Return whether the comparison value of two values

Return a negative number if this value is lower than the other, return a positive number if this value is greater than the other, return 0 if they are equal.

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the comparison value of this and the other value

virtual ValuePtr stibbons : :Function : :exec (AgentPtr *agent*, TablePtr *params*) [pure virtual]

Execute the function

Parameters

<i>agent</i>	the agent to execute the function on
<i>params</i>	the parameters to execute the function with

Returns

the returned value

Implemented in stibbons : :SizeFunction, stibbons : :AskZonesFunction, stibbons : :InRadiusFunction, stibbons : :FaceFunction, stibbons : :DistanceToFunction, stibbons : :InboxFunction, stibbons : :SendFunction, stibbons : :TeleportFunction, stibbons : :PrintlnFunction, stibbons : :-PrintFunction, stibbons : :RandomFunction, stibbons : :UserFunction, stibbons : :RandFunction, and stibbons : :TypeOfFunction.

vector<string> stibbons : :Function : :getParams ()

Get the parameters of the function

Returns

the parameters of the function

virtual ValuePtr stibbons : :Function : :operator() (AgentPtr *agent*, TablePtr *params*) [virtual]

Execute the function

Parameters

<i>agent</i>	the agent to execute the function on
<i>params</i>	the parameters to execute the function with

Returns

the returned value

virtual string stibbons : :Function : :toString () [virtual]

Return a string corresponding to the value

Returns

a string corresponding to the value

Implements stibbons : :Value.

The documentation for this class was generated from the following file :

— src/model/function.h

G.14 stibbons : :GenericValue< T > Class Template Reference

An template class easing the implementation of a Value class.

#include <value.h>

Inheritance diagram for stibbons : :GenericValue< T > :

Public Member Functions

— virtual Type getType () const

G.14.1 Detailed Description

template<Type T>class stibbons : :GenericValue< T >

An template class easing the implementation of a Value class.

Author

Adrien Plazas

G.14.2 Member Function Documentation

**template<Type T> virtual Type stibbons : :GenericValue< T > : :getType ()
const [inline], [virtual]**

Return the type of the value

Returns

a value of the enumeration Type

Implements stibbons : :Value.

The documentation for this class was generated from the following file :

— src/model/value.h

G.15 stibbons : :InboxFunction Class Reference

A class returning the number of unread messages.

#include <standard-function.h>

Inheritance diagram for stibbons : :InboxFunction :

Public Member Functions

— virtual ValuePtr exec (AgentPtr agent, TablePtr params)

Additional Inherited Members

G.15.1 Detailed Description

A class returning the number of unread messages.

class InboxFunction

Author

Adrien Plazas

G.15.2 Member Function Documentation

virtual ValuePtr stibbons : :InboxFunction : :exec (AgentPtr *agent*, TablePtr *params*) [virtual]

Execute the function

Parameters

<i>an</i>	Agent pointer to execute on
<i>a</i>	Table pointer with the parameters

Implements stibbons : :Function.

The documentation for this class was generated from the following file :

— src/model/standard-function.h

G.16 stibbons : :InRadiusFunction Class Reference

A class making a turtle face another.

#include <standard-function.h>

Inheritance diagram for stibbons : :InRadiusFunction :

Public Member Functions

— virtual ValuePtr exec (AgentPtr agent, TablePtr params)

Additional Inherited Members

G.16.1 Detailed Description

A class making a turtle face another.

class InRadiusFunction

Author

Adrien Plazas

G.16.2 Member Function Documentation

virtual ValuePtr stibbons : :InRadiusFunction : :exec (AgentPtr *agent*, TablePtr *params*) [virtual]

Execute the function

Parameters

<i>agent</i>	the agent to execute the function on
<i>params</i>	the parameters to execute the function with

Returns

the returned value

Implements `stibbons : :Function`.

The documentation for this class was generated from the following file :

— `src/model/standard-function.h`

G.17 `stibbons : :Interpreter` Class Reference

Class that will interpret stibbons language.

`#include <interpreter.h>`

Inheritance diagram for `stibbons : :Interpreter` :

Public Member Functions

- `Interpreter ()=default`
- `virtual TablePtr getParams (InterpreterManager &manager, FunctionPtr, AgentPtr, const TreePtr, TablePtr, std : :string)`
- `virtual ValuePtr interpretFunction (FunctionPtr, AgentPtr, TablePtr)`
- `virtual FunctionPtr getFunctionFromTree (InterpreterManager &manager, const TreePtr)`
- `ValuePtr affectationOp (InterpreterManager &manager, AgentPtr, TreePtr, TablePtr)`
- `TablePtr newOp (InterpreterManager &manager, AgentPtr, TreePtr, TablePtr)`
- `ValuePtr callOp (InterpreterManager &manager, AgentPtr, TreePtr, TablePtr)`
- `virtual ValuePtr interpret (InterpreterManager &manager, AgentPtr agent, const TreePtr, TablePtr hashTable=nullptr)`

Static Public Member Functions

- `static yy : :position getPosition (const TreePtr)`

Public Attributes

- `mutex suspendMutex`
- `bool inPauseFlag`

G.17.1 Detailed Description

Class that will interpret stibbons language.

This class will parse the standart input, generate a syntactic tree and interpret it.

Author

Florian Galinier
Clément Simon

G.17.2 Constructor & Destructor Documentation

`stibbons : :Interpreter : :Interpreter () [default]`

Create a new interpreter

G.17.3 Member Function Documentation

ValuePtr *stibbons* : **Interpreter** : **affectationOp** (**InterpreterManager** & *manager*, **AgentPtr** , **TreePtr** , **TablePtr**) [inline]

Do the affectation operation if token detected is '='

Parameters

<i>agent</i>	The agent to run the program on.
<i>tree</i>	The syntactic tree to interpret.
<i>hashTable</i>	A hashtable which contain parameters

Returns

The result of the '=' operation on concerned value.

ValuePtr stibbons : :Interpreter : :callOp (InterpreterManager & *manager*, AgentPtr , TreePtr , TablePtr) [inline]

Do the calling operation if token detected is 'CALL'

Parameters

<i>agent</i>	The agent to run the program on.
<i>tree</i>	The syntactic tree to interpret.
<i>hashTable</i>	A hashtable which contain parameters

Returns

The result of the call operation on concerned value.

virtual FunctionPtr stibbons : :Interpreter : :getFunctionFromTree (InterpreterManager & *manager*, const TreePtr) [virtual]

Create a function from a tree with a FCT or AGT node.

Parameters

<i>tree</i>	A tree with the FCT or AGT root node.
-------------	---------------------------------------

Returns

The function corresponding to the tree

virtual TablePtr stibbons : :Interpreter : :getParams (InterpreterManager & *manager*, FunctionPtr , AgentPtr , const TreePtr , TablePtr , std : :string) [virtual]

Interpret a function (or function of a breed)

Parameters

<i>fct</i>	A function that will be interpreted
<i>turtle</i>	The turtle to run the program on
<i>tree</i>	The Tree corresponding to function call
<i>hashTable</i>	The hashtable that will contains parameters of the function
<i>id</i>	The name of function for error message

Returns

The Value returned by the Function fct

static yy : :position stibbons : :Interpreter : :getPosition (const TreePtr) [static]

Get the position of errors

Parameters

<i>tree</i>	The syntactic tree to interpret.
-------------	----------------------------------

Returns

The position of the error detected.

```
virtual ValuePtr stibbons : :Interpreter : :interpret ( InterpreterManager & manager, AgentPtr agent, const TreePtr , TablePtr hashTable = nullptr )  
[virtual]
```

Interpret the tree and apply it to the turtle.

Parameters

<i>agent</i>	The agent to run the program on.
<i>tree</i>	The syntactic tree to interpret.
<i>hashTable</i>	A hashtable which contain parameters

Returns

An int equal to 0 if no error has occurred.

Reimplemented in stibbons : :WorldInterpreter, and stibbons : :TurtleInterpreter.

```
virtual ValuePtr stibbons : :Interpreter : :interpretFunction ( FunctionPtr ,  
AgentPtr , TablePtr ) [virtual]
```

Interpret a function (or function of a breed)

Parameters

<i>fct</i>	A function that will be interpreted
<i>turtle</i>	The turtle to run the program on
<i>tree</i>	The Tree corresponding to function call
<i>hashTable</i>	The hashtable that will contains parameters of the function
<i>id</i>	The name of function for error message

Returns

The Value returned by the Function fct

```
TablePtr stibbons : :Interpreter : :newOp ( InterpreterManager & manager,  
AgentPtr , TreePtr , TablePtr ) [inline]
```

Do the new operation if token detected is 'NEW'

Parameters

<i>agent</i>	The agent to run the program on.
<i>tree</i>	The syntactic tree to interpret.
<i>hashTable</i>	A hashtable which contain parameters

Returns

the newest turtles created by the new operation

G.17.4 Member Data Documentation

bool stibbons : :Interpreter : :inPauseFlag

A flag which signal if the thread is paused. Useful when export is called and every thread need to be in pause before exporting model.

mutex stibbons : :Interpreter : :suspendMutex

A mutex used to lock the thread when pause is called.

The documentation for this class was generated from the following file :

— src/interpreter/interpreter.h

G.18 stibbons : :InterpreterException Class Reference

Abstraction of interpreter exceptions.

#include <interpreter-exception.h>

Inheritance diagram for stibbons : :InterpreterException :

Public Member Functions

- InterpreterException (const char *, yy : :position)
- InterpreterException (std : :string msg, yy : :position)
- virtual const char * what () const =0 throw ()

G.18.1 Detailed Description

Abstraction of interpreter exceptions.

This kind of exception contains a error message and a position.

Author

Florian Galinier

G.18.2 Constructor & Destructor Documentation

**stibbons : :InterpreterException : :InterpreterException (const char * ,
yy : :position)**

Create a new InterpreterException

Parameters

<i>msg</i>	The error message.
<i>pos</i>	The position of error.

**stibbons : :InterpreterException : :InterpreterException (std : :string msg,
yy : :position)**

Create a new InterpreterException

Parameters

<i>msg</i>	The error message.
------------	--------------------

<i>pos</i>	The position of error.
------------	------------------------

G.18.3 Member Function Documentation

virtual const char* stibbons : :InterpreterException : :what () const throw)
[pure virtual]

Return a message with the error that occurred.

Returns

A message with the error and the position

Implemented in stibbons : :SemanticException, and stibbons : :SyntaxException.

The documentation for this class was generated from the following file :

— src/interpreter/interpreter-exception.h

G.19 stibbons : :InterpreterManager Class Reference

Class that will interpret stibbons language.

#include <interpreter-manager.h>

Public Member Functions

— InterpreterManager (std : :string program) throw (InterpreterException)
— virtual WorldPtr getWorld ()
— void **onErrors** (std : :function< void(string, string)> callback)
— virtual void run ()
— virtual void **wait** ()
— virtual void **checkExit** () throw (exit_requested_exception)
— virtual void setWaitTime (size_t waitTime)
— virtual void **checkHalt** ()
— virtual void halt ()
— virtual void unhalt ()
— template<class I >
void interpret_async (FunctionPtr function, AgentPtr agent, TablePtr params)

Static Public Member Functions

— static yy : :position **getPosition** (const TreePtr tree)

Static Public Attributes

— static bool suspendFlag
— static condition_variable resumeCond

G.19.1 Detailed Description

Class that will interpret stibbons language.

This class will parse the standart input, generate a syntactic tree and interpret it.

Author

Florian Galinier

Adrien Plazas

G.19.2 Constructor & Destructor Documentation

**stibbons : :InterpreterManager : :InterpreterManager (std : :string *program*)
throw InterpreterException)**

Create an interpreter manager

It parse a Stibbons program, store its syntactic tree and return the Wolrd generated by this tree.

Parameters

<i>program</i>	The Stibbons program to parse.
----------------	--------------------------------

G.19.3 Member Function Documentation

virtual WorldPtr stibbons : :InterpreterManager : :getWorld () [virtual]

Get the world.

Returns

The world associated to given program.

virtual void stibbons : :InterpreterManager : :halt () [virtual]

Halt the execution of the interpreter.

template<class I > void stibbons : :InterpreterManager : :interpret_async (FunctionPtr *function*, AgentPtr *agent*, TablePtr *params*) [inline]

Run the interpreter with the parsed program.

virtual void stibbons : :InterpreterManager : :run () [virtual]

Run the interpreter with the parsed program.

virtual void stibbons : :InterpreterManager : :setWaitTime (size_t *waitTime*) [virtual]

Set the wait time used to slow down the interpretations.

Parameters

<i>waitTime</i>	the waited time
-----------------	-----------------

virtual void stibbons : :InterpreterManager : :unhalt () [virtual]

Un-halt the execution of the interpreter.

G.19.4 Member Data Documentation

condition_variable stibbons : :InterpreterManager : :resumeCond [static]

A condition variable to control pause with all interpreters threads

bool stibbons : :InterpreterManager : :suspendFlag [static]

A boolean to signal when pause button is pressed

The documentation for this class was generated from the following file :

— src/interpreter/interpreter-manager.h

G.20 stibbons : :Line Class Reference

A colored polyline.
`#include <line.h>`

Public Member Functions

- `Line (Point point)`
- `Line (Line &other, size_t since=0)`
- `Line (Line &&other)`
- `Line & operator= (Line &other)`
- `Line & operator= (Line &&other)`
- `void setColor (Color color)`
- `Color & getColor ()`
- `const Color & getColor () const`
- `void push_back (Point point)`
- `size_t size ()`
- `void getBox (Point &begin, Point &end)`
- `void for_each (std : :function< void(Point)> foreachFunc)`

G.20.1 Detailed Description

A colored polyline.

Author

Adrien Plazas

G.20.2 Constructor & Destructor Documentation

stibbons : :Line : :Line (Point *point*)

Create a line

Parameters

<i>point</i>	the initial point
--------------	-------------------

stibbons : :Line : :Line (Line & *other*, size_t *since* = 0)

Create a copy of a line

Parameters

<i>other</i>	the other line
<i>since</i>	the point to copy the line from

stibbons : :Line : :Line (Line && *other*)

Move a line

Parameters

<i>other</i>	the other line
--------------	----------------

G.20.3 Member Function Documentation

void stibbons : :Line : :for_each (std : :function< void(Point)> *foreachFunc*)

Do something for a copy of each point

Parameters

<i>foreachFunc</i>	the fonction to call for each point
--------------------	-------------------------------------

void stibbons : :Line : :getBox (Point & *begin*, Point & *end*)

Get the points of the box containing the line

Parameters

<i>begin</i>	the first corner of the box
<i>end</i>	the last corner of the box

Color& stibbons : :Line : :getColor ()

Get the value for an axis

Returns

a reference to the value for an axis

const Color& stibbons : :Line : :getColor () const

Get the value for an axis

Returns

a constant reference to the value for an axis

Line& stibbons : :Line : :operator= (Line & *other*)

Copy of a line

Parameters

<i>other</i>	the other line
--------------	----------------

Line& stibbons : :Line : :operator= (Line && *other*)

Move a line

Parameters

<i>other</i>	the other line
--------------	----------------

void stibbons : :Line : :push_back (Point *point*)

Add a point

Parameters

<i>point</i>	the point to add
--------------	------------------

void stibbons : :Line : :setColor (Color *color*)

Set the value for an axis

Parameters

<i>axis</i>	the axis
<i>value</i>	the value

size_t stibbons : :Line : :size ()

Get the number of points

Returns

the number of points in the line

The documentation for this class was generated from the following file :

— src/model/line.h

G.21 LineNumberArea Class Reference

Widget that contain the line number area.

#include <line-number-area.h>

Inheritance diagram for LineNumberArea :

Public Member Functions

- LineNumberArea (StibbonsEditor *editor)
- QSize **sizeHint** () const Q_DECL_OVERRIDE

Protected Member Functions

- void **paintEvent** (QPaintEvent *event) Q_DECL_OVERRIDE

G.21.1 Detailed Description

Widget that contain the line number area.

Author

Florian Galinier

G.21.2 Constructor & Destructor Documentation

LineNumberArea : :LineNumberArea (StibbonsEditor * *editor*) [inline]

Constructor

Parameters

<i>editor</i>	the stibbons editor
---------------	---------------------

The documentation for this class was generated from the following file :

— src/qt/line-number-area.h

G.22 stibbons : :Nil Class Reference

A class to represent the null value.

#include <nil.h>

Inheritance diagram for stibbons : :Nil :

Public Member Functions

— virtual std::string toString ()

Additional Inherited Members

G.22.1 Detailed Description

A class to represent the null value.
class Nil

Author

Julia Bassoumi, Adrien Plazas

G.22.2 Member Function Documentation

virtual std::string stibbons::Nil::toString () [virtual]

Return a string corresponding to the value

Returns

a string corresponding to the value

Implements stibbons::Value.

The documentation for this class was generated from the following file :

— src/model/nil.h

G.23 stibbons::Number Class Reference

A class representing a real number.

#include <number.h>

Inheritance diagram for stibbons::Number :

Public Member Functions

— virtual void reset ()
— virtual ValuePtr add (ValuePtr other) throw (std::domain_error)
— virtual NumberPtr add (NumberPtr other)
— virtual ValuePtr subtract (ValuePtr other) throw (std::domain_error)
— virtual NumberPtr subtract (NumberPtr other)
— virtual ValuePtr multiply (ValuePtr other) throw (std::domain_error)
— virtual NumberPtr multiply (NumberPtr other)
— virtual ValuePtr divide (ValuePtr other) throw (std::domain_error)
— virtual NumberPtr divide (NumberPtr other) throw (std::domain_error)
— virtual ValuePtr modulo (ValuePtr other) throw (std::domain_error)
— virtual NumberPtr modulo (NumberPtr other) throw (std::domain_error)
— virtual NumberPtr unaryMinus () throw (std::domain_error)
— virtual double compare (ValuePtr other)
— virtual double compare (NumberPtr other)
— virtual string toString ()

Additional Inherited Members

G.23.1 Detailed Description

A class representing a real number.

class Number

Author

Julia Bassoumi, Adrien Plazas

G.23.2 Member Function Documentation

**virtual ValuePtr stibbons : :Number : :add (ValuePtr *other*) throw
std : :domain_error) [virtual]**

Add a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

Reimplemented from stibbons : :Value.

virtual NumberPtr stibbons : :Number : :add (NumberPtr *other*) [virtual]

Add a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

virtual double stibbons : :Number : :compare (ValuePtr *other*) [virtual]

Return whether the comparison value of two values

Return a negative number if this value is lower than the other, return a negative number if this value is greater than the other, return 0 if they are equal.

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the comparison value of this and the other value

Reimplemented from stibbons : :Value.

virtual double stibbons : :Number : :compare (NumberPtr *other*) [virtual]

Return whether the comparison value of two values

Return a negative number if this value is lower than the other, return a positive number if this value is greater than the other, return 0 if they are equal.

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the comparison value of this and the other value

**virtual ValuePtr stibbons : :Number : :divide (ValuePtr *other*) throw
std : :domain_error) [virtual]**

Divide a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

Reimplemented from `stibbons : :Value`.

**virtual NumberPtr stibbons : :Number : :divide (NumberPtr *other*) throw
std : :domain_error) [virtual]**

Divide a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

**virtual ValuePtr stibbons : :Number : :modulo (ValuePtr *other*) throw
std : :domain_error) [virtual]**

Get the remainder of the division of a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

Reimplemented from `stibbons : :Value`.

**virtual NumberPtr stibbons : :Number : :modulo (NumberPtr *other*) throw
std : :domain_error) [virtual]**

Get the remainder of the division of a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

**virtual ValuePtr stibbons : :Number : :multiply (ValuePtr *other*) throw
std : :domain_error) [virtual]**

Multiply a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

Reimplemented from `stibbons : :Value`.

```
virtual NumberPtr stibbons : :Number : :multiply ( NumberPtr other )  
[virtual]
```

Multiply a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

virtual void stibbons : :Number : :reset () [virtual]

Reset to the default value

Implements stibbons : :SimpleValue< double >.

**virtual ValuePtr stibbons : :Number : :substract (ValuePtr *other*) throw
std : :domain_error) [virtual]**

Subtract a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

Reimplemented from stibbons : :Value.

**virtual NumberPtr stibbons : :Number : :substract (NumberPtr *other*)
[virtual]**

Subtract a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

virtual string stibbons : :Number : :toString () [virtual]

Return a string corresponding to the value

Returns

a string corresponding to the value

Implements stibbons : :Value.

**virtual NumberPtr stibbons : :Number : :unaryMinus () throw
std : :domain_error) [virtual]**

Apply the unary minus operation to current number

Returns

the resulting value

The documentation for this class was generated from the following file :

— src/model/number.h

G.24 stibbons : :Parser Class Reference

Class for syntactic analysis.

`#include <parser.h>`

Public Member Functions

- Parser (stibbons : :TreePtr tree, stibbons : :TablePtr worldDir, std : :istream *in)
- int parse ()

G.24.1 Detailed Description

Class for syntactic analysis.

This class contain yy : :parser, initialized with a FlexScanner. The parse() method is used to do syntax parsing.

Author

Florian Galinier

G.24.2 Constructor & Destructor Documentation

stibbons : :Parser : :Parser (stibbons : :TreePtr *tree*, stibbons : :TablePtr *worldDir*, std : :istream * *in*) [inline]

Parser constructor.

Parameters

out	<i>tree</i>	The abstract tree generated by our parser
out	<i>worldDir</i>	a Table that contains world specifications
	<i>in</i>	The stream that will be parsed.

G.24.3 Member Function Documentation

int stibbons : :Parser : :parse () [inline]

Method used to generate abstract tree from input stream.

Returns

0 if no error occurred, 1 if parsing failed, 2 if parsing failed due to memory exhaustion (see Bison manual).

The documentation for this class was generated from the following file :

- src/interpreter/parser.h

G.25 stibbons : :Point Class Reference

A point with a parametrable number of dimensions.

`#include <point.h>`

Public Member Functions

- Point (unsigned dimensions=2)
- Point (Point &other)
- Point (Point &&other)
- Point & operator= (Point &other)
- Point & operator= (Point &&other)
- virtual ~Point ()

- void setValue (unsigned axis, double value) throw (out_of_range)
- double getValue (unsigned axis) throw (out_of_range)
- unsigned getDimensions () const
- Point getClosestImage (Point &other, Size &environment, vector< BorderType > borderTypes)
- double getDistanceTo (Point &other)
- double getAngleTo (Point &other)
- bool warp (Size &environment, vector< BorderType > borderTypes)
- double operator[] (unsigned axis) throw (out_of_range)

G.25.1 Detailed Description

A point with a parametrable number of dimensions.

Author

Adrien Plazas

G.25.2 Constructor & Destructor Documentation

stibbons : :Point : :Point (unsigned *dimensions* = 2)

Create a point of a given dimension

Parameters

<i>dimensions</i>	the number of dimensions of the point
-------------------	---------------------------------------

stibbons : :Point : :Point (Point & *other*)

Create a copy of a point

Parameters

<i>other</i>	the other point
--------------	-----------------

stibbons : :Point : :Point (Point && *other*)

Move a point

Parameters

<i>other</i>	the other point
--------------	-----------------

virtual stibbons : :Point : :~Point () [virtual]

Destroy a point

G.25.3 Member Function Documentation

double stibbons : :Point : :getAngleTo (Point & *other*)

Get the angle with the horizontal axis to another point in radian

Parameters

<i>other</i>	the other point
--------------	-----------------

Returns

the angle with the horizontal axis to another point

**Point stibbons : :Point : :getClosestImage (Point & *other*, Size & *environment*,
vector< BorderType > *borderTypes*)**

Get the closest image of another point in a warped environment

Parameters

<i>other</i>	the other point
<i>environment</i>	the size of the environment
<i>borderTypes</i>	whether the environment warps

Returns

the closest image of the other point

unsigned stibbons : :Point : :getDimensions () const

Get the number of dimensions

Returns

the number of dimensions

double stibbons : :Point : :getDistanceTo (Point & *other*)

Get the distance to another point

Parameters

<i>other</i>	the other point
--------------	-----------------

Returns

the distance to the other point

double stibbons : :Point : :getValue (unsigned *axis*) throw out_of_range)

Get the value for an axis

Returns

the value for an axis

Point& stibbons : :Point : :operator= (Point & *other*)

Copy of a point

Parameters

<i>other</i>	the other point
--------------	-----------------

Point& stibbons : :Point : :operator= (Point && *other*)

Move a point

Parameters

<i>other</i>	the other point
--------------	-----------------

double stibbons : :Point : :operator[] (unsigned *axis*) throw out_of_range)

Get the value for an axis

Returns

the value for an axis

```
void stibbons : :Point : :setValue ( unsigned axis, double value ) throw  
out_of_range)
```

Set the value for an axis

Parameters

<i>axis</i>	the axis
<i>value</i>	the value

bool stibbons : :Point : :warp (Size & *environment*, vector< BorderType > *borderTypes*)

Warp the point in the environment

Parameters

<i>environment</i>	the size of the environment
<i>borderTypes</i>	whether the environment warps

Returns

whether the point have been warped or not

The documentation for this class was generated from the following file :

— src/model/point.h

G.26 stibbons : :PrintFunction Class Reference

A class printing on the standard output.

#include <standard-function.h>

Inheritance diagram for stibbons : :PrintFunction :

Public Member Functions

— virtual ValuePtr exec (AgentPtr agent, TablePtr params)

Additional Inherited Members

G.26.1 Detailed Description

A class printing on the standard output.

class PrintFunction

Author

Adrien Plazas

G.26.2 Member Function Documentation

virtual ValuePtr stibbons : :PrintFunction : :exec (AgentPtr *agent*, TablePtr *params*) [virtual]

Execute the function

Parameters

<i>an</i>	Agent pointer to execute on
<i>a</i>	Table pointer with the parameters

Implements stibbons : :Function.

The documentation for this class was generated from the following file :

— src/model/standard-function.h

G.27 stibbons : :PrintlnFunction Class Reference

A class printing a new line on the standard output.

```
#include <standard-function.h>
```

Inheritance diagram for stibbons : :PrintlnFunction :

Public Member Functions

— virtual ValuePtr exec (AgentPtr agent, TablePtr params)

Additional Inherited Members

G.27.1 Detailed Description

A class printing a new line on the standard output.

```
class PrintlnFunction
```

Author

Adrien Plazas

G.27.2 Member Function Documentation

virtual ValuePtr stibbons : :PrintlnFunction : :exec (AgentPtr *agent*, TablePtr *params*) [virtual]

Execute the function

Parameters

<i>an</i>	Agent pointer to execute on
<i>a</i>	Table pointer with the parameters

Implements stibbons : :Function.

The documentation for this class was generated from the following file :

— src/model/standard-function.h

G.28 stibbons : :RandFunction Class Reference

A class returning a random number.

```
#include <standard-function.h>
```

Inheritance diagram for stibbons : :RandFunction :

Public Member Functions

— virtual ValuePtr exec (AgentPtr agent, TablePtr params)

Additional Inherited Members

G.28.1 Detailed Description

A class returning a random number.

```
class RandFunction
```

Author

Adrien Plazas

G.28.2 Member Function Documentation

```
virtual ValuePtr stibbons : :RandFunction : :exec ( AgentPtr agent, TablePtr  
params ) [virtual]
```

Execute the function

Parameters

<i>an</i>	Agent pointer to execute on
<i>a</i>	Table pointer with the parameters

Implements `stibbons : :Function`.

The documentation for this class was generated from the following file :

— `src/model/standard-function.h`

G.29 `stibbons : :RandomFunction` Class Reference

A class returning a random number in a range.

`#include <standard-function.h>`

Inheritance diagram for `stibbons : :RandomFunction` :

Public Member Functions

— `virtual ValuePtr exec (AgentPtr agent, TablePtr params)`

Additional Inherited Members

G.29.1 Detailed Description

A class returning a random number in a range.

`class RandomFunction`

Author

Florian Galinier

G.29.2 Member Function Documentation

`virtual ValuePtr stibbons : :RandomFunction : :exec (AgentPtr agent, TablePtr params) [virtual]`

Execute the function

Parameters

<i>an</i>	Agent pointer to execute on
<i>a</i>	Table pointer with the parameters

Implements `stibbons : :Function`.

The documentation for this class was generated from the following file :

— `src/model/standard-function.h`

G.30 `stibbons : :Runner` Class Reference

Bridge between the interpreter and Qt application.

`#include <runner.h>`

Inheritance diagram for `stibbons : :Runner` :

Public Slots

— `void setWaitTime (size_t waitTime)`

Signals

— `void error (QString type, QString what)`

Public Member Functions

- Runner (std : :string &program)
- ~Runner ()
- WorldPtr getWorld ()
- void start ()
- void run ()
- void halt ()
- std : :string exportModel ()
- bool isRunning ()

G.30.1 Detailed Description

Bridge between the interpreter and Qt application.

Author

Adrien Plazas

G.30.2 Constructor & Destructor Documentation

stibbons : :Runner : :Runner (std : :string & *program*)

Constructor

Parameters

<i>program</i>	the code to interpret
----------------	-----------------------

stibbons : :Runner : :~Runner ()

Destructor

G.30.3 Member Function Documentation

void stibbons : :Runner : :error (QString *type*, QString *what*) [signal]

Signal when an error occurred

Parameters

<i>type</i>	the error type
<i>what</i>	the error text

std : :string stibbons : :Runner : :exportModel ()

Export the model

Returns

a json string

WorldPtr stibbons : :Runner : :getWorld ()

Get the current world

Returns

the current world

void stibbons : :Runner : :halt ()

Halt the program

bool stibbons : :Runner : :isRunning ()

Return true if the program is running

Returns

a boolean corresponding to the program's state

void stibbons : :Runner : :run ()

Run the program

void stibbons : :Runner : :setWaitTime (size_t *waitTime*) [slot]

Set the wait time used to slow down the interpretations.

Parameters

<i>waitTime</i>	the waited time
-----------------	-----------------

void stibbons : :Runner : :start ()

Start the QThread

The documentation for this class was generated from the following file :

— src/qt/runner.h

G.31 stibbons : :SemanticException Class Reference

Exception thrown when a semantic error occurred.

#include <semantic-exception.h>

Inheritance diagram for stibbons : :SemanticException :

Public Member Functions

- SemanticException (const char *, yy : :position)
- SemanticException (std : :string msg, Type expectedType, Type actualType, yy : :position pos)
- SemanticException (std : :string msg, Type expectedType1, Type expectedType2, Type actualType1, Type actualType2, yy : :position pos)
- SemanticException (std : :string msg, Type expectedType1, Type expectedType2, Type actualType, yy : :position pos)
- virtual const char * what () const throw ()

G.31.1 Detailed Description

Exception thrown when a semantic error occurred.

This exception is used to inform that a semantic error occurred.

Author

Florian Galinier

G.31.2 Constructor & Destructor Documentation

stibbons : :SemanticException : :SemanticException (const char * , yy : :position)

Create a new SemanticException

Parameters

<i>msg</i>	The error message.
<i>pos</i>	The position of error.

stibbons : :SemanticException : :SemanticException (std : :string *msg*, Type *expectedType*, Type *actualType*, yy : :position *pos*)

Create a new SemanticException

Parameters

<i>msg</i>	The error message.
<i>expectedType</i>	The expected type.
<i>actualType</i>	The actual type.
<i>pos</i>	The position of error.

stibbons : :SemanticException : :SemanticException (std : :string *msg*, Type *expectedType1*, Type *expectedType2*, Type *actualType1*, Type *actualType2*, yy : :position *pos*)

Create a new SemanticException

Parameters

<i>msg</i>	The error message.
<i>expectedType1</i>	The first expected type.
<i>expectedType2</i>	The second expected type.
<i>actualType1</i>	The first actual type.
<i>actualType2</i>	The second actual type.
<i>pos</i>	The position of error.

stibbons : :SemanticException : :SemanticException (std : :string *msg*, Type *expectedType1*, Type *expectedType2*, Type *actualType*, yy : :position *pos*)

Create a new SemanticException

Parameters

<i>msg</i>	The error message.
<i>expectedType1</i>	One of the expected type.
<i>expectedType2</i>	The other expected type.
<i>actualType</i>	The actual type.
<i>pos</i>	The position of error.

G.31.3 Member Function Documentation

virtual const char* stibbons : :SemanticException : :what () const throw)
[virtual]

Return a message with the error that occurred.

Returns

A message with the error and the position

Implements `stibbons : :InterpreterException`.

The documentation for this class was generated from the following file :

— `src/interpreter/semantic-exception.h`

G.32 `stibbons : :SendFunction` Class Reference

A class sending a message from a turtle to another.

`#include <standard-function.h>`

Inheritance diagram for `stibbons : :SendFunction` :

Public Member Functions

— `virtual ValuePtr exec (AgentPtr agent, TablePtr params)`

Additional Inherited Members

G.32.1 Detailed Description

A class sending a message from a turtle to another.

`class SendFunction`

Author

Adrien Plazas

G.32.2 Member Function Documentation

`virtual ValuePtr stibbons : :SendFunction : :exec (AgentPtr agent, TablePtr params) [virtual]`

Execute the function

Parameters

<i>an</i>	Agent pointer to execute on
<i>a</i>	Table pointer with the parameters

Implements `stibbons : :Function`.

The documentation for this class was generated from the following file :

— `src/model/standard-function.h`

G.33 `stibbons : :SimpleValue< T >` Class Template Reference

A class representing a simple value.

`#include <simple-value.h>`

Inheritance diagram for `stibbons : :SimpleValue< T > :`

Public Member Functions

— `T getValue ()`
— `SimpleValue (T value)`
— `SimpleValue (SimpleValue &&other)`
— `SimpleValue (SimpleValue &other)`
— `SimpleValue & operator= (SimpleValue &&other)`
— `SimpleValue & operator= (SimpleValue &other)`

Protected Member Functions

— virtual void reset ()=0

Protected Attributes

— T **value**
— recursive__mutex **value__m**

G.33.1 Detailed Description

```
template<typename T>class stibbons : :SimpleValue< T >
```

A class representing a simple value.

Author

Adrien Plazas

G.33.2 Constructor & Destructor Documentation

```
template<typename T> stibbons : :SimpleValue< T > : :SimpleValue ( T value )  
[inline]
```

Copy constructor.

Parameters

<i>value</i>	the value to copy
--------------	-------------------

```
template<typename T> stibbons : :SimpleValue< T > : :SimpleValue ( SimpleValue< T > && other ) [inline]
```

Move initialization.

Parameters

<i>other</i>	the value to move
--------------	-------------------

```
template<typename T> stibbons : :SimpleValue< T > : :SimpleValue ( SimpleValue< T > & other ) [inline]
```

Copy initialization.

Parameters

<i>other</i>	the value to copy
--------------	-------------------

G.33.3 Member Function Documentation

```
template<typename T> T stibbons : :SimpleValue< T > : :getValue ( ) [inline]
```

Get the value of value

Returns

the value of value

```
template<typename T> SimpleValue& stibbons : :SimpleValue< T > : :operator=  
( SimpleValue< T > && other ) [inline]
```

Move assignment.

Parameters

<i>other</i>	the value to move
--------------	-------------------

```
template<typename T> SimpleValue& stibbons : :SimpleValue< T > : :operator=
( SimpleValue< T > & other ) [inline]
```

Copy assignment.

Parameters

<i>other</i>	the value to copy.
--------------	--------------------

```
template<typename T> virtual void stibbons : :SimpleValue< T > : :reset (    )
[protected], [pure virtual]
```

Reset to the default value

Implemented in stibbons : :String, stibbons : :Boolean, stibbons : :Number, and stibbons : :-
TypeValue.

The documentation for this class was generated from the following file :

— src/model/simple-value.h

G.34 stibbons : :Singleton< T > Class Template Reference

A helper class to implementing a singleton.

```
#include <singleton.h>
```

Inheritance diagram for stibbons : :Singleton< T > :

Public Member Functions

- **Singleton** (const Singleton< T > &)=delete
- Singleton & **operator=** (const Singleton< T > &)=delete

Static Public Member Functions

- static shared_ptr< T > getInstance ()

G.34.1 Detailed Description

```
template<class T>class stibbons : :Singleton< T >
```

A helper class to implementing a singleton.

```
class Singleton
```

Author

Adrien Plazas

G.34.2 Member Function Documentation

```
template<class T> static shared_ptr<T> stibbons : :Singleton< T > : :getInstance
(    ) [inline], [static]
```

Get the instance of the type T

Returns

a reference of the instance

The documentation for this class was generated from the following file :

— src/model/singleton.h

G.35 stibbons : :Size Class Reference

A size with a parametrable number of dimensions.

```
#include <size.h>
```

Public Member Functions

- Size (unsigned dimensions=2)
- Size (Size &other)
- Size (Size &&other)
- Size & operator= (Size &other)
- Size & operator= (Size &&other)
- virtual ~Size ()
- void setValue (unsigned axis, size_t value) throw (out_of_range)
- size_t getValue (unsigned axis) throw (out_of_range)
- unsigned getDimensions () const
- size_t operator[] (unsigned axis) throw (out_of_range)

G.35.1 Detailed Description

A size with a parametrable number of dimensions.

Author

Adrien Plazas

G.35.2 Constructor & Destructor Documentation

stibbons : :Size : :Size (unsigned *dimensions* = 2)

Create a size of a given dimension

Parameters

<i>dimensions</i>	the number of dimensions of the size
-------------------	--------------------------------------

stibbons : :Size : :Size (Size & *other*)

Create a copy of a size

Parameters

<i>other</i>	the other size
--------------	----------------

stibbons : :Size : :Size (Size && *other*)

Move a size

Parameters

<i>other</i>	the other size
--------------	----------------

virtual stibbons : :Size : :~Size () [virtual]

Destroy a size

G.35.3 Member Function Documentation

unsigned stibbons : :Size : :getDimensions () const

Get the number of dimensions

Returns

the number of dimensions

size_t stibbons : :Size : :getValue (unsigned *axis*) throw out_of_range)

Get the value for an axis

Returns

the value for an axis

Size& stibbons : :Size : :operator= (Size & *other*)

Copy of a size

Parameters

<i>other</i>	the other size
--------------	----------------

Size& stibbons : :Size : :operator= (Size && *other*)

Move a size

Parameters

<i>other</i>	the other size
--------------	----------------

size_t stibbons : :Size : :operator[] (unsigned *axis*) throw out_of_range)

Get the value for an axis

Returns

the value for an axis

void stibbons : :Size : :setValue (unsigned *axis*, size_t *value*) throw out_of_range)

Set the value for an axis

Parameters

<i>axis</i>	the axis
<i>value</i>	the value

The documentation for this class was generated from the following file :

— src/model/size.h

G.36 stibbons : :SizeFunction Class Reference

A class that return an array size.

```
#include <standard-function.h>
```

Inheritance diagram for stibbons : :SizeFunction :

Public Member Functions

— virtual ValuePtr exec (AgentPtr agent, TablePtr params)

Additional Inherited Members

G.36.1 Detailed Description

A class that return an array size.

```
class SizeFunction
```

Author

Florian Galinier

G.36.2 Member Function Documentation

virtual ValuePtr stibbons : :SizeFunction : :exec (AgentPtr *agent*, TablePtr *params*) [virtual]

Execute the function

Parameters

<i>an</i>	Agent pointer to execute on
<i>a</i>	Table pointer with the parameters

Implements stibbons : :Function.

The documentation for this class was generated from the following file :

— src/model/standard-function.h

G.37 StibbonsEditor Class Reference

The editor class in Qt application.

```
#include <stibbons-editor.h>
```

Inheritance diagram for StibbonsEditor :

Public Member Functions

— StibbonsEditor (QWidget *parent=0)
— void lineNumberAreaPaintEvent (QPaintEvent *event)
— int areaWidth () const

Protected Member Functions

— void **resizeEvent** (QResizeEvent *event) Q_DECL_OVERRIDE

G.37.1 Detailed Description

The editor class in Qt application.

Author

Florian Galiner

G.37.2 Constructor & Destructor Documentation

StibbonsEditor : :StibbonsEditor (QWidget * *parent* = 0)

Constructor

Parameters

<i>parent</i>	the widget parent
---------------	-------------------

G.37.3 Member Function Documentation

int StibbonsEditor : :areaWidth () const

Return the line number area width

Returns

the line number area width

void StibbonsEditor : :lineNumberAreaPaintEvent (QPaintEvent * *event*)

Repaint the lines number

Parameters

<i>event</i>	the event that called the method
--------------	----------------------------------

The documentation for this class was generated from the following file :

— src/qt/stibbons-editor.h

G.38 StibbonsHighlighter Class Reference

The highlighter class that highlight code in Qt application.

#include <stibbons-highlighter.h>

Inheritance diagram for StibbonsHighlighter :

Public Member Functions

- StibbonsHighlighter (QTextDocument *document)
- virtual void highlightBlock (const QString &line)

G.38.1 Detailed Description

The highlighter class that highlight code in Qt application.

Author

Florian Galiner

G.38.2 Constructor & Destructor Documentation

StibbonsHighlighter : :StibbonsHighlighter (QTextDocument * *document*)

Constructor

Parameters

<i>document</i>	the QTextDocument which contains the code
-----------------	---

G.38.3 Member Function Documentation

virtual void StibbonsHighlighter : :highlightBlock (const QString & *line*)
[virtual]

Method called on each document's line that will highlight special language structure.

Parameters

<i>line</i>	the line to highlight
-------------	-----------------------

The documentation for this class was generated from the following file :

— src/qt/stibbons-highlighter.h

G.39 stibbons : :String Class Reference

A class representing a string.

#include <string.h>

Inheritance diagram for stibbons : :String :

Public Member Functions

- virtual void reset ()
- virtual ValuePtr add (ValuePtr other) throw (std : :domain_error)
- virtual StringPtr add (StringPtr other)
- virtual double compare (ValuePtr other)
- virtual double compare (StringPtr other)
- virtual string toString ()

Additional Inherited Members

G.39.1 Detailed Description

A class representing a string.

Author

Julia Bassoumi, Adrien Plazas

G.39.2 Member Function Documentation

virtual ValuePtr stibbons : :String : :add (ValuePtr *other*) throw
std : :domain_error) [virtual]

Add a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

Reimplemented from stibbons : :Value.

virtual StringPtr stibbons : :String : :add (StringPtr *other*) [virtual]

Add a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

virtual double stibbons : :String : :compare (ValuePtr *other*) [virtual]

Return whether the comparison value of two values

Return a negative number if this value is lower than the other, return a positive number if this value is greater than the other, return 0 if they are equal.

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the comparison value of this and the other value

Reimplemented from stibbons : :Value.

virtual double stibbons : :String : :compare (StringPtr *other*) [virtual]

Return whether the comparison value of two values

Return a negative number if this value is lower than the other, return a negative number if this value is greater than the other, return 0 if they are equal.

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the comparison value of this and the other value

virtual void stibbons : :String : :reset () [virtual]

Reset to the default value

Implements stibbons : :SimpleValue< std : :string >.

virtual string stibbons : :String : :toString () [virtual]

Return a string corresponding to the value

Returns

a string corresponding to the value

Implements stibbons : :Value.

The documentation for this class was generated from the following file :

— src/model/string.h

G.40 stibbons : :SyntaxException Class Reference

Exception thrown when a syntax error occurred.

#include <syntax-exception.h>

Inheritance diagram for stibbons : :SyntaxException :

Public Member Functions

- `SyntaxException (const char *, yy : :position)`
- `virtual const char * what () const throw ()`

G.40.1 Detailed Description

Exception thrown when a syntax error occurred.

This exception is used to inform that a syntax error occurred.

Author

Florian Galinier

G.40.2 Constructor & Destructor Documentation

stibbons : :SyntaxException : :SyntaxException (const char * , yy : :position)

Create a new `SyntaxException`

Parameters

<i>msg</i>	The error message.
<i>pos</i>	The position of error.

G.40.3 Member Function Documentation

virtual const char* stibbons : :SyntaxException : :what () const throw)
[virtual]

Return a message with the error that occurred.

Returns

A message with the error and the position

Implements `stibbons : :InterpreterException`.

The documentation for this class was generated from the following file :

- `src/interpreter/syntax-exception.h`

G.41 stibbons : :Table Class Reference

class `Table` in `stibbons`

`#include <table.h>`

Inheritance diagram for `stibbons : :Table` :

Public Member Functions

- `Table ()=default`
- `virtual void setValue (pair< string, ValuePtr > pair)`
- `virtual void setValue (string key, ValuePtr value)`
- `virtual ValuePtr getValue (string key)`
- `virtual bool exists (string key)`
- `virtual const unordered_map`
`< string, ValuePtr > & getNamedValues ()`
- `virtual const map< long,`
`ValuePtr > & getIndexedValues ()`
- `virtual void setValue (pair< long, ValuePtr > pair)`
- `virtual void setValue (long key, ValuePtr value)`
- `virtual void append (ValuePtr value)`
- `virtual ValuePtr getValue (long key)`

- virtual double compare (ValuePtr other)
- virtual double compare (TablePtr other)
- virtual string toString ()
- virtual int length ()

Protected Attributes

- recursive_mutex **value__m**

G.41.1 Detailed Description

class Table in stibbons

G.41.2 Constructor & Destructor Documentation

stibbons : :Table : :Table () [default]

Empty Constructor

G.41.3 Member Function Documentation

virtual void stibbons : :Table : :append (ValuePtr *value*) [virtual]

Add a value to the end

Parameters

<i>value</i>	the value of the value
--------------	------------------------

virtual double stibbons : :Table : :compare (ValuePtr *other*) [virtual]

Return whether the comparison value of two values

Return a negative number if this value is lower than the other, return a negative number if this value is greater than the other, return 0 if they are equal.

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the comparison value of this and the other value

Reimplemented from stibbons : :Value.

virtual double stibbons : :Table : :compare (TablePtr *other*) [virtual]

Return whether the comparison value of two values

Return a negative number if this value is lower than the other, return a positive number if this value is greater than the other, return 0 if they are equal.

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the comparison value of this and the other value

virtual bool stibbons : :Table : :exists (string *key*) [virtual]

Check if a key exists in the table

Parameters

<i>key</i>	the key
------------	---------

Returns

wether the key exists or not

```
virtual const map<long, ValuePtr>& stibbons : :Table : :getIndexedValues (   )  
[virtual]
```

Get all the indexed values

Returns

values contained in the table

```
virtual const unordered_map<string, ValuePtr>& stibbons : :Table : :getNamed-  
Values (   ) [virtual]
```

Get all the named values

Returns

values contained in the table

```
virtual ValuePtr stibbons : :Table : :getValue ( string key ) [virtual]
```

Get a value

Parameters

<i>key</i>	the key
------------	---------

Returns

the value

```
virtual ValuePtr stibbons : :Table : :getValue ( long key ) [virtual]
```

Get a value

Parameters

<i>key</i>	the key
------------	---------

Returns

the value

```
virtual int stibbons : :Table : :length (   ) [virtual]
```

Get the length of the table

Returns

a int representing the table's length

```
virtual void stibbons : :Table : :setValue ( pair< string, ValuePtr > pair )  
[virtual]
```

Add a value

Parameters

<i>pair</i>	the key-value pair to set
-------------	---------------------------

virtual void stibbons : :Table : :setValue (string *key*, ValuePtr *value*) [virtual]

Add a value

Parameters

<i>key</i>	the key of the value
<i>value</i>	the value of the value

virtual void stibbons : :Table : :setValue (pair< long, ValuePtr > *pair*) [virtual]

Add a value

Parameters

<i>pair</i>	the key-value pair to set
-------------	---------------------------

virtual void stibbons : :Table : :setValue (long *key*, ValuePtr *value*) [virtual]

Add a value

Parameters

<i>key</i>	the key of the value
<i>value</i>	the value of the value

virtual string stibbons : :Table : :toString () [virtual]

Get a string representing the table

Returns

a string representing the table

Implements stibbons : :Value.

The documentation for this class was generated from the following file :

— src/model/table.h

G.42 stibbons : :TeleportFunction Class Reference

A class teleporting a turtle to another location.

#include <standard-function.h>

Inheritance diagram for stibbons : :TeleportFunction :

Public Member Functions

— virtual ValuePtr exec (AgentPtr agent, TablePtr params)

Additional Inherited Members

G.42.1 Detailed Description

A class teleporting a turtle to another location.

class TeleportFunction

Author

Adrien Plazas

G.42.2 Member Function Documentation

virtual ValuePtr stibbons : :TeleportFunction : :exec (AgentPtr *agent*, TablePtr *params*) [virtual]

Execute the function

Parameters

<i>an</i>	Agent pointer to execute on
<i>a</i>	Table pointer with the parameters

Implements stibbons : :Function.

The documentation for this class was generated from the following file :

— src/model/standard-function.h

G.43 stibbons : :Tree Class Reference

Class that will represent a syntactic tree.

#include <tree.h>

Public Member Functions

- Tree (int token=0, ValuePtr val=nullptr)
- virtual ~Tree ()=default
- virtual bool isLeaf () const
- virtual std : :pair< int, ValuePtr > getNode () const
- void mergeTree (TreePtr)
- void appendChildren (TreePtr)
- virtual void addChild (int, ValuePtr)
- virtual void addChild (TreePtr)
- virtual const std : :vector< TreePtr > & getChildren () const
- virtual TreePtr getChild (size_t pos=0) const
- virtual void setPosition (std : :pair< int, int >)
- virtual std : :pair< int, int > getPosition () const
- virtual void output (std : :ostream &, std : :string dec="") const

G.43.1 Detailed Description

Class that will represent a syntactic tree.

This tree will be generated by syntactic analyser and can be interpreted.

Author

Florian Galinier

Clément Simon

G.43.2 Constructor & Destructor Documentation

stibbons : :Tree : :Tree (int *token* = 0, ValuePtr *val* = nullptr)

Create a new tree from a token_type and a value

Parameters

<i>token</i>	The kind of token that is used to create the node
<i>val</i>	The value (if needed) corresponding to the token

virtual stibbons : :Tree : :~Tree () [virtual], [default]

Destroy a tree and his children.

G.43.3 Member Function Documentation

virtual void stibbons : :Tree : :addChild (int , ValuePtr) [virtual]

Create and append a child to current tree.

Parameters

<i>token</i>	The kind of token that is used to create the child
<i>val</i>	The value (if needed) corresponding to the child's token

virtual void stibbons : :Tree : :addChild (TreePtr) [virtual]

Append a tree child to current tree.

Parameters

<i>t</i>	The tree to append
----------	--------------------

void stibbons : :Tree : :appendChildren (TreePtr)

Append the children of t to the current node.

Parameters

<i>t</i>	A Tree.
----------	---------

virtual TreePtr stibbons : :Tree : :getChild (size_t *pos* = 0) const [virtual]

Return a child identify by his position

Parameters

<i>pos</i>	the position of the needed child
------------	----------------------------------

Returns

the requested child (TreePtr type)

virtual const std : :vector<TreePtr >& stibbons : :Tree : :getChildren () const [virtual]

Return all children of current tree

Returns

a vector that contains all children of the Tree

virtual std : :pair<int,ValuePtr> stibbons : :Tree : :getNode () const [virtual]

Return current node.

Returns

A pair of <token_type,Value> corresponding to current node

virtual std : :pair<int,int> stibbons : :Tree : :getPosition () const [virtual]

Return the position of the token of current node.

Returns

a pair <line,column>

virtual bool stibbons : :Tree : :isLeaf () const [virtual]

Test if the node is a leaf.

Returns

true is current tree has no children

void stibbons : :Tree : :mergeTree (TreePtr)

If current tree node is equal to t root node, append the children of t to the current node.

Parameters

<i>t</i>	A Tree with root node equal to current Tree root node.
----------	--

virtual void stibbons : :Tree : :output (std : :ostream & , std : :string *dec* = "") const [virtual]

Print the abstract tree in the given stream.

Parameters

<i>os</i>	the ostream where the tree is printed
<i>dec</i>	the shifting symbol (used for pretty print)

virtual void stibbons : :Tree : :setPosition (std : :pair< int, int >) [virtual]

Set the position of the token in file

Parameters

<i>pos</i>	a pair <line,column>
------------	----------------------

The documentation for this class was generated from the following file :

— src/interpreter/tree.h

G.44 stibbons : :Turtle Class Reference

class Turtle stibbons

#include <turtle.h>

Inheritance diagram for stibbons : :Turtle :

Public Member Functions

- virtual Type getType () const
- virtual void setProperty (string key, ValuePtr value)
- virtual ValuePtr getProperty (string p)
- void setPosition (Point position)
- Point getPosition ()
- void setAngle (double new_var)
- double getAngle ()
- void face (Point &point)
- double getDistanceTo (Point &other)
- void setId (turtle_id new_var)
- turtle_id getId ()
- WorldPtr getWorld ()
- ZonePtr getZone ()
- void setColor (Color color)
- Color & getColor ()
- const Color & getColor () const
- void forward (double dist)
- void turnRight (double angle)
- void turnLeft (double angle)
- void penDown () throw (future_error)
- void penUp () throw (future_error)
- TurtlePtr createChild ()
- void **changed** ()
- pair< TurtlePtr, ValuePtr > getFirstMessage ()
- pair< TurtlePtr, ValuePtr > getLastMessage ()
- void send (TurtlePtr t, ValuePtr v)
- void send (vector< TurtlePtr > t, ValuePtr v)
- void sendAll (ValuePtr)
- pair< TurtlePtr, ValuePtr > recv ()
- size_t checkMessage ()
- Object exportTurtle (string name="")
- virtual string toString ()
- void die ()
- virtual void destroy ()

Static Public Member Functions

- static TurtlePtr construct (Breed *breed=nullptr)
- static TurtlePtr construct (Breed *breed, AgentPtr parent, turtle_id id=0)
- static TurtlePtr construct (TurtlePtr parent)

Protected Member Functions

- Turtle (Breed *breed)
- Turtle (Breed *breed, AgentPtr parent, turtle_id id=0)
- Turtle (TurtlePtr parent)
- void init ()

Additional Inherited Members

G.44.1 Detailed Description

class Turtle stibbons

Author

Julia Bassoumi

G.44.2 Constructor & Destructor Documentation

stibbons : :Turtle : :Turtle (Breed * *breed*) [protected]

Create a turtle with breed

Parameters

<i>breed</i>	the turtle's breed
--------------	--------------------

stibbons : :Turtle : :Turtle (Breed * *breed*, AgentPtr *parent*, turtle_id *id* = 0) [protected]

Create a turtle

Parameters

<i>breed</i>	the turtle's breed
<i>parent</i>	a turtle who create the current turtle
<i>id</i>	the id of the turtle

stibbons : :Turtle : :Turtle (TurtlePtr *parent*) [protected]

Special constructor for Breed

Parameters

<i>parent</i>	a turtle who create the current turtle
---------------	--

G.44.3 Member Function Documentation

size_t stibbons : :Turtle : :checkMessage ()

Check if you have receive some messages

Returns

the number of message available

static TurtlePtr stibbons : :Turtle : :construct (Breed * *breed* = nullptr) [static]

Create a turtle

static TurtlePtr stibbons : :Turtle : :construct (Breed * *breed*, AgentPtr *parent*, turtle_id *id* = 0) [static]

Construct a turtle

Parameters

<i>breed</i>	the turtle's breed
<i>parent</i>	a turtle who create the current turtle
<i>id</i>	the id of the turtle

static TurtlePtr stibbons : :Turtle : :construct (TurtlePtr *parent*) [static]

Special constructor for Breed

TurtlePtr stibbons : :Turtle : :createChild ()

Create a child turtle

Create a new turtle whose parent is this turtle, and whose breed and color are this turtle's breed and color.

Returns

the new child turtle

virtual void stibbons : :Turtle : :destroy () [virtual]

Remove as much references to this agent as possible
Reimplemented from stibbons : :Agent.

void stibbons : :Turtle : :die ()

Kill a turtle

Object stibbons : :Turtle : :exportTurtle (string *name* = "")

Export turtle's state

Parameters

<i>a</i>	string represent the name of his breed, which can not exist
----------	---

Returns

a json_spirit object

void stibbons : :Turtle : :face (Point & *point*)

Face a point

Parameters

<i>point</i>	the point to face
--------------	-------------------

void stibbons : :Turtle : :forward (double *dist*)

Go forward

Parameters

<i>the</i>	distance to go
------------	----------------

double stibbons : :Turtle : :getAngle ()

Get the value of angle

Returns

the value of angle

Color& stibbons : :Turtle : :getColor ()

Get the value for an axis

Returns

a reference to the value for an axis

const Color& stibbons : :Turtle : :getColor () const

Get the value for an axis

Returns

a constant reference to the value for an axis

double stibbons : :Turtle : :getDistanceTo (Point & *other*)

Get the distance to another point

Parameters

<i>other</i>	the other point
--------------	-----------------

Returns

the distance to the other point

pair<TurtlePtr,ValuePtr> stibbons : :Turtle : :getFirstMessage ()

Get the first message receive

Returns

the first Message

turtle_id stibbons : :Turtle : :getId ()

Get the value of id

Returns

the value of id

pair<TurtlePtr,ValuePtr> stibbons : :Turtle : :getLastMessage ()

Get the last message receive

Returns

the last Message

Point stibbons : :Turtle : :getPosition ()

Get the position of the turtle

Returns

the position of the turtle

virtual ValuePtr stibbons : :Turtle : :getProperty (string *p*) [virtual]

Get the value of the propertie p

Returns

the value of propertie p

Reimplemented from stibbons : :Agent.

virtual Type stibbons : :Turtle : :getType () const [virtual]

Return the type of the value

Returns

a Type

Implements stibbons : :Value.

WorldPtr stibbons : :Turtle : :getWorld () [virtual]

Get the value of world

Returns

the value of world

Implements stibbons : :Agent.

ZonePtr stibbons : :Turtle : :getZone ()

Get the hovered zone

Returns

the hovered zone

void stibbons : :Turtle : :init () [protected]

Initialize the world

void stibbons : :Turtle : :penDown () throw future_error)

Let a trace on the map behind the turtle

void stibbons : :Turtle : :penUp () throw future_error)

Stop to trace on the map behind the turtle

pair<TurtlePtr,ValuePtr> stibbons : :Turtle : :recv ()

Read the first message of the deque and delete it from the deque

Parameters

<i>the</i>	etiquette
------------	-----------

Returns

Value, the first message receive

void stibbons : :Turtle : :send (TurtlePtr *t*, ValuePtr *v*)

Send a message to the Turtle *t*

Parameters

<i>t</i>	a turtle
----------	----------

<i>v</i>	the message to send
----------	---------------------

void stibbons : :Turtle : :send (vector< TurtlePtr > *t*, ValuePtr *v*)

Send a message to all the turtle in *t*

Parameters

<i>t</i>	a vector of turtle
<i>v</i>	the message to send

void stibbons : :Turtle : :sendAll (ValuePtr)

Send a message to all turtles

Parameters

<i>the</i>	message a Value
------------	-----------------

void stibbons : :Turtle : :setAngle (double *new_var*)

Set the value of angle

Parameters

<i>new_var</i>	the new value of angle
----------------	------------------------

void stibbons : :Turtle : :setColor (Color *color*)

Set the value for an axis

Parameters

<i>axis</i>	the axis
<i>value</i>	the value

void stibbons : :Turtle : :setId (turtle_id *new_var*)

Set the value of id

Parameters

<i>new_var</i>	the new value of id
----------------	---------------------

void stibbons : :Turtle : :setPosition (Point *position*)

Set the position of the turtle

Parameters

<i>position</i>	the new position of the turtle
-----------------	--------------------------------

virtual void stibbons : :Turtle : :setProperty (string *key*, ValuePtr *value*)
[virtual]

Add a property

Parameters

<i>key</i>	the key of the property
<i>value</i>	the value of the property

Reimplemented from `stibbons : :Agent`.

virtual string stibbons : :Turtle : :toString () [virtual]

Return a string corresponding to the value

Returns

a string corresponding to the value

Implements `stibbons : :Value`.

void stibbons : :Turtle : :turnLeft (double *angle*)

Turn left at dist degree

Parameters

<i>the</i>	degree to turn
------------	----------------

void stibbons : :Turtle : :turnRight (double *angle*)

Turn right at dist degree

Parameters

<i>the</i>	degree to turn
------------	----------------

The documentation for this class was generated from the following file :

— `src/model/turtle.h`

G.45 stibbons : :TurtleInterpreter Class Reference

Inheritance diagram for `stibbons : :TurtleInterpreter` :

Public Member Functions

- `TurtleInterpreter ()=default`
- `virtual ValuePtr interpret (InterpreterManager &manager, AgentPtr agent, const TreePtr, TablePtr hashTable)`
- `virtual ValuePtr interpret (InterpreterManager &manager, TurtlePtr agent, const TreePtr, TablePtr hashTable)`

Additional Inherited Members

G.45.1 Constructor & Destructor Documentation

stibbons : :TurtleInterpreter : :TurtleInterpreter () [default]

Create a new interpreter for a turtle

G.45.2 Member Function Documentation

virtual ValuePtr stibbons : :TurtleInterpreter : :interpret (InterpreterManager & *manager*, AgentPtr *agent*, const TreePtr , TablePtr *hashTable*) [virtual]

Check if the agent is a world and call the interpret for a turtle

Parameters

<i>agent</i>	The agent to run the program on.
<i>tree</i>	The syntactic tree to interpret.
<i>hashTable</i>	A hashtable which contain parameters

Returns

An int equal to 0 if no error has occurred.

Reimplemented from stibbons : :Interpreter.

virtual ValuePtr stibbons : :TurtleInterpreter : :interpret (InterpreterManager & manager, TurtlePtr agent, const TreePtr , TablePtr hashTable) [virtual]

Interpret the tree and apply it to the turtle.

Parameters

<i>agent</i>	The agent to run the program on.
<i>tree</i>	The syntactic tree to interpret.
<i>hashTable</i>	A hashtable which contain parameters

Returns

An int equal to 0 if no error has occurred.

The documentation for this class was generated from the following file :

— src/interpreter/turtle-interpreter.h

G.46 Type Class Reference

The available value types.

#include <type.h>

G.46.1 Detailed Description

The available value types.

Author

Julia Bassoumi

The documentation for this class was generated from the following file :

— src/model/type.h

G.47 stibbons : :TypeOfFunction Class Reference

A class returning the type of a value.

#include <standard-function.h>

Inheritance diagram for stibbons : :TypeOfFunction :

Public Member Functions

— virtual ValuePtr exec (AgentPtr agent, TablePtr params)

Additional Inherited Members

G.47.1 Detailed Description

A class returning the type of a value.

class TypeOfFunction

Author

Adrien Plazas

G.47.2 Member Function Documentation

virtual ValuePtr stibbons : :TypeOfFunction : :exec (AgentPtr *agent*, TablePtr *params*) [virtual]

Execute the function

Parameters

<i>an</i>	Agent pointer to execute on
<i>a</i>	Table pointer with the parameters

Implements stibbons : :Function.

The documentation for this class was generated from the following file :

— src/model/standard-function.h

G.48 stibbons : :TypeValue Class Reference

A class representing a type.

#include <type-value.h>

Inheritance diagram for stibbons : :TypeValue :

Public Member Functions

- virtual void reset ()
- virtual double compare (ValuePtr other)
- virtual double compare (TypeValuePtr other)
- virtual string toString ()

Additional Inherited Members

G.48.1 Detailed Description

A class representing a type.

class TypeValue

Author

Adrien Plazas

G.48.2 Member Function Documentation

virtual double stibbons : :TypeValue : :compare (ValuePtr *other*) [virtual]

Return whether the comparison value of two values

Return a negative number if this value is lower than the other, return a negative number if this value is greater than the other, return 0 if they are equal.

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the comparison value of this and the other value

Reimplemented from `stibbons : :Value`.

virtual double stibbons : :TypeValue : :compare (TypeValuePtr *other*)
[virtual]

Return whether the comparison value of two values

Return a negative number if this value is lower than the other, return a positive number if this value is greater than the other, return 0 if they are equal.

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the comparison value of this and the other value

virtual void stibbons : :TypeValue : :reset () [virtual]

Reset to the default value

Implements `stibbons : :SimpleValue< Type >`.

virtual string stibbons : :TypeValue : :toString () [virtual]

Return a string corresponding to the value

Returns

a string corresponding to the value

Implements `stibbons : :Value`.

The documentation for this class was generated from the following file :

— `src/model/type-value.h`

G.49 stibbons : :UserFunction Class Reference

A class representing a user defined function.

#include <user-function.h>

Inheritance diagram for `stibbons : :UserFunction` :

Public Member Functions

- `UserFunction (InterpreterManager &manager, TreePtr tree, vector< string > params=vector< string >())`
- `UserFunction (TreePtr tree, vector< string > params=vector< string >())`
- `virtual ~UserFunction ()=default`
- `virtual ValuePtr exec (AgentPtr agent, TablePtr params)`

Additional Inherited Members

G.49.1 Detailed Description

A class representing a user defined function.

class UserFunction

Author

Adrien Plazas

G.49.2 Constructor & Destructor Documentation

stibbons : :UserFunction : :UserFunction (InterpreterManager & *manager*, TreePtr *tree*, vector< string > *params* = vector< string >())

Constructor

Parameters

<i>manager</i>	the interpreter manager
<i>tree</i>	the syntactic tree of the function
<i>params</i>	the parameters that the function expect to be executed with

stibbons : :UserFunction : :UserFunction (TreePtr *tree*, vector< string > *params* = vector< string >())

Constructor

Parameters

<i>tree</i>	the syntactic tree of the function
<i>params</i>	the parameters that the function expect to be executed with

virtual stibbons : :UserFunction : :~UserFunction () [virtual], [default]

Destructor

G.49.3 Member Function Documentation

virtual ValuePtr stibbons : :UserFunction : :exec (AgentPtr *agent*, TablePtr *params*) [virtual]

Execute the function

Parameters

<i>agent</i>	the agent to execute the function on
<i>params</i>	the parameters to execute the function with

Returns

the returned value

Implements stibbons : :Function.

The documentation for this class was generated from the following file :

— src/model/user-function.h

G.50 stibbons : :Value Class Reference

An abstract class who represent a values of a given type.

`#include <value.h>`

Inheritance diagram for stibbons : :Value :

Public Member Functions

- virtual Type getType () const =0
- virtual ValuePtr add (ValuePtr other) throw (std : :domain_error)
- virtual ValuePtr substract (ValuePtr other) throw (std : :domain_error)
- virtual ValuePtr multiply (ValuePtr other) throw (std : :domain_error)
- virtual ValuePtr divide (ValuePtr other) throw (std : :domain_error)
- virtual ValuePtr modulo (ValuePtr other) throw (std : :domain_error)
- virtual double compare (ValuePtr other)
- virtual bool isEqual (ValuePtr other)
- virtual bool isDifferent (ValuePtr other)
- virtual bool isLower (ValuePtr other)
- virtual bool isGreater (ValuePtr other)
- virtual bool isLowerOrEqual (ValuePtr other)
- virtual bool isGreaterOrEqual (ValuePtr other)
- virtual std : :string toString ()=0

G.50.1 Detailed Description

An abstract class who represent a values of a given type.

Author

Julia Bassoumi

G.50.2 Member Function Documentation

**virtual ValuePtr stibbons : :Value : :add (ValuePtr *other*) throw
std : :domain_error) [inline], [virtual]**

Add a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

Reimplemented in stibbons : :Color, stibbons : :String, and stibbons : :Number.

**virtual double stibbons : :Value : :compare (ValuePtr *other*) [inline],
[virtual]**

Return whether the comparison value of two values

Return a negative number if this value is lower than the other, return a positive number if this value is greater than the other, return 0 if they are equal.

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the comparison value of this and the other value

Reimplemented in stibbons : :Color, stibbons : :Number, stibbons : :Table, stibbons : :Agent, stibbons : :Function, stibbons : :String, stibbons : :Boolean, and stibbons : :TypeValue.


```
virtual ValuePtr stibbons : :Value : :divide ( ValuePtr other ) throw  
std : :domain_error) [inline], [virtual]
```

Divide a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

Reimplemented in `stibbons : :Color`, and `stibbons : :Number`.

virtual Type stibbons : :Value : :getType () const [pure virtual]

Return the type of the value

Returns

a Type

Implemented in `stibbons : :GenericValue< T >`, `stibbons : :GenericValue< Type : :NUMBER >`, `stibbons : :GenericValue< Type : :NIL >`, `stibbons : :GenericValue< Type : :STRING >`, `stibbons : :GenericValue< Type : :BOOLEAN >`, `stibbons : :GenericValue< Type : :TYPE >`, `stibbons : :GenericValue< Type : :FUNCTION >`, `stibbons : :GenericValue< Type : :COLOR >`, `stibbons : :GenericValue< Type : :TABLE >`, `stibbons : :Zone`, `stibbons : :Turtle`, and `stibbons : :World`.

virtual bool stibbons : :Value : :isDifferent (ValuePtr *other*) [inline], [virtual]

Return whether this value is different from the other

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

whether this value is different from the other

virtual bool stibbons : :Value : :isEqual (ValuePtr *other*) [inline], [virtual]

Return whether this value is equal to the other

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

whether this value is equal to the other

virtual bool stibbons : :Value : :isGreater (ValuePtr *other*) [inline], [virtual]

Return whether this value is greater than the other

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

whether this value is greater than the other

virtual bool stibbons : :Value : :isGreaterOrEqual (ValuePtr *other*) [inline], [virtual]

Return whether this value is greater than or equal to the other

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

whether this value is greater than or equal to the other

virtual bool stibbons : :Value : :isLower (ValuePtr *other*) [inline], [virtual]

Return whether this value is lower than the other

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

whether this value is lower than the other

virtual bool stibbons : :Value : :isLowerOrEqual (ValuePtr *other*) [inline], [virtual]

Return whether this value is lower than or equal to the other

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

whether this value is lower than or equal to the other

virtual ValuePtr stibbons : :Value : :modulo (ValuePtr *other*) throw std : :domain_error) [inline], [virtual]

Get the remainder of the division of a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

Reimplemented in stibbons : :Number.

virtual ValuePtr stibbons : :Value : :multiply (ValuePtr *other*) throw std : :domain_error) [inline], [virtual]

Multiply a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

Reimplemented in stibbons : :Color, and stibbons : :Number.

virtual ValuePtr stibbons : :Value : :subtract (ValuePtr *other*) throw std : :domain_error) [inline], [virtual]

Subtract a value to another

Parameters

<i>other</i>	the other value
--------------	-----------------

Returns

the resulting value

Reimplemented in `stibbons : :Color`, and `stibbons : :Number`.

virtual std : :string stibbons : :Value : :toString () [pure virtual]

Return a string corresponding to the value

Returns

a string corresponding to the value

Implemented in `stibbons : :Turtle`, `stibbons : :World`, `stibbons : :Color`, `stibbons : :Number`, `stibbons : :Zone`, `stibbons : :Table`, `stibbons : :Function`, `stibbons : :String`, `stibbons : :Boolean`, `stibbons : :TypeValue`, and `stibbons : :Nil`.

The documentation for this class was generated from the following file :

— `src/model/value.h`

G.51 stibbons : :Window Class Reference

The window of Qt application.

#include <window.h>

Inheritance diagram for `stibbons : :Window` :

Public Slots

- void **open** ()
- void **reset** ()
- void **run** ()
- void **rerun** ()
- void **halt** ()
- void **exportModel** ()
- void **exportModel** (QString fileName)
- void **about** ()
- void **updateInterpreterWaitTime** (int waitTime)
- void **save** ()
- void **saveUnder** ()
- void **error** (QString type, QString what)

Signals

- void **change** ()

Public Member Functions

- Window ()
- ~Window ()

Protected Member Functions

- void **closeEvent** (QCloseEvent *event) Q_DECL_OVERRIDE

G.51.1 Detailed Description

The window of Qt application.

Author

Adrien Plazas

G.51.2 Constructor & Destructor Documentation

stibbons : :Window : :Window ()

Constructor

stibbons : :Window : :~Window ()

Destructor

The documentation for this class was generated from the following file :

— src/qt/window.h

G.52 stibbons : :World Class Reference

A world consisting of zones, breeds, turtles and lines.

#include <world.h>

Inheritance diagram for stibbons : :World :

Public Member Functions

- ~World ()
- virtual Type getType () const
- virtual void setProperty (string key, ValuePtr value)
- virtual ValuePtr getProperty (string p)
- WorldPtr getWorld ()
- void addLine (Line *add_object)
- vector< Line * > getLines ()
- vector< Line > getLinesSince (vector< size_t > &sizes)
- BreedPtr getBreed (string name) throw (out_of_range)
- BreedPtr createBreed (FunctionPtr function, string name) throw (invalid_argument)
- BreedPtr createBreed (FunctionPtr function)
- unordered_set< TurtlePtr > getTurtles ()
- vector< ZonePtr > getZone ()
- ZonePtr getZone (Size &coordinates) throw (domain_error)
- ZonePtr getZone (Point &point) throw (domain_error)
- unsigned getDimensions () const
- Size getSize ()
- Size getWorldSize ()
- Size getZoneSize ()
- vector< BorderType > getBorderTypes ()
- turtle_id getTurtleId ()
- void nextTurtleId ()
- turtle_id putTurtleId ()
- zone_id getZoneId ()
- void nextZoneId ()
- zone_id putZoneId ()
- bool exporte ()
- Object exportWorld ()
- virtual string toString ()

Static Public Member Functions

- static WorldPtr construct (Size worldSize, Size zoneSize, vector< BorderType > borderTypes) throw (domain_error)

Protected Member Functions

- World (Size worldSize, Size zoneSize, vector< BorderType > borderTypes) throw (domain_error)
- void init ()

Additional Inherited Members

G.52.1 Detailed Description

A world consisting of zones, breeds, turtles and lines.

Author

Julia Bassoumi

G.52.2 Constructor & Destructor Documentation

stibbons : :World : ~World ()

Empty Destructor

stibbons : :World : :World (Size *worldSize*, Size *zoneSize*, vector< BorderType > *borderTypes*) throw domain_error [protected]

Create a world.

G.52.3 Member Function Documentation

void stibbons : :World : :addLine (Line * *add_object*)

Add a line.

Parameters

<i>add_object</i>	the line to add
-------------------	-----------------

static WorldPtr stibbons : :World : :construct (Size *worldSize*, Size *zoneSize*, vector< BorderType > *borderTypes*) throw domain_error [static]

Create a world

BreedPtr stibbons : :World : :createBreed (FunctionPtr *function*, string *name*) throw invalid_argument

Create and add a new named breed.

Parameters

<i>function</i>	the function of the breed
<i>name</i>	the name of the breed

Returns

a reference to the new breed

BreedPtr stibbons : :World : :createBreed (FunctionPtr *function*)

 Create and add a new anonymous breed.

Parameters

<i>function</i>	the function of the breed
-----------------	---------------------------

Returns

a reference to the new breed

bool stibbons : :World : :exporte ()

Export the model.

Returns

true if sucess, else false

Object stibbons : :World : :exportWorld ()

Export the world's model.

Returns

the json object with the properties

vector<BorderType> stibbons : :World : :getBorderTypes ()

Get the border types of the world.

Returns

the border types

BreedPtr stibbons : :World : :getBreed (string *name*) throw out_of_range)

Get a named breed.

Parameters

<i>name</i>	the name of the breed
-------------	-----------------------

Returns

the breed

unsigned stibbons : :World : :getDimensions () const

Get the world's number of dimensions.

Returns

the world's number of dimensions

vector<Line*> stibbons : :World : :getLines ()

Get the lines.

Returns

the lines

vector<Line> stibbons : :World : :getLinesSince (vector< size_t > & *sizes*)

Get the new lines since a certain state.

It also updates the sizes parameter to the current sizes.

Parameters

<i>sizes</i>	the previous sizes of the lines
--------------	---------------------------------

Returns

the new lines

virtual ValuePtr stibbons : :World : :getProperty (string *p*) [virtual]

Get the value of the propertie *p*.

Returns

the value of propertie *p*

Reimplemented from stibbons : :Agent.

Size stibbons : :World : :getSize ()

Get the world's full size.

Returns

the world's full size

turtle_id stibbons : :World : :getTurtleId ()

Return the global variable Tid.

Returns

Tid, type turtle_id

unordered_set<TurtlePtr> stibbons : :World : :getTurtles ()

Get the turtles.

Returns

the turtles

virtual Type stibbons : :World : :getType () const [virtual]

Return the type of the value

Returns

a Type

Implements stibbons : :Value.

WorldPtr stibbons : :World : :getWorld () [virtual]

Get the value of world.

Returns

the value of world

Implements stibbons : :Agent.

Size stibbons : :World : :getWorldSize ()

Get the world's size.

Returns

the world's size

vector<ZonePtr> stibbons : :World : :getZone ()

Get the list of all the zone in the world.

Returns

a vector of zone

ZonePtr stibbons : :World : :getZone (Size & *coordinates*) throw domain__error)

Get the zone at a given coordinates.

Returns

the zone

ZonePtr stibbons : :World : :getZone (Point & *point*) throw domain__error)

Get the zone at a given point.

Returns

the zone

zone__id stibbons : :World : :getZoneId ()

Return the global variable Zid.

Returns

Zid, type zone__id

Size stibbons : :World : :getZoneSize ()

Get the zones' size.

Returns

the zones' size

void stibbons : :World : :init () [protected]

Initialize the world.

void stibbons : :World : :nextTurtleId ()

Increment the Tid variable.

void stibbons : :World : :nextZoneId ()

Increment the Zid variable.

turtle_id stibbons : :World : :putTurtleId ()

Return the id for a turtle and increment the world's variable Tid.
Returns

Tid, type turtle_id

zone_id stibbons : :World : :putZoneId ()

Return the id for a zone and increment the world's variable Zid.
Returns

Zid, type zone_id

virtual void stibbons : :World : :setProperty (string *key*, ValuePtr *value*)
[virtual]

Add a property.
Parameters

<i>key</i>	the key of the property
<i>value</i>	the value of the property

Reimplemented from stibbons : :Agent.

virtual string stibbons : :World : :toString () [virtual]

Return a string corresponding to the value.
Returns

a string corresponding to the value

Implements stibbons : :Value.

The documentation for this class was generated from the following file :
— src/model/world.h

G.53 stibbons : :WorldInterpreter Class Reference

Class that will interpret stibbons language.

#include <world-interpreter.h>

Inheritance diagram for stibbons : :WorldInterpreter :

Public Member Functions

- WorldInterpreter ()=default
- virtual ValuePtr interpret (InterpreterManager &manager, AgentPtr agent, const TreePtr, TablePtr hashTable)
- virtual ValuePtr interpret (InterpreterManager &manager, WorldPtr agent, const TreePtr, TablePtr hashTable)

Additional Inherited Members

G.53.1 Detailed Description

Class that will interpret stibbons language.

This class will parse the standart input, generate a syntactic tree and interpret it.

Author

Florian Galinier
Clément Simon

G.53.2 Constructor & Destructor Documentation

stibbons : :WorldInterpreter : :WorldInterpreter () [default]

Create a new interpreter for a turtle

G.53.3 Member Function Documentation

virtual ValuePtr stibbons : :WorldInterpreter : :interpret (InterpreterManager & manager, AgentPtr agent, const TreePtr , TablePtr hashTable) [virtual]

Check if the agent is a world and call the interpret for a world

Parameters

<i>agent</i>	The agent to run the program on.
<i>tree</i>	The syntactic tree to interpret.
<i>hashTable</i>	A hashtable which contain parameters

Returns

An int equal to 0 if no error has occurred.

Reimplemented from stibbons : :Interpreter.

virtual ValuePtr stibbons : :WorldInterpreter : :interpret (InterpreterManager & manager, WorldPtr agent, const TreePtr , TablePtr hashTable) [virtual]

Interpret the tree and apply it to the world.

Parameters

<i>agent</i>	The agent to run the program on.
<i>tree</i>	The syntactic tree to interpret.
<i>hashTable</i>	A hashtable which contain parameters

Returns

An int equal to 0 if no error has occurred.

The documentation for this class was generated from the following file :

— src/interpreter/world-interpreter.h

G.54 stibbons : :WorldPainter Class Reference

The painter of the world in Qt application.

#include <world-painter.h>

Public Member Functions

- WorldPainter (WorldPtr world)
- void resetLinesBuffer ()
- void paint (QPainter &p)
- void paint (QPainter &p, int xt, int yt)

G.54.1 Detailed Description

The painter of the world in Qt application.

Author

Adrien Plazas

G.54.2 Constructor & Destructor Documentation

stibbons : :WorldPainter : :WorldPainter (WorldPtr *world*)

Constructor

Parameters

<i>world</i>	the world to paint
--------------	--------------------

G.54.3 Member Function Documentation

void stibbons : :WorldPainter : :paint (QPainter & *p*)

Paint an object on the world view

Parameters

<i>p</i>	object to paint
----------	-----------------

void stibbons : :WorldPainter : :paint (QPainter & *p*, int *xt*, int *yt*)

Paint an object on the world view

Parameters

<i>p</i>	object to paint
<i>xt</i>	x translation
<i>yt</i>	y translation

void stibbons : :WorldPainter : :resetLinesBuffer ()

Reset the buffer which contains lines

The documentation for this class was generated from the following file :

— src/qt/world-painter.h

G.55 stibbons : :WorldView Class Reference

The view of the world in Qt application.

#include <world-view.h>

Inheritance diagram for stibbons : :WorldView :

Signals

— void changed ()

Public Member Functions

- WorldView (QWidget *parent=nullptr)
- void setWorld (WorldPtr world)
- WorldPtr getWorld ()
- virtual QSize **sizeHint** () const

Protected Member Functions

— virtual void **paintEvent** (QPaintEvent *event)

G.55.1 Detailed Description

The view of the world in Qt application.

Author

Adrien Plazas

G.55.2 Constructor & Destructor Documentation

stibbons : :WorldView : :WorldView (QWidget * *parent* = *nullptr*)

Constructor

Parameters

<i>parent</i>	the parent widget
---------------	-------------------

G.55.3 Member Function Documentation

void stibbons : :WorldView : :changed () [signal]

Method to call when the world changed

WorldPtr stibbons : :WorldView : :getWorld ()

Get the current world

Returns

the current world

void stibbons : :WorldView : :setWorld (WorldPtr *world*)

Set the current world with the param

Parameters

<i>world</i>	the new world
--------------	---------------

The documentation for this class was generated from the following file :

— src/qt/world-view.h

G.56 stibbons : :Zone Class Reference

class Zone

#include <zone.h>

Inheritance diagram for stibbons : :Zone :

Public Member Functions

- Zone (Zone &other)
- Zone (Zone &&other)
- Zone & operator= (Zone &other)
- Zone & operator= (Zone &&other)
- virtual Type getType () const
- virtual void setProperty (string key, ValuePtr value)
- virtual ValuePtr getProperty (string p)
- void setColor (Color color)

- Color getColor ()
- WorldPtr getWorld ()
- void setId (zone_id z)
- zone_id getId ()
- void **changed** ()
- virtual string toString ()
- Object exportZone ()

Static Public Member Functions

- static ZonePtr construct (AgentPtr parent)

Protected Member Functions

- Zone (AgentPtr parent)
- void init ()

Additional Inherited Members

G.56.1 Detailed Description

class Zone

Author

Julia Bassoumi

G.56.2 Constructor & Destructor Documentation

stibbons : :Zone : :Zone (Zone & other)

Create a copy of a zone

Parameters

<i>other</i>	the other zone
--------------	----------------

stibbons : :Zone : :Zone (Zone && other)

Move a zone

Parameters

<i>other</i>	the other zone
--------------	----------------

stibbons : :Zone : :Zone (AgentPtr parent) [protected]

Create a zone

Parameters

<i>parent</i>	the parent of the current zone
---------------	--------------------------------

G.56.3 Member Function Documentation

static ZonePtr stibbons : :Zone : :construct (AgentPtr parent) [static]

Create a zone

Parameters

<i>parent</i>	the parent of the current zone
---------------	--------------------------------

Object stibbons : :Zone : :exportZone ()

Return an object which contain the zone's properties

Returns

an json Object

Color stibbons : :Zone : :getColor ()

Get the value for an axis

Returns

a reference to the value for an axis

zone_id stibbons : :Zone : :getId ()

Get the value of id

Returns

the value of id

virtual ValuePtr stibbons : :Zone : :getProperty (string *p*) [virtual]

Get the value of the propertie p

Returns

the value of propertie p

Reimplemented from stibbons : :Agent.

virtual Type stibbons : :Zone : :getType () const [virtual]

Return the type of the value

Returns

a Type

Implements stibbons : :Value.

WorldPtr stibbons : :Zone : :getWorld () [virtual]

Get the value of world

Returns

the value of world

Implements stibbons : :Agent.

void stibbons : :Zone : :init () [protected]

Initialize the zone

Zone& stibbons : :Zone : :operator= (Zone & *other*)

Copy of a zone

Parameters

<i>other</i>	the other zone
--------------	----------------

Zone& stibbons : :Zone : :operator= (Zone && *other*)

Move a zone

Parameters

<i>other</i>	the other zone
--------------	----------------

void stibbons : :Zone : :setColor (Color *color*)

Set the value for an axis

Parameters

<i>axis</i>	the axis
<i>value</i>	the value

void stibbons : :Zone : :setId (zone_id *z*)

Set the value of id

Parameters

<i>the</i>	id
------------	----

virtual void stibbons : :Zone : :setProperty (string *key*, ValuePtr *value*)
[virtual]

Add a property

Parameters

<i>key</i>	the key of the property
<i>value</i>	the value of the property

Reimplemented from stibbons : :Agent.

virtual string stibbons : :Zone : :toString () [virtual]

Return a string corresponding to the value

Returns

a string corresponding to the value

Implements stibbons : :Value.

The documentation for this class was generated from the following file :

— src/model/zone.h

Annexe H

File Documentation

H.1 src/interpreter/flex-scanner.h File Reference

FlexScanner class header.

```
#include <FlexLexer.h>
```

```
#include "y.tab.h"
```

Include dependency graph for flex-scanner.h : This graph shows which files directly or indirectly include this file :

Classes

— class stibbons : :FlexScanner
Class for lexical analysis.

Macros

— #define **YY_DECL** int stibbons : :FlexScanner : :yylex()
— #define **YY_DECL** int stibbons : :FlexScanner : :yylex()
— #define **YY_DECL** int stibbons : :FlexScanner : :yylex()

H.1.1 Detailed Description

FlexScanner class header.

Author

Florian Galinier

Version

1.1

Date

26/04/15

Definition of the FlexScanner class inheriting yyFlexLexer.

H.2 src/interpreter/interpreter-exception.h File Reference

InterpreterException class header.

```

#include <exception>
#include <string>
#include <sstream>
#include "y.tab.h"

```

Include dependency graph for interpreter-exception.h : This graph shows which files directly or indirectly include this file :

Classes

- class stibbons : :InterpreterException
Abstraction of interpreter exceptions.

H.2.1 Detailed Description

InterpreterException class header.

Author

Florian Galinier

Version

1.1

Date

28/03/15

Declaration of InterpreterException class inherited by syntax and semantic exceptions.

H.3 src/interpreter/interpreter-manager.h File Reference

InterpreterManager class header.

```

#include "interpreter.h"
#include "tree.h"
#include "syntax-exception.h"
#include "../model/world.h"
#include <mutex>
#include <thread>
#include <vector>

```

Include dependency graph for interpreter-manager.h : This graph shows which files directly or indirectly include this file :

Classes

- class stibbons : :exit_requested_exception
- class stibbons : :InterpreterManager
Class that will interpret stibbons language.

H.3.1 Detailed Description

InterpreterManager class header.

Author

Florian Galinier

Adrien Plazas

Version

1.1

Date

30/04/15

H.4 src/interpreter/parser.h File Reference

Parser class header.

```
#include "flex-scanner.h"
```

Include dependency graph for parser.h :

Classes

— class stibbons : :Parser
Class for syntactic analysis.

H.4.1 Detailed Description

Parser class header.

Author

Florian Galinier

Version

1.1

Date

26/04/15

Definition of the Parser class.

H.5 src/interpreter/semantic-exception.h File Reference

SemanticException class header.

```
#include "interpreter-exception.h"
```

```
#include "../model/type.h"
```

```
#include <string>
```

Include dependency graph for semantic-exception.h : This graph shows which files directly or indirectly include this file :

Classes

— class stibbons : :SemanticException
Exception thrown when a semantic error occurred.

H.5.1 Detailed Description

SemanticException class header.

Author

Florian Galinier

Version

1.1

Date

28/03/15

Declaration of SemanticException class, used to prevent when interpreter find a semantic error.

H.6 src/interpreter/syntax-exception.h File Reference

SyntaxException class header.

```
#include "interpreter-exception.h"
```

Include dependency graph for syntax-exception.h : This graph shows which files directly or indirectly include this file :

Classes

— class stibbons : :SyntaxException
Exception thrown when a syntax error occurred.

H.6.1 Detailed Description

SyntaxException class header.

Author

Florian Galinier

Version

1.1

Date

28/03/15

Declaration of SyntaxException class, used to prevent when parser find a syntax error.

H.7 src/interpreter/tree.h File Reference

Interpreter class header.

```
#include <iostream>
#include <memory>
#include <vector>
#include <utility>
#include <tuple>
#include "../model/value.h"
```

Include dependency graph for tree.h : This graph shows which files directly or indirectly include this file :

Classes

— class stibbons : :Tree
Class that will represent a syntactic tree.

Typedefs

— typedef std : :shared_ptr< Tree > **stibbons** : :TreePtr

H.7.1 Detailed Description

Interpreter class header. WorldInterpreter class header.

Tree class header.

Author

Florian Galinier
Clément Simon

Version

1.1

Date

15/03/15

Declaration of Interpreter class.

Author

Florian Galinier
Clément Simon

Version

1.1

Date

13/03/15

Declaration of Tree class (used by Interpreter class).

Author

Florian Galinier
Clément Simon

Version

0.3

Date

15/03/15

Declaration of Interpreter class.

Author

Florian Galinier
Clément Simon
Adrien Plazas

Version

1.1

Date

08/04/15

Declaration of WorldInterpreter class.

H.8 src/model/agent.h File Reference

A class representing an agent.

```
#include "value.h"
#include <exception>
#include <utility>
#include <unordered_set>
#include <unordered_map>
#include <iostream>
#include <string>
#include <mutex>
#include "json_spirit.h"
#include "json_spirit_writer_template.h"
```

Include dependency graph for agent.h : This graph shows which files directly or indirectly include this file :

Classes

— class stibbons : :Agent
class Agent in stibbons

H.8.1 Detailed Description

A class representing an agent.

Author

Julia Bassoumi, Adrien Plazas

Version

1.1

Date

14/03/2015

H.9 src/model/boolean.h File Reference

A class representing a boolean.

```
#include "value.h"
#include "simple-value.h"
#include <memory>
```

Include dependency graph for boolean.h : This graph shows which files directly or indirectly include this file :

Classes

— class stibbons : :Boolean
A class representing a boolean.

Typedefs

— typedef std : :shared_ptr< Boolean > **stibbons : :BooleanPtr**

H.9.1 Detailed Description

A class representing a boolean.

Author

Julia Bassoumi, Adrien Plazas

Version

1.1

Date

14/03/2015

H.10 src/model/border-type.h File Reference

The BorderType header class.

This graph shows which files directly or indirectly include this file :

Enumerations

— enum **BorderType** { **NONE**, **BOUNCE**, **WRAP** }

H.10.1 Detailed Description

The BorderType header class.

Author

Julia Bassoumi, Adrien Plazas

Version

1.1

Date

14/03/2015

H.11 src/model/breed.h File Reference

A class representing a breed.

```
#include "agent.h"
#include "function.h"
#include "turtle.h"
#include "world.h"
#include <mutex>
#include <string>
#include <unordered_set>
```

Include dependency graph for breed.h : This graph shows which files directly or indirectly include this file :

Classes

— class stibbons : :Breed
A class representing a breed.

H.11.1 Detailed Description

A class representing a breed.

Author

Julia Bassoumi, Adrien Plazas

Version

1.1

Date

29/03/2015

H.12 src/model/changeable.h File Reference

The Changeable header class.

```
#include "color.h"
```

```
#include <functional>
```

Include dependency graph for `changeable.h` : This graph shows which files directly or indirectly include this file :

Classes

- class `stibbons` : `:Changeable`
A class which can trigger a callback when its object has changed.

H.12.1 Detailed Description

The Changeable header class.

Author

Julia Bassoumi, Adrien Plazas

Version

1.1

Date

14/03/2015

H.13 src/model/color.h File Reference

A color.

```
#include "value.h"
```

```
#include <stdexcept>
```

```
#include <string>
```

```
#include <mutex>
```

```
#include <memory>
```

Include dependency graph for `color.h` : This graph shows which files directly or indirectly include this file :

Classes

- class `stibbons` : `:Color`
A color.

Typedefs

— typedef std : :shared_ptr< Color > **stibbons** : :ColorPtr

H.13.1 Detailed Description

A color.

Author

Julia Bassoumi, Adrien Plazas

Version

1.1

Date

14/03/2015

H.14 src/model/function.h File Reference

A class representing a function.

```
#include "agent.h"
#include "table.h"
#include "value.h"
#include <mutex>
#include <string>
#include <vector>
```

Include dependency graph for function.h : This graph shows which files directly or indirectly include this file :

Classes

— class stibbons : :Function
A class representing a function.

H.14.1 Detailed Description

A class representing a function.

Author

Adrien Plazas

Version

1.1

Date

10/04/2015

H.15 src/model/line.h File Reference

A colored polyline.

```

#include <vector>
#include <mutex>
#include "point.h"
#include "color.h"

```

Include dependency graph for line.h : This graph shows which files directly or indirectly include this file :

Classes

- class stibbons : :Line
A colored polyline.

H.15.1 Detailed Description

A colored polyline.

Author

Julia Bassoumi, Adrien Plazas

Version

1.1

Date

14/03/2015

H.16 src/model/nil.h File Reference

A class to represent the null value.

```

#include "value.h"
#include "singleton.h"

```

Include dependency graph for nil.h : This graph shows which files directly or indirectly include this file :

Classes

- class stibbons : :Nil
A class to represent the null value.

Typedefs

- typedef std : :shared_ptr< Nil > **stibbons** : :NilPtr

H.16.1 Detailed Description

A class to represent the null value.

Author

Julia Bassoumi, Adrien Plazas

Version

1.1

Date

14/03/2015

H.17 src/model/number.h File Reference

A class representing a real number.

```
#include "value.h"
#include "simple-value.h"
#include <memory>
```

Include dependency graph for number.h : This graph shows which files directly or indirectly include this file :

Classes

— class stibbons : :Number
A class representing a real number.

Typedefs

— typedef std : :shared_ptr< Number > **stibbons** : :NumberPtr

H.17.1 Detailed Description

A class representing a real number.

Author

Julia Bassoumi, Adrien Plazas

Version

1.1

Date

14/03/2015

H.18 src/model/point.h File Reference

A point with a parametrable number of dimensions.

```
#include "border-type.h"
#include "size.h"
#include <stdexcept>
#include <mutex>
#include <vector>
```

Include dependency graph for point.h : This graph shows which files directly or indirectly include this file :

Classes

— class stibbons : :Point
A point with a parametrable number of dimensions.

H.18.1 Detailed Description

A point with a parametrable number of dimensions.

Author

Julia Bassoumi, Adrien Plazas

Version

1.1

Date

14/03/2015

H.19 src/model/simple-value.h File Reference

An abstract class who represent a values of a given type.

```
#include <mutex>
```

Include dependency graph for simple-value.h : This graph shows which files directly or indirectly include this file :

Classes

— class stibbons : :SimpleValue< T >
A class representing a simple value.

H.19.1 Detailed Description

An abstract class who represent a values of a given type.

Author

Julia Bassoumi, Adrien Plazas

Version

1.1

Date

14/03/2015

H.20 src/model/singleton.h File Reference

A class representing a singleton.

```
#include "value.h"
```

Include dependency graph for singleton.h : This graph shows which files directly or indirectly include this file :

Classes

— class stibbons : :Singleton< T >
A helper class to implementing a singleton.

H.20.1 Detailed Description

A class representing a singleton.

Author

Adrien Plazas

Version

1.1

Date

19/03/2015

H.21 src/model/size.h File Reference

A size with a parametrable number of dimensions.

```
#include <stdexcept>
```

```
#include <mutex>
```

Include dependency graph for size.h : This graph shows which files directly or indirectly include this file :

Classes

- class stibbons : :Size
A size with a parametrable number of dimensions.

H.21.1 Detailed Description

A size with a parametrable number of dimensions.

Author

Julia Bassoumi, Adrien Plazas

Version

1.1

Date

08/04/2015

H.22 src/model/standard-function.h File Reference

Classes implementing standard functions.

```
#include "function.h"
```

```
#include "singleton.h"
```

Include dependency graph for standard-function.h : This graph shows which files directly or indirectly include this file :

Classes

- class stibbons : :TypeOfFunction
A class returning the type of a value.
- class stibbons : :RandFunction
A class returning a random number.
- class stibbons : :RandomFunction
A class returning a random number in a range.
- class stibbons : :PrintFunction
A class printing on the standard output.
- class stibbons : :PrintlnFunction
A class printing a new line on the standard output.
- class stibbons : :TeleportFunction
A class teleporting a turtle to another location.
- class stibbons : :SendFunction
A class sending a message from a turtle to another.
- class stibbons : :InboxFunction

- *A class returning the number of unread messages.*
class stibbons : :DistanceToFunction
- *A class returning the distance to another turtle.*
class stibbons : :FaceFunction
- *A class making a turtle face another.*
class stibbons : :InRadiusFunction
- *A class making a turtle face another.*
class stibbons : :AskZonesFunction
- *A class applying a function to every zone.*
class stibbons : :SizeFunction
- *A class that return an array size.*

H.22.1 Detailed Description

Classes implementing standard functions.

Author

Adrien Plazas
Florian Galinier

Version

1.1

Date

10/04/2015

H.23 src/model/string.h File Reference

A class representing a string.

```
#include "value.h"
#include "simple-value.h"
#include <string>
#include <memory>
```

Include dependency graph for string.h : This graph shows which files directly or indirectly include this file :

Classes

- class stibbons : :String
A class representing a string.

Typedefs

- typedef std : :shared_ptr< String > stibbons : :StringPtr

H.23.1 Detailed Description

A class representing a string.

Author

Julia Bassoumi, Adrien Plazas

Version

1.1

Date

14/03/2015

H.24 src/model/turtle.h File Reference

The Turtle class header.

```
#include "changeable.h"
#include "color.h"
#include "point.h"
#include "world.h"
#include "zone.h"
#include "line.h"
#include "agent.h"
#include "breed.h"
#include "value.h"
#include <future>
#include <cmath>
#include <string>
#include <fstream>
#include <deque>
#include <stdexcept>
#include <system_error>
#include <unordered_map>
#include <mutex>
#include "json_spirit.h"
#include "json_spirit_writer_template.h"
```

Include dependency graph for turtle.h : This graph shows which files directly or indirectly include this file :

Classes

— class stibbons : :Turtle
class Turtle stibbons

Typedefs

— typedef std : :shared_ptr< World > **stibbons : :WorldPtr**
— typedef unsigned long **stibbons : :turtle_id**

H.24.1 Detailed Description

The Turtle class header.

Author

Julia Bassoumi, Adrien Plazas

Version

1.1

Date

08/04/2015

H.25 src/model/type-value.h File Reference

A class representing a type.

```
#include "value.h"
```

```
#include "simple-value.h"
```

```
#include <memory>
```

Include dependency graph for type-value.h :

Classes

— class stibbons : :TypeValue
A class representing a type.

Typedefs

— typedef std : :shared_ptr
< TypeValue > **stibbons** : :TypeValuePtr

H.25.1 Detailed Description

A class representing a type.

Author

Adrien Plazas

Version

1.1

Date

29/04/2015

H.26 src/model/user-function.h File Reference

A class representing a user defined function.

```
#include "function.h"
```

```
#include "../interpreter/interpreter-manager.h"
```

```
#include "../interpreter/tree.h"
```

Include dependency graph for user-function.h :

Classes

— class stibbons : :UserFunction
A class representing a user defined function.

H.26.1 Detailed Description

A class representing a user defined function.

Author

Adrien Plazas

Version

1.1

Date

10/04/2015

H.27 src/model/value.h File Reference

An abstract class who represent a values of a given type.

```
#include "type.h"
#include <memory>
#include <stdexcept>
```

Include dependency graph for value.h : This graph shows which files directly or indirectly include this file :

Classes

- class stibbons : :Value
An abstract class who represent a values of a given type.
- class stibbons : :GenericValue< T >
An template class easing the implementation of a Value class.

H.27.1 Detailed Description

An abstract class who represent a values of a given type.

Author

Julia Bassoumi, Adrien Plazas

Version

1.1

Date

08/04/2015

Author

Julia Bassoumi, Adrien Plazas

Version

1.1

Date

14/03/2015

H.28 src/model/world.h File Reference

The World class header.

```

#include "border-type.h"
#include "changeable.h"
#include "size.h"
#include "zone.h"
#include "turtle.h"
#include "line.h"
#include "breed.h"
#include "function.h"
#include "number.h"
#include "standard-function.h"
#include "json_spirit.h"
#include "json_spirit_writer_template.h"
#include <ctime>
#include <vector>
#include <memory>
#include <mutex>
#include <fstream>
#include <unordered_map>
#include <unordered_set>
#include <stdexcept>

```

Include dependency graph for world.h : This graph shows which files directly or indirectly include this file :

Classes

- class stibbons : :World
A world consisting of zones, breeds, turtles and lines.

Typedefs

- typedef unsigned long stibbons : :zone_id

H.28.1 Detailed Description

The World class header.

Author

Julia Bassoumi, Adrien Plazas

Version

1.1

Date

14/03/2015

H.29 src/model/zone.h File Reference

The Zone header class.

```

#include "world.h"
#include "color.h"
#include "number.h"
#include "string.h"
#include "boolean.h"
#include "type.h"
#include "agent.h"
#include <stdexcept>
#include <system_error>
#include <unordered_map>
#include <mutex>
#include "json_spirit.h"
#include "json_spirit_writer_template.h"

```

Include dependency graph for zone.h : This graph shows which files directly or indirectly include this file :

Classes

— class stibbons : :Zone
class Zone

H.29.1 Detailed Description

The Zone header class.

Author

Julia Bassoumi, Adrien Plazas

Version

1.1

Date

14/03/2015

H.30 src/qt/line-number-area.h File Reference

The LineNumberArea class header.

```

#include <QPlainTextEdit>
#include <QWidget>
#include <QSize>
#include "stibbons-editor.h"

```

Include dependency graph for line-number-area.h :

Classes

— class LineNumberArea
Widget that contain the line number area.

H.30.1 Detailed Description

The LineNumberArea class header.

Author

Florian Galinier

Version

1.1

Date

05/05/2015

H.31 src/qt/runner.h File Reference

The Runner class header.

```
#include <QThread>
#include <mutex>
#include <string>
#include "../interpreter/interpreter-manager.h"
```

Include dependency graph for runner.h : This graph shows which files directly or indirectly include this file :

Classes

- class stibbons : :Runner
Bridge between the interpreter and Qt application.

H.31.1 Detailed Description

The Runner class header.

Author

Adrien Plazas

Version

1.1

Date

05/05/2015

H.32 src/qt/stibbons-highlighter.h File Reference

The Highlighter header class.

```
#include <QSyntaxHighlighter>
#include <QTextDocument>
#include <vector>
```

Include dependency graph for stibbons-highlighter.h : This graph shows which files directly or indirectly include this file :

Classes

- class StibbonsHighlighter
The highlighter class that highlight code in Qt application.

H.32.1 Detailed Description

The Highlighter header class.

Author

Florian Galiner

Version

1.1

Date

26/02/2015

H.33 src/qt/window.h File Reference

The Stibbons main window.

```
#include <QGridLayout>
#include <QMainWindow>
#include <QtWidgets>
#include <QApplication>
#include <QtGui>
#include <QTextEdit>
#include <QFile>
#include <QString>
#include <QTextCursor>
#include <QTextStream>
#include "../model/world.h"
#include "world-view.h"
#include "runner.h"
#include "stibbons-editor.h"
#include "stibbons-highlighter.h"
Include dependency graph for window.h :
```

Classes

— class stibbons : :Window
The window of Qt application.

H.33.1 Detailed Description

The Stibbons main window.

Author

Adrien Plazas

Version

1.1

Date

26/02/2015

H.34 src/qt/world-painter.h File Reference

The Stibbons world painter.

```
#include "../model/line.h"
#include "../model/turtle.h"
#include "../model/world.h"
#include <QPainter>
```

Include dependency graph for world-painter.h : This graph shows which files directly or indirectly include this file :

Classes

— class stibbons : :WorldPainter
The painter of the world in Qt application.

H.34.1 Detailed Description

The Stibbons world painter.

Author

Adrien Plazas

Version

1.1

Date

05/05/2015

H.35 src/qt/world-view.h File Reference

The Stibbons world view.

```
#include <QWidget>
#include "world-painter.h"
#include "../model/line.h"
#include "../model/turtle.h"
#include "../model/world.h"
```

Include dependency graph for world-view.h : This graph shows which files directly or indirectly include this file :

Classes

— class stibbons : :WorldView
The view of the world in Qt application.

H.35.1 Detailed Description

The Stibbons world view.

Author

Adrien Plazas

Version

1.1

Date

28/02/2015

Annexe I

Listing

I.1 Flex

```
/*
 * This file is part of Stibbons.
 *
 * Stibbons is free software: you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License as
 * published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Stibbons is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License
 * along with Stibbons. If not, see <http://www.gnu.org/licenses/>.
 */

%{
#include "flex-scanner.h"
#define YY_NO_UNISTD_H
%}

%option c++
%option nodefault
%option yyclass="FlexScanner"
%option yywrap nounput

%option case-insensitive

/* Define tokens */
%{
#include <sstream>
#include <cstdlib>
#include "y.tab.h"
```

```

#include "syntax-exception.h"
#include "../model/nil.h"
#include "../model/number.h"
#include "../model/string.h"
#include "../model/boolean.h"
#include "../model/color.h"
#include "../model/type-value.h"

#define YY_USER_ACTION loc->columns(yyval);
#define yyterminate() {}

size_t countNewLine(char* str) {
    auto count = 0;
    for(auto i=strlen(str)-1;i>=0;i--) {
        if(str[i] == '\n')
            ++count;
        else
            break;
    }

    return count;
}

int yyFlexLexer::yywrap() {
    return 1;
}
%}

id [_a-z][_a-z0-9]*

%x comment
%x spl_quote
%x dbl_quote
%x tpl_quote
%s for_state
%x end

/* Rules */
%%

std::string tmp;
%{
    loc->step();
%}

<INITIAL><<EOF>> { BEGIN(end); return yy::parser::token_type('\n'); }
<end><<EOF>> { BEGIN(INITIAL); return 0; }

/* Creating instruction tokens */
true|false { if(yytext[0] == 't' || yytext[0] == 'T') { pyylval->v =
    make_shared<stibbons::Boolean>(true); } else { pyylval->v =
    make_shared<stibbons::Boolean>(false); } return yy::parser::token

```

```

::BOOLEAN;}

null {return yy::parser::token::NIL;}

fd|forward {return yy::parser::token::FD;}

lt|turn_left {return yy::parser::token::LT;}

rt|turn_right {return yy::parser::token::RT;}

pd|pen_down {return yy::parser::token::PD;}

pu|pen_up {return yy::parser::token::PU;}

send {return yy::parser::token::SEND;}

recv {return yy::parser::token::RECV;}

die {return yy::parser::token::DIE;}

/*Creating comparison operations tokens*/
"==" {return yy::parser::token::EQ;}

"!=" {return yy::parser::token::NEQ;}

">" {return yy::parser::token::GT;}

">=" {return yy::parser::token::GEQ;}

"<" {return yy::parser::token::LS;}

"<=" {return yy::parser::token::LEQ;}

and|& {return yy::parser::token::AND;}

or|\| {return yy::parser::token::OR;}

xor|^ {return yy::parser::token::XOR;}

not|! {return yy::parser::token::NOT;}

/*Creating loops (except 'for') and conditional tokens*/
repeat {return yy::parser::token::RPT;}

while {return yy::parser::token::WHL;}

if {return yy::parser::token::IF;}

else {return yy::parser::token::ELSE;}

/*Creating agent and function tokens*/

```

```

new {return yy::parser::token::NEW;}

agent {return yy::parser::token::AGT;}

function {return yy::parser::token::FCT;}

/*Creating tokens type*/
null_t {pyylval->v=make_shared<stibbons::TypeValue>(stibbons::Type::
    NIL);
    return yy::parser::token::TYPE;}

number_t {pyylval->v=make_shared<stibbons::TypeValue>(stibbons::Type
    ::NUMBER);
    return yy::parser::token::TYPE;}

boolean_t {pyylval->v=make_shared<stibbons::TypeValue>(stibbons::Type
    ::BOOLEAN);
    return yy::parser::token::TYPE;}

string_t {pyylval->v=make_shared<stibbons::TypeValue>(stibbons::Type
    ::STRING);
    return yy::parser::token::TYPE;}

color_t {pyylval->v=make_shared<stibbons::TypeValue>(stibbons::Type::
    COLOR);
    return yy::parser::token::TYPE;}

table_t {pyylval->v=make_shared<stibbons::TypeValue>(stibbons::Type::
    TABLE);
    return yy::parser::token::TYPE;}

type_t {pyylval->v=make_shared<stibbons::TypeValue>(stibbons::Type::
    TYPE);
    return yy::parser::token::TYPE;}

turtle_t {pyylval->v=make_shared<stibbons::TypeValue>(stibbons::Type
    ::TURTLE);
    return yy::parser::token::TYPE;}

zone_t {pyylval->v=make_shared<stibbons::TypeValue>(stibbons::Type::
    ZONE);
    return yy::parser::token::TYPE;}

world_t {pyylval->v=make_shared<stibbons::TypeValue>(stibbons::Type::
    WORLD);
    return yy::parser::token::TYPE;}

/*Creation loop 'for' tokens*/
for { BEGIN(for_state); return yy::parser::token::FOR; }

<for_state>in {BEGIN(INITIAL); return yy::parser::token_type(' ');}

```

```

<for_state>': ' {BEGIN(INITIAL); return yy::parser::token_type(':');}

/*Creating string tokens*/
\" \" \" { tmp = std::string(); BEGIN(tpl_quote); }

<tpl_quote>\" \" \" { BEGIN(INITIAL);
    pyylval->v=make_shared<stibbons::String>(tmp);
    return yy::parser::token::STRING;}
<tpl_quote>[^\" ]+ { tmp += yytext; }
<tpl_quote>\" [^\" ] { tmp += yytext; }
<tpl_quote>\" \" [^\" ] { tmp += yytext; }

\" { tmp = std::string(); BEGIN(dbl_quote); }

<dbl_quote>[^\\\"\\n\\"]+ { tmp += yytext; }
<dbl_quote>\\\" { tmp += yytext[1]; }
<dbl_quote>\\n { throw stibbons::SyntaxException("New line in a \"
    string\",loc->begin); }
<dbl_quote>\" { BEGIN(INITIAL);
    pyylval->v=make_shared<stibbons::String>(tmp);
    return yy::parser::token::STRING;}

\' { tmp = std::string(); BEGIN(spl_quote); }

<spl_quote>[^\\\"\\n\\'] + { tmp += yytext; }
<spl_quote>\\\' { tmp += yytext[1]; }
<spl_quote>\\n { throw stibbons::SyntaxException("New line in a \'
    string\",loc->begin); }
<spl_quote>\' { BEGIN(INITIAL);
    pyylval->v=make_shared<stibbons::String>(tmp);
    return yy::parser::token::STRING;}

<spl_quote,dbl_quote>\\\\\\\\ { tmp += yytext[1]; }
<spl_quote,dbl_quote>{
    \\n { tmp += '\\n'; }
    \\r { tmp += '\\r'; }
    \\t { tmp += '\\t'; }
    \\f { tmp += '\\f'; }
    \\0 { tmp += '\\0'; }
}

/*Creating number tokens*/
[0-9]+(\\. [0-9]*)?\\. [0-9]+ {std::istringstream iss(yytext); double n;
    iss >> n; pyylval->v=make_shared<stibbons::Number>(n); return yy
    ::parser::token::NUMBER;}

/*Creating color tokens*/
#([a-f0-9]{6}|[a-f0-9]{3}) {pyylval->v=make_shared<stibbons::Color>(
    yytext); return yy::parser::token::COLOR;}

/*Creating identifying variables tokens*/

```

```

{id} {pyylval->v=make_shared<stibbons::String>(yytext); return yy::
    parser::token::ID;}

/*Creating comment tokens*/
"//[^\n]*\n* { loc->lines(countNewLine(yytext)); loc->step(); return
    yy::parser::token_type('\n');}

"/*" {BEGIN(comment);}

<comment>[^\n]* {loc->lines(countNewLine(yytext)); loc->step();}
<comment>"*" + [^\n]*
<comment>"*" + "/" {BEGIN(INITIAL);}

[ \r\t]+ {loc->step();}

\n\n* { loc->lines(yyval); loc->step(); return yy::parser::
    token_type(yytext[0]);}

. {return yy::parser::token_type(yytext[0]);}

%%

/*
 * Editor modelines - http://www.wireshark.org/tools/modelines.html
 *
 * Local variables:
 * mode: c++
 * c-basic-offset: 4
 * tab-width: 4
 * indent-tabs-mode: t
 * truncate-lines: 1
 * End:
 *
 * vim: set ft=cpp ts=4 sw=4 sts=4
 */

```

I.2 Bison

```

/*
 * This file is part of Stibbons.

 * Stibbons is free software: you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License as
 * published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.

 * Stibbons is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU Lesser General Public License for more details.

```

```

* You should have received a copy of the GNU Lesser General Public
   License
   * along with Stibbons. If not, see <http://www.gnu.org/licenses/>.
   */

%skeleton "lalr1.cc"
%defines
%locations
%parse-param { stibbons::FlexScanner &scanner }
%parse-param { stibbons::TreePtr t }
%parse-param { stibbons::TablePtr w }
%lex-param { stibbons::FlexScanner &scanner }

%code requires {
    namespace stibbons {
        class FlexScanner;
    }
#include "tree.h"
#include "../model/table.h"
#include "../model/string.h"
#include "../model/number.h"
#define YYSTYPE struct { stibbons::ValuePtr v; stibbons::TreePtr tr;
    int tok; }
    std::string toString(const int& tok);
}

%code {
#include "flex-scanner.h"
#include "syntax-exception.h"
    using namespace std;

    static int yylex(yy::parser::semantic_type* pyylval,
                    yy::parser::location_type* loc,
                    stibbons::FlexScanner &scanner);

    void yy::parser::error(const location& loc, string const& s) {
        throw stibbons::SyntaxException(s.c_str(), loc.begin);
    }

    static int yylex(yy::parser::semantic_type* pyylval,
                    yy::parser::location_type* loc,
                    stibbons::FlexScanner &scanner) {
        return scanner.yylex(pyylval, loc);
    }

    std::string toString(const int& tok) {
        switch(tok) {
            case yy::parser::token::SEQ:
                return "SEQ";
            case yy::parser::token::FD:

```

```

    return "FD" ;
case yy::parser::token::RT:
    return "RT" ;
case yy::parser::token::LT:
    return "LT" ;
case yy::parser::token::PU:
    return "PU" ;
case yy::parser::token::PD:
    return "PD" ;
case yy::parser::token::SEND:
    return "SEND" ;
case yy::parser::token::RECV:
    return "RECV" ;
case yy::parser::token::DIE:
    return "DIE" ;
case yy::parser::token::AND:
    return "AND" ;
case yy::parser::token::OR:
    return "OR" ;
case yy::parser::token::XOR:
    return "XOR" ;
case yy::parser::token::NOT:
    return "NOT" ;
case yy::parser::token::EQ:
    return "EQ" ;
case yy::parser::token::NEQ:
    return "NEQ" ;
case yy::parser::token::GT:
    return "GT" ;
case yy::parser::token::GEQ:
    return "GEQ" ;
case yy::parser::token::LS:
    return "LS" ;
case yy::parser::token::LEQ:
    return "LEQ" ;
case yy::parser::token::CALL:
    return "CALL" ;
case yy::parser::token::RPT:
    return "RPT" ;
case yy::parser::token::WHL:
    return "WHL" ;
case yy::parser::token::IF:
    return "IF" ;
case yy::parser::token::ELSE:
    return "ELSE" ;
case yy::parser::token::FCT:
    return "FCT" ;
case yy::parser::token::NEW:
    return "NEW" ;
case yy::parser::token::AGT:
    return "AGT" ;

```



```

    case yy::parser::token::NUMBER:
        return "NUMBER";
    case yy::parser::token::STRING:
        return "STRING";
    case yy::parser::token::COLOR:
        return "COLOR";
    case yy::parser::token::BOOLEAN:
        return "BOOLEAN";
    case yy::parser::token::NIL:
        return "NIL";
    case yy::parser::token::ID:
        return "ID";
    case yy::parser::token::TABLE:
        return "TABLE";
    case yy::parser::token::PAIR:
        return "PAIR";
    case yy::parser::token::UNARYMINUS:
        return "-";
    case yy::parser::token::FOR:
        return "FOR";
    default:
        return std::string(1,static_cast<char>(tok));
}
}
}

```

```

%token SEQ 0          "sequence "
%token FD             "forward "
%token LT             "turn-left "
%token RT             "turn-right "
%token PU             "pen-up "
%token PD             "pen-down "
%token SEND           "send "
%token RECV           "recv "
%token DIE            "die "
%token AND            "and "
%token OR             "or "
%token XOR            "xor "
%token NOT            "not "
%token EQ             "=="
%token NEQ            "!="
%token GT             ">"
%token GEQ            ">="
%token LS             "<"
%token LEQ            "<="
%token RPT            "repeat "
%token WHL            "while "
%token IF             "if "
%token ELSE           "else "
%token FCT            "function "
%token NEW            "new "

```

```

%token AGT          "agent "
%token CALL         "call "
%token TABLE       "table "
%token PAIR          "pair "
%token UNARYMINUS    "-"
%token FOR           "for "
%token IN            "in "
%token ATT_ID
%token TAB_ID
%token <v> NUMBER
%token <v> STRING
%token <v> COLOR
%token <v> BOOLEAN
%token <v> NIL
%token <v> TYPE
%token <v> ID
%type <tr> nb_agt
%type <tr> statement_list
%type <tr> statement_list_bloc
%type <tr> statement
%type <tr> instr_turtle
%type <tr> compl_statement
%type <tr> bloc
%type <tr> expr
%type <tr> lit
%type <tr> assignment_expression
%type <tr> expr_statement
%type <tr> expr_no_separator
%type <tr> decl_statement
%type <tr> creat_statement
%type <tr> primary_expr
%type <tr> loop
%type <tr> for_expr
%type <tr> selection
%type <tr> decl_list
%type <tr> id_list
%type <tr> initializer_list
%type <tr> table_list
%type <tr> expr_list
%type <tr> pair_list
%type <tr> pair
%type <tok> binary_operator
%type <tok> decl_id

%right '='
%left AND OR XOR
%left EQ NEQ
%left GT GEQ LS LEQ
%left '+' '-'
%left '*' '/' '%'
%right MOINSUNAIRE NOT

```

```

%nonassoc '{' '}' '(' ')',
%%
code : world_dir_list statement_list { t->addChild($2); }
| statement_list { t->addChild($1); }
;

//Storage of the world's directives

world_dir_list : world_dir {}
| world_dir_list world_dir {}
;

world_dir : '%' ID lit '\n'
{
    auto val = std::get<1>($3->getNode());
    w->setValue(dynamic_pointer_cast<stibbons::String>($2)->getValue(),
        val);
}
| '%' ID ID '\n'
{
    w->setValue(dynamic_pointer_cast<stibbons::String>($2)->getValue(),
        $3);
}
| '\n' {}
;

// General languages struct
// Storage of tokens and contents in a tree

//Storage of statement and bloc
statement_list : statement
{
    $$ = $1;
}
| statement_list statement
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(0,nullptr);
    try {
        t1->mergeTree($1);
    }
    catch(std::exception& e) {
        t1->addChild($1);
    }
    t1->addChild($2);
    $$ = t1;
}
| { $$ = nullptr; }
;

statement : expr_statement { $$ = $1; }
| compl_statement { $$ = $1; }

```

```

;

compl_statement : bloc { $$ = $1; }
| decl_statement { $$ = $1; }
| selection { $$ = $1; }
| loop { $$ = $1; }
| compl_statement '\n' { $$ = $1; }
;

bloc : '{' '}' { $$ = nullptr; }
| '{' statement_list_bloc '}' { $$ = $2; }
;

statement_list_bloc : statement_list { $$ = $1; }
| statement_list expr_no_separator {
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(0, nullptr);
    try {
        t1->mergeTree($1);
    }
    catch(std::exception& e) {
        t1->addChild($1);
    }
    t1->addChild($2);
    $$ = t1;
}
;

//Storage of declaration of object (id, variables, list, ...)
decl_statement : decl_id ID decl_list bloc {
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>($1, $2);
    t1->setPosition({@1.begin.line, @1.begin.column});
    t1->addChild($4);
    t1->appendChildren($3);
    $$ = t1;
}
;

decl_id : AGT { $$ = yy::parser::token::AGT; }
| FCT { $$ = yy::parser::token::FCT; }
;

decl_list : '(' ')' { $$ = nullptr; }
| '(' id_list ')' { $$ = $2; }
;

id_list : ID
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::ID, nullptr);
    t1->addChild(make_shared<stibbons::Tree>(yy::parser::token::ID, $1))
    ;
}
;

```

```

    $$ = t1;
}
| id_list ',' ID
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::ID, nullptr);
    t1->appendChildren($1);
    t1->addChild(make_shared<stibbons::Tree>(yy::parser::token::ID,$3))
        ;
    $$ = t1;
};

//Storage of conditionnal expression
selection : IF expr statement
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::IF, nullptr);
    t1->addChild($2);
    t1->addChild($3);
    t1->setPosition({@1.begin.line ,@1.begin.column});
    $$ = t1;
}
| IF expr statement ELSE statement
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::IF, nullptr);
    t1->addChild($2);
    t1->addChild($3);
    t1->addChild($5);
    t1->setPosition({@1.begin.line ,@1.begin.column});
    $$ = t1;
};

//Storage of loop expression
loop : RPT expr statement
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::RPT, nullptr);
    t1->addChild($2);
    t1->addChild($3);
    t1->setPosition({@1.begin.line ,@1.begin.column});
    $$ = t1;
}
| WHL expr statement
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::WHL, nullptr);
    t1->addChild($2);
    t1->addChild($3);
    t1->setPosition({@1.begin.line ,@1.begin.column});
    $$ = t1;
}

```

```

}
| FOR for_expr statement
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::FOR, nullptr);
    t1->addChild($2);
    t1->addChild($3);
    t1->setPosition({@1.begin.line ,@1.begin.column});
    $$ = t1;
}
;

for_expr : '(' for_expr ')' { $$ = $2; }
| ID ':' expr
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::FOR,$1);
    t1->addChild($3);
    t1->setPosition({@1.begin.line ,@1.begin.column});
    $$ = t1;
}
| ID ',' ID ':' expr
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::FOR,$3);
    t1->addChild($5);
    t1->addChild(make_shared<stibbons::Tree>(yy::parser::token::ID,$1))
        ;
    t1->setPosition({@1.begin.line ,@1.begin.column});
    $$ = t1;
}
;

//Storage of expression
expr_statement : '\n' { $$ = nullptr; }
| expr '\n' { $$ = $1; }
| instr_turtle '\n' { $$ = $1; }
| creat_statement '\n' { $$ = $1; }
;

expr_no_separator : expr { $$ = $1; }
| instr_turtle { $$ = $1; }
| creat_statement { $$ = $1; }
;

expr : assignment_expression { $$ = $1; }
| '(' expr ')' { $$ = $2; }
| expr binary_operator expr
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>($2, nullptr);
    t1->addChild($1);

```

```

    t1->addChild($3);
    t1->setPosition({@1.begin.line,@1.begin.column});
    $$ = t1;
}
| '-' expr %prec UNARYMINUS
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::UNARYMINUS, nullptr);
    t1->addChild($2);
    $$ = t1;
}
| NOT expr
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::NOT, nullptr);
    t1->addChild($2);
    $$ = t1;
}
| primary_expr { $$ = $1; }
| lit { $$ = $1; }
;

//Storage of binary operator
binary_operator : '+' { $$ = '+'; }
| '-' { $$ = '-'; }
| '/' { $$ = '/'; }
| '*' { $$ = '*'; }
| '%' { $$ = '%'; }
| AND { $$ = yy::parser::token::AND; }
| OR { $$ = yy::parser::token::OR; }
| XOR { $$ = yy::parser::token::XOR; }
| EQ { $$ = yy::parser::token::EQ; }
| NEQ { $$ = yy::parser::token::NEQ; }
| GT { $$ = yy::parser::token::GT; }
| GEQ { $$ = yy::parser::token::GEQ; }
| LS { $$ = yy::parser::token::LS; }
| LEQ { $$ = yy::parser::token::LEQ; }
;

//Storage of assignment expression
assignment_expression : primary_expr '=' expr
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>('=', nullptr);
    t1->addChild($1);
    t1->addChild($3);
    $$ = t1;
}
| primary_expr '=' '{' table_list '}'
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>('=', nullptr);
    t1->addChild($1);

```

```

    t1->addChild($4);
    $$ = t1;
}
;

//Storage of identifying attributes
primary_expr : ID
{
    $$ = make_shared<stibbons::Tree>(yy::parser::token::ID,$1);
}
| primary_expr '.' ID
{
    auto t1 = make_shared<stibbons::Tree>(yy::parser::token::ATT_ID,$3)
    ;
    t1->addChild($1);
    $$ = t1;
}
| primary_expr '[' expr ']'
{
    auto t1 = make_shared<stibbons::Tree>(yy::parser::token::TAB_ID,
        nullptr);
    t1->addChild($1);
    t1->addChild($3);
    $$ = t1;
}
| primary_expr '[' ']'
{
    auto t1 = make_shared<stibbons::Tree>(yy::parser::token::TAB_ID,
        nullptr);
    t1->addChild($1);
    $$ = t1;
}
| ID '(' ')'
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::CALL,$1);
    t1->setPosition({@1.begin.line,@1.begin.column});
    $$ = t1;
}
| ID '(' initializer_list ')'
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::CALL,$1);
    t1->setPosition({@1.begin.line,@1.begin.column});
    t1->appendChildren($3);
    $$ = t1;
}
;

//Storage for table object

```



```

table_list : expr_list { $$ = $1; }
| pair_list { $$ = $1; }
| { $$ = make_shared<stibbons::Tree>(yy::parser::token::TABLE, nullptr); };

expr_list : expr
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::token::TABLE, nullptr);
    t1->addChild($1);
    $$ = t1;
}
| expr_list ',' expr
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::token::TABLE, nullptr);
    t1->appendChildren($1);
    t1->addChild($3);
    $$ = t1;
}
;

pair_list : pair
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::token::TABLE, nullptr);
    t1->addChild($1);
    $$ = t1;
}
| pair_list ',' pair
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::token::TABLE, nullptr);
    t1->appendChildren($1);
    t1->addChild($3);
    $$ = t1;
}
;

pair : expr ':' expr
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::token::PAIR, nullptr);
    t1->addChild($1);
    t1->addChild($3);
    $$ = t1;
}
;

//Storage of call expression
initializer_list : expr

```

```

{
  stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
    token::CALL, nullptr);
  t1->addChild($1);
  $$ = t1;
}
| initializer_list ',' expr
{
  stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
    token::CALL, nullptr);
  t1->appendChildren($1);
  t1->addChild($3);
  $$ = t1;
}
;

//Storage for creation of new agent
creat_statement : nb_agt NEW AGT bloc
{
  stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
    token::NEW, nullptr);
  t1->setPosition({@1.begin.line ,@1.begin.column});
  t1->addChild($1);
  t1->addChild($4);
  $$ = t1;
}
| nb_agt NEW ID '(' initializer_list ')'
{
  stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
    token::NEW,$3);
  t1->setPosition({@1.begin.line ,@1.begin.column});
  t1->addChild($1);
  t1->addChild($5);
  $$ = t1;
}
| nb_agt NEW ID '(' ')'
{
  stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
    token::NEW,$3);
  t1->addChild($1);
  t1->addChild(make_shared<stibbons::Tree>(yy::parser::token::NEW,
    nullptr));
  t1->setPosition({@1.begin.line ,@1.begin.column});
  $$ = t1;
}
| primary_expr '=' creat_statement
{
  stibbons::TreePtr t1 = make_shared<stibbons::Tree>('=', nullptr);
  t1->addChild($1);
  t1->addChild($3);
  $$ = t1;
}

```

```

}
;

nb_agt : NUMBER
{
    $$ = make_shared<stibbons::Tree>(yy::parser::token::NUMBER, $1);
}
| primary_expr
{
    $$ = $1;
}
|
{
    stibbons::ValuePtr nb = make_shared<stibbons::Number>(1.0);
    $$ = make_shared<stibbons::Tree>(yy::parser::token::NUMBER, nb);
}
;

// Storage of turtle instructions

instr_turtle : FD expr
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::FD, nullptr);
    t1->addChild($2);
    t1->setPosition({@1.begin.line, @1.begin.column});
    $$ = t1;
}
| LT expr
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::LT, nullptr);
    t1->addChild($2);
    t1->setPosition({@1.begin.line, @1.begin.column});
    $$ = t1;
}
| RT expr
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::RT, nullptr);
    t1->addChild($2);
    t1->setPosition({@1.begin.line, @1.begin.column});
    $$ = t1;
}
| PU
{
    $$ = make_shared<stibbons::Tree>(yy::parser::token::PU, nullptr);
}
| PD
{
    $$ = make_shared<stibbons::Tree>(yy::parser::token::PD, nullptr);
}

```

```

}
| SEND expr expr
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::SEND, nullptr);
    t1->addChild($2);
    t1->addChild($3);
    t1->setPosition({@1.begin.line ,@1.begin.column});
    $$ = t1;
}
| SEND expr
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::SEND, nullptr);
    t1->addChild($2);
    t1->setPosition({@1.begin.line ,@1.begin.column});
    $$ = t1;
}
| RECV primary_expr primary_expr
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::RECV, nullptr);
    t1->addChild($2);
    t1->addChild($3);
    t1->setPosition({@1.begin.line ,@1.begin.column});
    $$ = t1;
}
| RECV primary_expr
{
    stibbons::TreePtr t1 = make_shared<stibbons::Tree>(yy::parser::
        token::RECV, nullptr);
    t1->addChild($2);
    t1->setPosition({@1.begin.line ,@1.begin.column});
    $$ = t1;
}
| DIE
{
    $$ = make_shared<stibbons::Tree>(yy::parser::token::DIE, nullptr);
};

//Storage of literals
lit : NUMBER { $$ = make_shared<stibbons::Tree>(yy::parser::token::
    NUMBER,$1); }
| STRING { $$ = make_shared<stibbons::Tree>(yy::parser::token::STRING
    ,$1); }
| BOOLEAN { $$ = make_shared<stibbons::Tree>(yy::parser::token::
    BOOLEAN,$1); }
| COLOR { $$ = make_shared<stibbons::Tree>(yy::parser::token::COLOR,
    $1); }
| NIL { $$ = make_shared<stibbons::Tree>(yy::parser::token::NIL,$1);
    }

```

```
| TYPE { $$ = make_shared<stibbons::Tree>(yy::parser::token::TYPE, $1)
| ; };

%%

/*
 * Editor modelines  -  http://www.wireshark.org/tools/modelines.
 *    html
 *
 * Local variables:
 * mode: c++
 * c-basic-offset: 4
 * tab-width: 4
 * indent-tabs-mode: t
 * truncate-lines: 1
 * End:
 *
 * vim: set ft=cpp ts=4 sw=4 sts=4
 */
```

I.3 CppUnit

```
/*
 * This file is part of Stibbons.
 *
 * Stibbons is free software: you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License as
 * published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Stibbons is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License
 * along with Stibbons. If not, see <http://www.gnu.org/licenses/>.
 */

#include "../model/agent.h"
#include "../model/zone.h"
#include "../model/turtle.h"
#include "../model/number.h"
#include "../model/string.h"
#include "../model/color.h"

#include <cppunit/TestCase.h>
#include <cppunit/extensions/HelperMacros.h>
using namespace stibbons;
using namespace std;
```

```

using namespace CppUnit;

class TestAgent : public TestCase {
    CPPUNIT_TEST_SUITE(TestAgent);
    CPPUNIT_TEST(getValuesT);
    CPPUNIT_TEST(getValuesZ);
    CPPUNIT_TEST(changeValueT);
    CPPUNIT_TEST(getValuesW);
    CPPUNIT_TEST_SUITE_END();

public :
    TurtlePtr t;
    ZonePtr z;
    WorldPtr w;

    void setUp() {
        auto worldSize = Size(2);
        worldSize.setValue(0, 10);
        worldSize.setValue(1, 10);
        auto zoneSize = Size(2);
        zoneSize.setValue(0, 10);
        zoneSize.setValue(1, 10);
        auto warp = vector<BorderType>();
        warp.push_back(BorderType::NONE);
        warp.push_back(BorderType::NONE);
        w=World::construct(worldSize, zoneSize, warp);
        w->setProperty("tortue", make_shared<String>("bleu"));
        w->setProperty("color", make_shared<Color>());

        z = Zone::construct(w);
        z->setProperty("couleur", make_shared<String>("chat"));

        t=Turtle::construct(nullptr, w, 0);
        t->setProperty("chasse", make_shared<Number>(6.0));
        t->setProperty("couleur", make_shared<String>("chat"));
    }

    void getValuesT() {
        cout << "TestAgent::getValuesT" << endl;
        auto chasse = t->getProperty("chasse");
        CPPUNIT_ASSERT (Type::NUMBER == chasse->getType());
        auto chasse_reel = dynamic_pointer_cast<Number>(chasse);
        CPPUNIT_ASSERT_EQUAL (6.0, chasse_reel->getValue());
    }

    void changeValueT(){
        cout << "TestAgent::changeValueT" << endl;
        t->setProperty("chasse", make_shared<Number>(7.7));

        auto search = t->getProperty("chasse");
        CPPUNIT_ASSERT (

```

```

        search->getType() == Type::NUMBER &&
        dynamic_pointer_cast<Number>(search)->getValue() == 7.7
    );
}

void getValuesZ() {
    cout << "TestAgent::getValuesZ" << endl;
    auto couleur = z->getProperty("couleur");
    CPPUNIT_ASSERT (Type::STRING == couleur->getType());
    auto couleur_reel = dynamic_pointer_cast<String> (couleur);
    CPPUNIT_ASSERT ("chat" == couleur_reel->getValue());
}

void getValuesW() {
    cout << "TestAgent::getValuesW" << endl;
    auto couleur = w->getProperty("color");
    CPPUNIT_ASSERT (Type::COLOR == couleur->getType());
    auto tortue = w->getProperty("tortue");
    auto tt = dynamic_pointer_cast<String> (tortue);
    CPPUNIT_ASSERT ("bleu" == tt->getValue());
}

};

/* enregistrement du nom des test dans le registre */
CPPUNIT_TEST_SUITE_NAMED_REGISTRATION(TestAgent, "TestAgent");

/*
 * Editor modelines  -  http://www.wireshark.org/tools/modelines.html
 *
 * Local variables:
 * c-basic-offset: 4
 * tab-width: 4
 * indent-tabs-mode: t
 * truncate-lines: 1
 * End:
 *
 * vim: set ft=cpp ts=4 sw=4 sts=4
 */

```

I.4 Json Spirit

```

{
    "time" : "Mon Jun  1 17:53:28 2015\n",
    "World" : {
        "WorldSize" : [
            3.0000000000000000,
            3.0000000000000000
        ],
        "ZoneSize" : [
            10.000000000000000,
            10.000000000000000
        ]
    }
}

```

```

],
"properties" : {
},
"Turtles" : [
  {
    "1" : {
      "color" : "#000000",
      "angle" : 0.0000000000000000,
      "position" : [
        0.0000000000000000,
        0.0000000000000000
      ],
      "properties" : {
      },
      "parent" : "world"
    }
  },
  {
    "0" : {
      "Breed" : "lila",
      "color" : "#000000",
      "angle" : 0.0000000000000000,
      "position" : [
        0.0000000000000000,
        0.0000000000000000
      ],
      "properties" : {
        "find" : true,
        "name" : "Kelly",
        "a" : 4.0000000000000000
      },
      "parent" : "world"
    }
  }
],
"Zones" : [
  {
    "0" : {
      "color" : "#ffffff",
      "properties" : {
      }
    }
  },
  {
    "1" : {
      "color" : "#ffffff",
      "properties" : {
      }
    }
  }
],
{

```



```

    "2" : {
      "color" : "#ffffff",
      "properties" : {
      }
    },
    {
      "3" : {
        "color" : "#ffffff",
        "properties" : {
        }
      },
      {
        "4" : {
          "color" : "#ffffff",
          "properties" : {
          }
        },
        {
          "5" : {
            "color" : "#ffffff",
            "properties" : {
            }
          },
          {
            "6" : {
              "color" : "#ffffff",
              "properties" : {
              }
            },
            {
              "7" : {
                "color" : "#ffffff",
                "properties" : {
                }
              },
              {
                "8" : {
                  "color" : "#ffffff",
                  "properties" : {
                  }
                }
              }
            }
          }
        }
      }
    }
  ]
}

```

Bibliographie

- [bis, 2015] (2015). GNU Bison - The Yacc-compatible Parser Generator - GNU Project - Free Software Foundation. <http://www.gnu.org/software/bison/>. 20
- [Aho et al., 2007] Aho, A., Lam, M., Sethi, R., and Ullman, J. (2007). *Compilateurs : Principes, techniques et outils*. Pearson Education. 30
- [Ferber, 1995] Ferber, J. (1995). *Les Systèmes Multi Agents : vers une intelligence collective*. InterEditions. 10
- [Navarro, 2003] Navarro, J. (2003). Unit testing with CppUnit. <http://www.codeproject.com/Articles/5660/Unit-testing-with-CPPUnit>. 18
- [Paxson, 2014] Paxson, V. (2014). flex : The Fast Lexical Analyzer. <http://flex.sourceforge.net/>. 19
- [Pea, 1987] Pea, R. D. (1987). Logo programming and problem solving. *HAL : archives-ouvertes*. 12
- [Resnick et al., 2008] Resnick, M., Klopfer, E., et al. (2008). StarLogo on the web. <http://education.mit.edu/starlogo/>. 13
- [Stallman, 1988] Stallman, R. (1988). GDB : The GNU Project Debugger. <http://www.gnu.org/software/gdb/>. 19
- [Wilensky, 1999] Wilensky, U. (1999). NetLogo. <http://ccl.northwestern.edu/netlogo/>. 13
- [Wilkinson, 2014] Wilkinson, J. W. (2014). JSON Spirit : A C++ JSON Parser/Generator Implemented with Boost Spirit. <http://www.codeproject.com/Articles/20027/JSON-Spirit-A-C-JSON-Parser-Generator-Implemented>. 24