UNIVERSITE CATHOLIQUE DE LOUVAIN

ECOLE POLYTECHNIQUE DE LOUVAIN

RSnap: Une plate-forme web pour accompagner l'apprentissage de la programmation des 10 à 14 ans

Superviseur : Pierre SCHAUS
Lecteurs : Chantal PONCIN
Sébastien COMBÉFIS
Nicolas DETIENNE

Mémoire présenté en vue de l'obtention du grade de master 60 crédits en sciences informatiques

par Simon HOCK

Louvain-la-Neuve

Année académique 2014-2015

RSnap:
Une plate-forme web pour accompagner l'apprentissage de la programmation des 10 à 14 ans

Mémoire - LSINF2991

Étudiant :

Simon HOCK

Année : SINF20M1

NOMA : 39-47-10-00

Superviseur :

Pierre SCHAUS

Lecteurs :

Chantal PONCIN Sébastien COMBÉFIS Nicolas DETIENNE

Remerciements

Au terme de la rédaction de ce mémoire, je suis convaincu que la mise en place d'un outil offrant une aide aux étudiants dans l'apprentissage de la programmation est loin d'être un travail solitaire. Je n'aurais jamais pu réaliser ce travail sans le soutien de plusieurs personnes dont l'intérêt exprimé à l'égard de l'avancement de mon projet m'a permis de progresser efficacement.

En premier lieu, je tiens à remercier mon promoteur de mémoire, Monsieur Pierre Schaus qui a accepté de superviser ce travail.

Mes remerciements vont également à Madame Chantal Poncin et Monsieur Sébastien Combéfis pour avoir participé à ce jury de mémoire. Je les remercie pour leur grande disponibilité dans la relecture du mémoire, leurs conseils éclairés ainsi que leur expérience pour la réalisation du projet.

Je souhaite aussi exprimer ma gratitude au personnel du service informatique qui a gracieusement mis les locaux et les outils informatiques à la disposition des étudiants afin de leur permettre l'apprentissage sur le programme.

Un tout grand merci à Monsieur Dominique Deconinck pour ses talents d'infographiste qui ont permis d'avoir un cours de base avec une charte graphique cohérente.

Mes sincères remerciements s'adressent à l'ensemble des professeurs et élèves qui ont eu la gentillesse de participer au projet et de répondre au questionnaire d'évaluation.

Enfin, je dédie ce mémoire aux membres de ma famille et à mes amis qui m'ont soutenu dans la réalisation de ce mémoire. Je tiens à remercier tout particulièrement mon père pour son soutien et son écoute ainsi que sa compagne pour ses relectures attentives et ses corrections pointilleuses.

Abstract

Informatics is everywhere and new technologies comes in everyday life. For this reason, computer scientists is one of the most wanted occupation. It is then important to teach it to people and especially young people because they will come across a lot of it during their life.

This thesis is about the extension of an online platform developped to be a course material for pupils of 10 to 14 years old. The goal of this extension is to make the platform become available to any primary or secondary school teacher. The extension allows teachers to create their own course, to build a teaching strategy for their students and to manage their progression. This thesis details the structure of the application, educational principles that were used during the development and arguments for the integration of computer science in primary and secondary school.

The platform provided for this thesis has been successfully tested by 5 classes of students during the activities of the "Printemps des Sciences" and through PullReview that is a website providing code reviews to ensure that the code is readable, has small complexity and has no vulnerabilities. The entire code of the application is available at `https://github.com/simonhock/rsnap` and `https://github.com/simonhock/Snap--Build-Your-Own-Blocks`.

At the term of the implementation of the extension of the platform, we conclude that the platform is ready to be deployed and to be used by teachers of primary and secondary school. For that purpose, a "Teacher's guide" has been written to explain how the platform works and how to create a course using it.

Table des matières

1 Introduction **13**

1.1 Le contexte . . . 13
1.2 La problématique . . . 13
1.3 La solution proposée . . . 13
1.3.1 Les rôles . . . 14
1.3.2 La stratégie didactique . . . 14
1.3.3 Le contenu . . . 14
1.3.4 Le système de vérification des connaissances . . . 14
1.4 Plan du mémoire . . . 14
1.4.1 État des lieux et introduction des extensions . . . 14
1.4.2 Les choix pédagogiques . . . 14
1.4.3 Du côté technique . . . 15
1.4.4 Les tests réalisés . . . 15

2 État des lieux et introduction des extensions **17**

2.1 La programmation dans l'enseignement . . . 17
2.1.1 Les besoins de l'informatique dans l'enseignement . . . 17
2.1.2 Les compétences scientifiques dans la Fédération Wallonie-Bruxelles . . . 18
2.2 Les plate-formes déjà existantes . . . 20
2.2.1 Move the turtle . . . 20
2.2.2 Robot school . . . 20
2.2.3 KidsRuby . . . 21
2.2.4 Code.org . . . 22
2.3 Résumé du travail précédent . . . 23
2.3.1 La problématique . . . 23
2.3.2 La solution . . . 23
2.4 Introduction des extensions . . . 24
2.4.1 La structure du contenu proposé aux élèves . . . 24
2.4.2 Les copies de missions . . . 25
2.4.3 Les blocs d'introspections . . . 25
2.4.4 Le contenu ajouté . . . 25

2.4.5 Les rôles . . . 26
2.4.6 Les modifications de Snap ! . . . 26

3 Les choix pédagogiques 27
3.1 Les choix pédagogiques . . . 27
3.1.1 Les principes . . . 27
3.1.2 Dans RSnap . . . 29
3.2 Les notions apprises avec leurs contextes . . . 29
3.2.1 La séquentialité . . . 30
3.2.2 Les paramètres . . . 30
3.2.3 Les boucles . . . 31
3.2.4 Les boucles imbriquées . . . 31
3.2.5 Les conditions simples . . . 32

4 Du côté technique 33
4.1 Ruby on Rails . . . 33
4.1.1 Le MVC . . . 33
4.2 Le modèle de RSnap . . . 35
4.3 Les modifications de l'environnement de programmation . . . 37
4.4 La mise en place d'un serveur pour le Printemps des Sciences . . . 38

5 Les tests réalisés 41
5.1 Le Printemps des Sciences . . . 41
5.1.1 Le déroulement des activités . . . 41
5.1.2 Analyse des formulaires . . . 42
5.2 PullReview . . . 44

6 Conclusion 45
6.1 La problématique . . . 45
6.2 La solution proposée . . . 45
6.3 Les résultats . . . 46
6.4 Le bilan . . . 46
6.5 Perspectives . . . 47
6.5.1 L'import et l'export de contenu . . . 47
6.5.2 La charte graphique du LMS . . . 47
6.5.3 Un système de messagerie . . . 47
6.5.4 La protection par un mot de passe pour l'inscription aux cours . . . 47

A Statistiques sur le Printemps des Sciences. 51

B Diagrammes 52

C Le guide du professeur **54**

C.1 La structure de l'application 54

C.1.1 Les rôles 54

C.1.2 Les missions 55

C.1.3 Les chapitres 56

C.1.4 Les cours 57

C.2 La création d'une mission 58

C.2.1 Les différentes parties d'un programme 58

C.2.2 Les fonctionnalités utiles 61

Chapitre 1

Introduction

Ce chapitre d'introduction va présenter le contexte dans lequel ce mémoire est réalisé ainsi que la problématique abordée et enfin la solution proposée.

1.1 Le contexte

Aujourd'hui, l'informatique est omniprésent et les nouvelles technologies s'immiscent dans notre quotidien. Pour cette raison, technicien en informatique est l'un des métiers les plus recherchés. Vous comprendrez sans difficulté qu'il est fondamental de sensibiliser les jeunes aux joies de la programmation afin de susciter leur intérêt vers des études dans ce domaine.

1.2 La problématique

La problématique abordée dans ce mémoire est que l'enseignement obligatoire n'a pas les moyens de donner des cours d'informatique aux étudiants car il manque de professeurs qualifiés et de supports de cours. L'application proposée par Simon Claessens et Gaëtan Collart dans leur mémoire en 2013-2014 était fonctionnelle et proposait un contenu mais celui-ci était restreint et l'application ne permettait pas aux enseignants de créer leurs cours et de gérer leurs classes d'élèves. De plus, l'environnement de programmation dans lequel les élèves étaient plongés laissait beaucoup de possibilités de distraction.

1.3 La solution proposée

Les modifications de la structure de la plate-forme visent à ajouter les fonctionnalités d'un LMS (Learning Management System ou Système d'accompagnement de l'apprentissage), c'est-à-dire : la possibilité d'être un élève ou un professeur, une stratégie didactique, du contenu didactique et un système de vérification des connaissances. [10] L'ensemble du code de ce mémoire est disponible aux adresses [1] et [2].

1.3.1 Les rôles

La création de 2 rôles distincts cloisonne les autorisations des utilisateurs. Ceci permet d'avoir une application adaptée à l'utilisation de chacun. Le rôle de "professeur" permettra à un enseignant de créer du contenu tandis que le rôle d'"étudiant" permettra aux élèves d'avoir accès au contenu proposé par les professeurs.

1.3.2 La stratégie didactique

La stratégie didactique permet d'améliorer l'efficacité de l'enseignement proposé sur l'application. Elle a pour but de s'assurer que les élèves assimilent la matière enseignée par le professeur.

1.3.3 Le contenu

L'application proposée verra sa structure modifiée pour permettre aux professeurs de créer différents cours selon leurs besoins. Le contenu de cette application passera donc d'un ensemble d'exercices identiques pour tous ses utilisateurs à un ensemble de cours structurés en chapitres composés d'exercices.

1.3.4 Le système de vérification des connaissances

Un système de vérification automatique des réponses est utile pour gérer l'avancement des élèves dans les exercices et débloquer un exercice seulement si le précédent a été réussi. La correction directe par le professeur permet également un retour personnalisé ce qui améliore d'autant plus l'efficacité de l'enseignement.

1.4 Plan du mémoire

1.4.1 État des lieux et introduction des extensions

Ce chapitre développera le contexte dans lequel se tient le mémoire. Il y sera présenté un ensemble de plate-formes déjà existantes offrant un apprentissage de la programmation ainsi que l'application proposée par Simon Claessens et Gaëtan Collart dans leur mémoire en 2013-2014, ensuite les modifications apportées seront introduites.

1.4.2 Les choix pédagogiques

Les différents choix concernant la structure de l'application, la structure du contenu ainsi que l'ordre d'apprentissage des notions de programmation seront développés dans le chapitre sur les choix pédagogiques.

1.4.3 Du côté technique

Cette partie du mémoire expliquera les modifications de la structure de la plate-forme et les modifications de l'environnement de programmation faites suite aux choix pédagogiques développés dans le chapitre qui y est dédié. Ensuite, la mise en place du serveur dédié aux activités du Printemps des Sciences y sera développée.

1.4.4 Les tests réalisés

Ce dernier chapitre présente les tests réalisés au cours de l'implémentation des modifications apportées. Il y sera expliqué le déroulement des activités du Printemps des Sciences et ensuite les analyses des réponses données par les élèves à un questionnaire qui leur a été soumis à la fin de l'activité.

Chapitre 2

État des lieux et introduction des extensions

Ce chapitre va développer les motivations du mémoire qui a été réalisé en présentant tout d'abord l'état des lieux de l'apprentissage de la programmation dans l'enseignement obligatoire. Il décrira ensuite les plate-formes permettant l'apprentissage de la programmation via une programmation visuelle (programmation par blocs). Enfin viendra un résumé du travail qui a servi de base à ce mémoire suivi d'une introduction des extensions qui ont été apportées à ce travail.

2.1 La programmation dans l'enseignement

L'ère de l'informatisation a commencé et il faut que nous nous y préparions ! Les technologies avancent à grand pas et n'attendent pas les personnes qui ne les utilisent que peu. De plus en plus souvent les informations sont numérisées sous forme de documents portables, de bases de données, ... Malgré cela, il est courant que des personnes n'aient aucune idée de comment utiliser un ordinateur ou un smartphone. Il faut que chacun puisse les utiliser correctement. C'est pourquoi il est important que les compétences requises soient intégrées dans l'enseignement obligatoire.

2.1.1 Les besoins de l'informatique dans l'enseignement

D'après un rapport écrit conjointement par l'ACM Europe Working Group on Informatics Education et Informatics Europe [3], les compétences de base, "digital literacy" ou "la culture du numérique" en français, ainsi que les règles de bonne utilisation d'un ordinateur devraient être acquises à l'âge de 12 ans. Ces compétences et règles de base regroupent aussi bien la dactylographie que la compréhension des unités de mesures, l'organisation de fichiers dans des répertoires et la communication via emails, réseaux sociaux ou forums. En ce qui concerne les règles d'efficience, d'éthique et de sécurité, elles reprennent la capacité à faire la différence entre la réalité et le virtuel, la reconnaissance de fraudes et des problèmes de sécurité, la capacité à rester critique par rapport aux informations tirées d'Internet et encore bien d'autres. La culture du

numérique n'est pas suffisante, elle ne regroupe que des compétences d'utilisation de l'ordinateur. Elle doit être accompagnée d'une autre matière reprenant la compréhension du fonctionnement de l'ordinateur appelée "Computer Science", "Computing Science" ou "Informatics" que nous traduirons ici par "Sciences Informatiques".

De plus, d'après la Commission Européenne [5] , il est créé 120.000 emplois par an dans le secteur de l'ICT (Information and communications technology ou TIC en français pour Technologies de l'information et de la communication) en Europe. Cette augmentation de l'offre d'emplois dans le secteur doit être suivie d'une augmentation conjointe de personnes ayant une formation suffisante dans ce domaine, sans quoi, d'ici 2020 il y aura une pénurie de près de 900.000 personnes qualifiées dans le secteur.

L'intégration des sciences informatiques dans l'enseignement pourrait être extrêmement favorable pour les besoins grandissant d'informaticiens. De plus, les compétences apportées par cette branche pourraient être bénéfiques pour les autres branches scientifiques et apporter une façon de raisonner plus rigoureuse. "Computational Thinking" est un terme créé par Jeannette Wing qui signifie "Le raisonnement informatique". Ce raisonnement regroupe des techniques de résolution de problèmes et des pratiques intellectuelles. Les techniques de résolution de problèmes vont de la représentation abstraite de données à la généralisation de solutions en passant par l'identification, l'analyse et l'implémentation de solutions. Les pratiques intellectuelles, quant à elles, regroupent la tolérance de l'ambiguïté, la capacité à résoudre des problèmes ouverts (où plusieurs réponses sont possibles), la capacité à réconcilier les aspects techniques et humains des programmes.

2.1.2 Les compétences scientifiques dans la Fédération Wallonie-Bruxelles

Cette partie va analyser les compétences scientifiques définies par le socle de compétences de la Fédération Wallonie-Bruxelles et les mettre en vis-à-vis avec les compétences que l'apprentissage de la programmation peut apporter.

La Fédération Wallonie-Bruxelles reconnaît trois étapes dans l'Enseignement fondamental et le premier degré de l'Enseignement secondaire : la première regroupe les deux premières années de primaire, la deuxième regroupe les quatre dernières années de primaire et la troisième regroupe les deux premières années de secondaire. Nous allons prendre en considération les 2 dernières étapes qui englobent l'âge ciblé par la plate-forme, c'est-à-dire 10 à 14 ans.

Les compétences comme définies dans le socle de compétences de la Fédération Wallonie-Bruxelles [7] entrent dans 3 niveaux d'apprentissage : la sensibilisation à l'exercice de celle-ci, la certification de cette compétence en fin d'étape ou l'entretien de la compétence.

Rencontrer et appréhender une réalité complexe

Compétence de la FWB	**Compétence similaire apportée par l'informatique**
Faire émerger une énigme à résoudre.	Analyser un énoncé et en comprendre les subtilités.
Identifier des indices et dégager des pistes de recherches propres à la situation.	Réécrire les fonctionnalités décrites dans un énoncé d'une façon non-ambiguë et écrire différents pseudo-codes.
Confronter les pistes perçues, préciser des critères de sélection des pistes et sélectionner selon les critères.	Comparer les différents pseudo-codes en termes de complexité grâce à des connaissances en algorithmique

Investiguer des pistes de recherches

Compétence de la FWB	**Compétence similaire apportée par l'informatique**
Récolter des informations par la recherche expérimentale, l'observation et la mesure.	Étoffer ses connaissances en programmation en fonctionnant par essai-erreur, faire de l'implémentation et du débugging.
Récolter des informations par la recherche documentaire et la consultation de personnes ressources.	Récolter des informations sur le web, garder un regard critique sur ces informations et faire la différence entre la réutilisation et le plagiat ainsi que faire des travaux par groupe de 2, s'entraider et au besoin, faire appel à l'enseignant.

Structurer les résultats, les communiquer, les valider et les synthétiser

Compétence de la FWB	**Compétence similaire apportée par l'informatique**
Rassembler et organiser des informations sous une forme qui favorise la compréhension et la communication	Savoir organiser des fichiers et dossiers, créer un document de rapport, une présentation, structurer son raisonnement et l'expliciter par un diagramme clair reprenant les principales étapes
S'interroger à propos des résultats d'une recherche et élaborer une synthèse des nouvelles connaissances.	Analyser les différentes erreurs faites lors des implémentations par essais-erreurs, en tirer des conclusions et évaluer leur pertinence afin d'acquérir de nouvelles connaissances.

Ces tableaux montrent une similarité entre les compétences apportées par l'informatique et celles fixées par la Fédération Wallonie-Bruxelles. Le fait que ces compétences soient similaires pourrait être un atout pour l'intégration de la programmation dans le cursus des études primaires et secondaires. En effet, la similarité entre des compétences dans différents domaines permet une compréhension des nouvelles informations plus rapidement. Par exemple, un physicien aura plus de facilités à comprendre de la biologie comparé à un psychologue puisque le physicien a déjà les compétences scientifiques de base.

2.2 Les plate-formes déjà existantes

Cette partie du chapitre va décrire différentes plate-formes et applications déjà existantes qui proposent un apprentissage de l'informatique, leurs avantages et inconvénients. Enfin, la plate-forme utilisée comme base de ce mémoire avant les ajouts des extensions sera décrite.

2.2.1 Move the turtle

Move The Turtle est une application iPad pour les enfants de 5 ans et plus. L'application vise l'apprentissage de la programmation pour ses utilisateurs. L'application se compose de deux écrans principaux. Le premier étant l'écran de scène et le deuxième celui de programmation. L'écran de scène est celui qui contient l'ensemble des animations qui seront écrites par l'utilisateur dans l'écran de programmation. Ce dernier contient une liste de commandes qui est à l'origine vide et que l'utilisateur doit remplir avec les actions qu'il veut que la tortue exécute. Ce dernier écran pourrait être considéré comme le "code" que l'utilisateur crée. Le contenu de l'application est un ensemble de puzzles que l'utilisateur doit résoudre en créant la bonne suite d'actions dans l'écran de programme.

Ses points forts :

— l'ensemble de la charte graphique de l'application est cohérente
— elle permet un apprentissage complet des notions de base de la programmation, comme les conditions, les variables, les boucles, les procédures

Ses points faibles :

— l'application n'est disponible que sur iPhone/iPad
— elle est payante
— elle n'est disponible qu'en anglais

2.2.2 Robot school

Robot school est une application pour android, iPhone/iPad et mac contenant 45 niveaux qui visent l'apprentissage de la logique de programmation. Elle convient pour les enfants de plus de 7 ans. Le but des utilisateurs est cette fois-ci de suivre un parcours. Chaque parcours est réussi quand l'utilisateur trouve la bonne suite d'actions pour que le robot arrive à son but. Pour cela, l'utilisateur a accès à des blocs qui représentent des actions (bouger, prendre un objet,...),

FIGURE 2.1 – Les écrans de scène et de programmation de l'application "Move The Turtle"

des senseurs ("quelle est la couleur de .. ?",...) et des opérations logiques (Si, boucles,..). La programmation que doit faire l'utilisateur se fait par déplacement des blocs dans le banc de travail par cliquer-tirer. Ce banc de travail est une zone dans le bas de l'écran dans lequel est dessiné l'arbre d'exécution du programme que l'élève construit.

Ses points forts :

— l'application est disponible sur tous types de tablettes et smartphones

— son utilisation est simple

Ses points faibles :

— l'application est entièrement en anglais

— elle est payante

FIGURE 2.2 – Robot School

2.2.3 KidsRuby

KidsRuby est une application pour Windows, linux et mac dédiée aux enfants. L'interface utilisateur est divisée horizontalement en deux parties. La partie de gauche est un éditeur de texte dans lequel l'utilisateur peut écrire son code Ruby. La partie de droite est une interface contenant 3 pages qui contiennent respectivement l'aide, la sortie et la "tortue". L'aide contient

tout le tutoriel pour apprendre la programmation aux utilisateurs, la sortie est la sortie standard et la tortue est un écran qui est utilisé dans certaines parties du tutoriel. Le tutoriel contient des explications sur le fonctionnement de l'application, ce qu'est la programmation et ensuite un tutoriel de programmation de base. La page d'aide propose beaucoup de tutoriels pour créer différents jeux.

Ses points forts :

— elle contient un tutoriel sur les bases du langage de programmation Ruby

— l'application comprend des tutoriels pour différents jeux

Ses points faibles :

— elle n'est pas complètement traduite en français

— l'accompagnement de l'apprentissage des notions de programmation est très basique

— les utilisateurs doivent écrire du code Ruby basé sur l'Anglais, ce qui n'est pas intuitif pour des utilisateurs non-anglophones de moins de 14 ans.

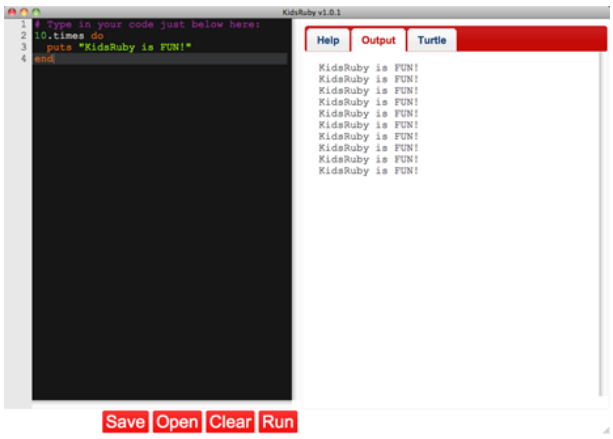


FIGURE 2.3 – Interface de KidsRuby

2.2.4 Code.org

Code.org est une plate-forme web utilisable sur les ordinateurs, tablettes et smartphones. Elle propose des cours débutants qui ciblent différentes tranches d'âge (4, 6, 8 et 10 ans et plus) ainsi que "l'heure de code" qui est un cours générique pour tout âge qui permet l'apprentissage des principes de base de la programmation. Cette plate-forme permet de créer un compte et donc d'avancer à son propre rythme dans les missions en étant n'importe où dans le monde car elle enregistre l'avancement de l'élève. La programmation que les élèves utilisent est une programmation "par bloc" en "drag and drop".

Ses points forts :

— elle est utilisable à tout âge

— les explications pour apprendre la programmation aux élèves couvrent les notions de base de la programmation

— les élèves ont accès à une aide lors de la réalisation des missions

— elle utilise des vidéos pour expliquer toutes les notions et missions

Ses points faibles :

— les descriptions des missions ne sont pas toujours traduites en français

— les vidéos d'explications des notions sont uniquement en anglais, mais sous-titrées en français, ce qui n'est pas évident pour les élèves ne sachant pas facilement lire.

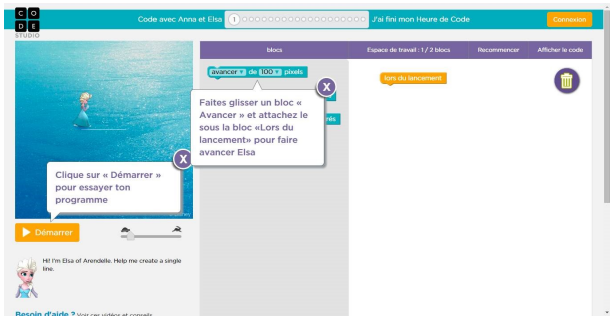


FIGURE 2.4 – Interface de KidsRuby

2.3 Résumé du travail précédent

Ce mémoire est l'extension du mémoire réalisé pendant l'année académique 2013-2014 par Gaëtan Collart et Simon Claessens. Nous allons d'abord citer la problématique et ensuite détailler le travail qui a été réalisé.

2.3.1 La problématique

La problématique qui avait été définie était la suivante : "Il y a un manque de soutien à l'apprentissage de la programmation pour les jeunes francophones. Il n'y a pas ou peu de moyens pédagogiques actuellement. Ce manque de moyens n'incite pas ceux qui souhaiteraient prendre le train en marche et commencer à enseigner la programmation."[6]. La problématique citée ci-dessus souligne le manque d'enseignants et de supports de cours pour l'apprentissage de la programmation.

2.3.2 La solution

La plate-forme proposée par Simon Claessens et Gaëtan Collart avait pour but de répondre à cette demande. La plate-forme a été créée pour fournir un environnement pour accompagner l'apprentissage de la programmation aux 10 à 14 ans. Son utilisation optimale serait de servir de support de cours à un professeur qui explique la matière proposée aux élèves et qui répond aux questions des élèves une fois qu'ils ont commencé à l'utiliser. L'application qui a été développée par les deux étudiants était structurée comme décrit ci-dessous.

L'application contient 2 parties distinctes. La première appelée "RSnap" sert à l'accompagnement de l'élève dans son apprentissage. Le seconde partie appelée "Snap !" est l'environnement de programmation dans lequel l'élève pourra travailler.

La partie RSnap de l'application contient tout le côté pédagogique. C'est la première partie de l'application que l'utilisateur verra. Lorsqu'il arrive sur la page d'accueil, il peut se créer un compte et ensuite aller au cœur de l'application qui est la liste des missions. L'utilisateur peut résoudre les missions une à une. Une mission contient un titre, une petite description qui est affichée dans la liste des missions, une description, une vidéo YouTube et un "programme de mission".

La description de la mission et la vidéo contiennent toutes les explications requises pour la réalisation de la mission. Le "programme de mission" est le code source utilisé pour la partie Snap ! de l'application. Quand l'élève est sur la description d'une mission, il a accès à un bouton "Réaliser la mission" qui l'envoie dans la partie Snap ! de l'application.

La partie Snap ! est l'environnement de programmation dans lequel l'élève réalise chaque mission. Cette partie est séparée en différentes zones : la zone de blocs, la zone de scripts, la scène et la zone des objets.

La zone de blocs contient tous les "blocs" de base que les élèves doivent utiliser pour réaliser la mission, la zone de scripts est l'endroit où l'élève doit créer son programme à partir des blocs de base. C'est dans cette partie qu'il créera ses scripts, ce sont des ensembles de blocs que l'utilisateur attache les uns aux autres. La scène peut être comparée à la sortie standard d'un programme, c'est là que l'élève verra les effets de son script. La zone des objets est la partie de l'écran où sont affichés tous les personnages existants (Sprites en anglais) ainsi que la scène. En fonction du Sprite sélectionné dans cette zone, la partie script affichera le script correspondant au personnage. Chaque bloc qui est contenu dans un script d'un personnage lui est lié. C'est-à-dire qu'un bloc "Avancer" fera avancer le personnage auquel il est lié.

Étant donné que chaque mission entre dans un contexte, il est donc logique que lorsque l'élève clique sur "Réaliser la mission", il arrive dans l'environnement de programmation qui reprend le contexte de cette mission, avec des personnages déjà définis et éventuellement des scripts déjà présents. C'est pour cela que chaque mission a un code source. Ce code source permet, au démarrage de la partie Snap, de créer chaque personnage avec leurs scripts, leurs images, ...

2.4 Introduction des extensions

Cette dernière partie du chapitre va introduire les changements majeurs qui ont été faits à l'application décrite ci-dessus. L'intégralité de ces changements ont été faits dans le but d'améliorer l'expérience des utilisateurs ainsi que pour faciliter la gestion de l'avancement des élèves et du contenu de l'application.

2.4.1 La structure du contenu proposé aux élèves

L'organisation des missions réalisées par les élèves a été changée d'une simple liste ordonnée à un ensemble de cours comprenant une liste ordonnée de chapitres composés, eux-mêmes, de missions. Le premier avantage de cette organisation est que l'on peut créer des ensembles de

missions ayant le même contexte que l'on introduit dans la description du chapitre. Le fait de regrouper les missions en chapitres permet aussi de faire en sorte que chaque chapitre corresponde à l'apprentissage d'une notion de programmation et ensuite que chaque mission du chapitre soit de plus en plus complexe tout en apprenant à l'élève cette notion. Un autre avantage est qu'au sein d'un même chapitre on peut faire des missions extrêmement basiques pour commencer, et ensuite complexifier les missions, tout en restant dans le même thème, pour enfin arriver à des missions qui demandent une utilisation des notions d'autres chapitres déjà réalisés par l'élève. Enfin, le fait d'organiser le contenu de l'application en cours permettra aux professeurs de suivre la structure de l'Enseignement et aux élèves de s'inscrire à un ou plusieurs de ces cours.

2.4.2 Les copies de missions

Du côté de l'administration, nous avons maintenant laissé la possibilité de créer une mission à partir d'une autre. Lors de la création d'une mission, on peut choisir de créer son programme, c'est-à-dire le contexte de l'environnement de programmation, à partir du programme vierge ou à partir du programme d'une autre mission. Cette fonction permet à un professeur de ne pas devoir recréer chaque programme à partir de zéro mais de simplement changer quelques propriétés de celui-ci pour qu'il corresponde à la nouvelle mission.

2.4.3 Les blocs d'introspections

Dans la partie Snap ! de l'application, de nouveaux blocs ont été créés dans le but de permettre aux professeurs de tester les réponses des élèves. Les blocs principaux qui ont été ajoutés sont deux blocs d'en-tête, c'est-à-dire des blocs qui sont activés lors de certains événements et sur lesquels on peut attacher d'autres blocs qui seront activés eux aussi lors de ce même événement. Le premier bloc est un bloc de démarrage, qui est activé lors du lancement du programme de l'élève, c'est à celui-ci que l'élève devra rajouter des blocs pour créer son programme. Ce bloc est très utile car une fois que tous les blocs de l'élève ont été exécutés, il créera un nouvel événement qui activera le deuxième bloc introduit. Ce deuxième bloc, comme le premier, est un bloc activé lors d'un événement. Ce dernier est donc la "fin" du programme de l'élève. Ce bloc permet au professeur de tester les effets du programme de l'élève. Par exemple : tester si le personnage est arrivé au bon endroit, tester si le personnage a dessiné ce qu'il fallait faire, ...

Dans la continuité du raisonnement, un autre bloc a été créé pour permettre à la partie "RSnap" de la plate-forme de marquer le programme créé par l'étudiant comme étant correct. Ce bloc permet au professeur de créer une vérification automatique du programme de l'étudiant et ainsi réguler l'avancement de celui-ci.

2.4.4 Le contenu ajouté

Du côté du contenu ajouté, l'application a été complétée avec différents chapitres et différentes missions. Les contextes de ces chapitres ont été inventés et inspirés des différentes plate-formes

et des missions créées dans la version précédente de RSnap.

Le ReadMe d'installation d'un serveur a été complété pour être plus précis et à jour par rapport aux mises à jour des programmes tiers utilisés par l'application. De plus, un script d'installation pour installation Debian a été implémenté afin de permettre à n'importe qui de mettre en place un serveur pour RSnap.

Enfin, l'ordre d'apprentissage des différentes notions a été défini comme ceci :

1. la séquentialité
2. les paramètres et constantes
3. les boucles avec nombre d'itérations
4. l'imbrication de boucles les unes dans les autres
5. les conditions
6. les variables
7. les boucles avec conditions
8. la logique propositionnelle

2.4.5 Les rôles

L'avant dernière modification a été de permettre aux professeurs de se créer un compte avec les autorisations adéquates. Lors de la création de son compte, l'utilisateur peut cocher "Je suis un professeur" pour que le compte créé soit de type professeur. Une fois le compte créé et l'utilisateur connecté, il peut créer des missions, des chapitres et des cours destinés aux étudiants ayant créé leur compte sans l'option "Je suis un professeur".

2.4.6 Les modifications de Snap !

Snap ! est l'environnement dans lequel les élèves vont créer leurs programmes. Il est donc important que celui-ci ne contienne pas de distraction. La première modification de Snap ! a été le retrait de tous les boutons n'ayant pas d'utilité pour l'apprentissage de la programmation. La seconde modification fût d'adapter les traductions aux autres modifications. Enfin, quelques bugs ont été trouvés et corrigés.

Chapitre 3

Les choix pédagogiques

Dans ce troisième chapitre, nous allons expliquer les choix pédagogiques que nous avons faits dans le développement de RSnap et nous expliquerons ensuite les histoires qui ont été créées comme contexte d'apprentissage des notions de base de la programmation.

3.1 Les choix pédagogiques

Être professeur ne s'improvise pas ! Pour être sûr que toutes les informations données soient bien comprises et que les élèves progressent correctement, il faut s'assurer de garder leur attention. C'est pourquoi il est impératif de suivre certains principes. Nous allons en expliquer quelques uns tels que la cohérence, la non-redondance, le pré-entraînement et d'autres.

3.1.1 Les principes

La cohérence

Le premier principe est la cohérence. C'est un principe selon lequel il faut se limiter aux informations essentielles car les informations qui ne le sont pas risquent de distraire les élèves du but principal du cours.

Le signalement

Le signalement est le principe selon lequel la mise en avant des informations permet une intégration plus rapide. Donc le fait de mettre les informations importantes en relief (**gras**, *italique*, couleur,...) favorisera l'apprentissage de l'élève.

La non-redondance

Un principe auquel il faut faire attention pour ne pas perdre l'attention est la non-redondance. Celui-ci dit que le fait de donner une information trop de fois peut être néfaste à l'attention de l'élève. Au moment où l'élève se rendra compte que les informations sont répétées, il ne prendra

plus la peine d'y faire attention en se disant que les informations importantes seront reprises plus loin dans le cours, ce qui n'est pas toujours le cas !

La contiguïté spatiale et temporelle

En accord avec le principe précédent, la contiguïté spatiale permet une intégration et une compréhension plus rapide et efficace. La contiguïté spatiale d'informations est le fait qu'elles soient à proximité spatiale l'une de l'autre, elle permet de faire des liens entre ces informations plus facilement et donc de mieux les retenir. Par exemple, une image qui est l'agrandissement d'une autre image sera mieux interprétée lorsqu'elle sera à côté de l'autre. Il en est de même pour la contiguïté temporelle, le principe est similaire que pour la distance entre les informations et le temps qui s'est écoulé entre celles-ci.

La segmentation

Le fait qu'un cours soit structuré et divisé en parties permet à l'élève de savoir pourquoi chaque partie du cours est importante mais aussi d'avoir une meilleure compréhension de chacune des parties ainsi que de l'ensemble du cours. La division du cours en sous-parties permet donc d'assimiler les points essentiels de la matière. Ce principe s'appelle la segmentation.

Le pré-entraînement

Le pré-entraînement, le fait de donner quelques informations générales sur ce qui sera vu dans les parties du cours, permet à l'élève de déjà créer quelques connexions entre ces parties. Cette préparation facilitera donc l'intégration de données et les connexions entre elles.

Le mode de diffusion

Le mode de diffusion des données permet, lui aussi, de faciliter l'apprentissage. Par exemple, un schéma sera plus parlant qu'un long discours. Le fait d'utiliser plusieurs canaux (image, oral, ...) de manière complémentaire permettra d'intégrer des informations plus aisément et donc de faciliter la compréhension des élèves.

L'intégration multimédia

Un principe assez ressemblant à ce dernier est l'intégration multimédia. Ce dernier nous dit qu'une combinaison de mots et d'images sera plus efficace qu'un long texte sans image. L'illustration des propos permettra aux personnes ayant une mémoire visuelle de retenir les images et ensuite d'y associer des mots.

La personnalisation

Le dernier principe facilitant l'apprentissage dont nous parlerons est la personnalisation. Le fait de parler directement à l'élève ou au groupe d'élève en les tutoyant ou en les vouvoyant

permet qu'ils se sentent impliqués, ce qui a pour effet de les garder attentifs. [9]

3.1.2 Dans RSnap

Nous avons choisi pour ce mémoire, de rendre la plate-forme plus agréable à utiliser et dans une structure la plus favorable à l'apprentissage des notions de base de la programmation. Cette partie développera les choix qui ont été faits en ce qui concerne la division du contenu en différents cours.

Il a été décidé que l'ensemble des notions apprises aux élèves seraient enseignées au cours de différents chapitres et que chacun de ceux-ci enseigneraient à l'élève une nouvelle notion de programmation ou constitueraient un approfondissement d'une notion déjà enseignée. Ces chapitres peuvent être assimilés à ceux d'un cours de n'importe quelle autre matière du secondaire. Chaque chapitre conçu dans le cadre de ce mémoire a été développé pour rentrer dans un canevas. Les chapitres ont leur propre contexte, c'est-à-dire une petite histoire pour tenir l'attention de l'élève et donner un but ludique à chaque travail fait par l'élève. Ensuite, chaque chapitre est divisé en plusieurs missions. Comme écrit ci-dessus, un chapitre est fait pour apprendre une nouvelle notion à l'élève ou pour approfondir une notion déjà apprise. Chaque chapitre commence par des missions simples en utilisant des notions déjà acquises, vient ensuite la mission d'introduction de la nouvelle notion à assimiler suivie d'autres missions plus complexes dans le but d'approfondir la notion. Ce schéma correspond à l'approche utilisée dans le cadre des cours donnés par l'ADEPS [4]

Le canevas de chaque chapitre peut être comparé aux différentes étapes de l'apprentissage des notions scientifiques dans l'Enseignement de la Fédération Wallonie-Bruxelles(Voir section 2.1.2). Chaque chapitre introduira une notion, ce qui correspondra à l'étape d'initiation. À la fin de chaque chapitre, l'apprentissage de la notion sera testé sur des missions plus compliquées, ce qui fera office de certification. Enfin, pour suivre la structure de l'Enseignement, les chapitres comme expliqués ci-dessus font partie de différents cours auxquels les élèves peuvent s'inscrire. Ces élèves peuvent donc suivre plusieurs cours composés de chapitres enseignant différentes notions de programmation via un ensemble de missions comparables à des exercices. Un cours proposant un apprentissage des notions de base de la programmation a été créé.

Pour rappel, chaque chapitre commence par un petit texte d'introduction agrémenté d'images et auquel on peut ajouter une vidéo. Cette possibilité d'ajout d'images ou de vidéo rejoint le principe d'intégration multimédia expliqué dans le point précédent(Voir section 3.1.1). La segmentation, elle aussi expliquée dans le point précédent, a été utilisée dans la découpe de l'apprentissage en chapitres et missions distinctes.

3.2 Les notions apprises avec leurs contextes

Un cours a été créé pour enseigner les notions de base de la programmation aux élèves. Cette partie expliquera l'ordre d'apprentissage des notions tout en les reliant à un contexte propice à

la motivation des élèves.

3.2.1 La séquentialité

La séquentialité est la première notion à apprendre quand on veut commencer la programmation. La séquentialité est le fait que, dans un code, toutes les actions, les méthodes, sont faites dans l'ordre dans lequel elles sont écrites. Dans l'application, les actions sont des blocs qui sont attachés les uns en dessous des autres. Il a donc fallu apprendre aux élèves que les blocs sont "activés" ou "exécutés" les uns après les autres en allant du bloc le plus au dessus au bloc le plus en dessous. C'est sur ce principe que reposeront toutes les notions suivantes car les blocs que les élèves utiliseront seront toujours attachés les uns aux autres et exécutés dans ce même ordre.

Pour cette première notion, le contexte utilisé est celui d'une souris qui doit parcourir des poutres pour atteindre un morceau de fromage. La première mission que l'élève doit réaliser est simplement de faire avancer la souris en ligne droite jusqu'au fromage. La difficulté est que la souris ne peut pas sortir des cases autorisées. Avec des termes du contexte : elle est sur une poutre et ne doit pas en tomber.

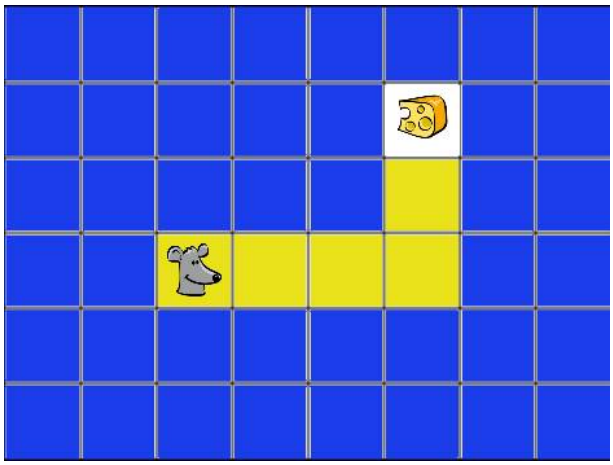


FIGURE 3.1 – Une des scènes de la souris

3.2.2 Les paramètres

La seconde notion importante est le fait que chaque action peut prendre des paramètres et produira un résultat différent en fonction de ce ou ces paramètres. Le contexte utilisé pour le chapitre qui apprend cette notion est celui d'une voiture que l'élève "conduit". Les blocs "avancer" ont maintenant un paramètre pour dire de combien la voiture doit avancer. Cela montre aux élèves que les paramètres sont intéressants pour réduire le nombre d'instructions (de bloc, dans la plate-forme). La voiture doit faire tout un circuit pour arriver à la ligne d'arrivée sans sortir de ce circuit sinon la voiture explose. Les circuits sont plus ou moins comparables aux parcours de la souris sur des poutres mais vu que les élèves ont accès à des blocs faisant avancer de plusieurs cases, le travail de l'élève sera de trouver les bons paramètres.

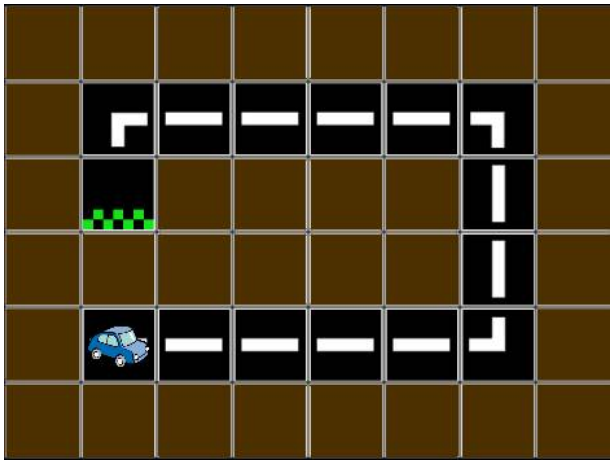


FIGURE 3.2 – Une des scènes de la voiture

3.2.3 Les boucles

L'apprentissage suivant est celui des boucles. Les boucles sont très importantes en programmation aussi bien quand on sait qu'on doit répéter un certain nombre de fois une action mais surtout l'inconnu du nombre de fois qu'il faudra la répéter. Le troisième chapitre apprend aux élèves comment utiliser une simple boucle en leur faisant déplacer un escargot sur un plateau blanc. Cet escargot laisse une traînée de couleur derrière lui et l'élève doit lui faire dessiner toutes sortes de formes en allant d'une simple ligne droite à un cercle en passant par le dessin d'un triangle et d'autres polygones.

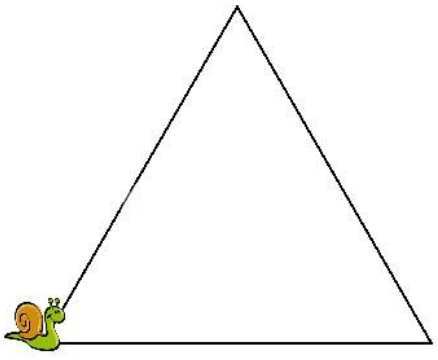


FIGURE 3.3 – Une des scènes de l'escargot

3.2.4 Les boucles imbriquées

Le chapitre sur les boucles imbriquées va montrer aux élèves qu'ils peuvent utiliser des boucles au sein même de boucles. L'histoire utilisée dans ce chapitre est la même que dans le chapitre précédent, c'est-à-dire un escargot qui dessine. Le chapitre commence par des missions où l'élève doit faire avancer l'escargot en alternant "avancer" et "avancer sans dessiner" pour dessiner une

ligne droite en pointillés et ensuite faire un angle droit tout en faisant des pointillés et enfin, l'élève doit utiliser une seconde boucle pour faire un carré tout en pointillés.

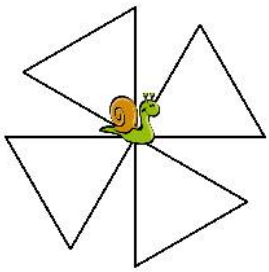


FIGURE 3.4 – Une des scènes de l'escargot pour les boucles imbriquées

3.2.5 Les conditions simples

Le dernier chapitre qui a été mis en place est celui sur les conditions simples. Les élèves doivent guider un vieux chercheur d'or. Le chercheur d'or doit se déplacer dans une grotte et ramasser les pierres précieuses qu'il trouve. L'élève a accès à un bloc "ramasser la pierre précieuse" et doit faire attention de ne l'utiliser que si le chercheur est réellement sur une pierre sinon la mission est ratée, ce qui oblige les élèves à utiliser la fonction "si".

FIGURE 3.5 – Une des scènes du chercheur d'or avec et sans les portions de terre

Chapitre 4

Du côté technique

Notre solution proposée implique que la plate-forme de base fournie par Simon Claessens et Gaëtan Collart soit modifiée pour y ajouter les fonctionnalités du système d'accompagnement de l'apprentissage (LMS). Ce chapitre fera une brève introduction de Ruby on Rails et de son fonctionnement. Ensuite le modèle utilisé pour l'implémentation de la plate-forme sera détaillé. Enfin, les modifications de l'environnement de programmation (Snap !) seront développées.

4.1 Ruby on Rails

"Ruby on Rails® est un framework web open-source qui est optimisé pour le bonheur du programmeur et une productivité constante." [8]. Un framework est une structure permettant la création d'applications. Ruby on Rails est le framework avec lequel RSnap a été implémenté. Le langage utilisé est le Ruby qui est un langage de programmation visant autant la productivité que le fait d'avoir un code lisible et compréhensible.

4.1.1 Le MVC

Ruby on Rails utilise la structure de "Model-View-Controller" (MVC). Cette structure permet de compartimenter les différentes parties de code en fonction de leur rôle.

Le modèle

Le *modèle* est la représentation abstraite de ce qu'on veut garder en mémoire. Dans ce cas, le modèle contient entre-autres le modèle de l'utilisateur (User), de mission (Mission), de chapitre (Chapter) qui seront créés et utilisés dans l'application. Chaque modèle définit les caractéristiques de ce qu'il est sensé représenter. Par exemple, un utilisateur (User) aura un prénom, un nom (first_name, last_name) et une adresse email (mail) alors qu'une mission aura un titre (title) et une description (description). Le *modèle* est relié à la base de données via une interface qu'on appelle ORM, abréviation de "Object Relational Mapping" qui veut dire "Cartographie

Objet-relationnel". Cette interface permettra dans le code de créer un objet et de le sauver directement en base de données via une simple fonction sans avoir besoin d'écrire de code SQL pour communiquer avec la base de données (voir Listing 4.1).

Listing 4.1– Création d'une mission et sauvegarde en base de données

```
mission = Mission.new(:title       => "Titre de la mission",
                      :description => "Description de la mission")
mission.save
```

La vue

La partie *Vue* du MVC s'occupe de gérer la partie visuelle de l'application. Elle va transformer les informations en page Internet. Le Listing 4.2 montre le code HAML d'une mission qui sera transformé en HTML. Le code HTML est le langage essentiellement utilisé pour les pages web.

Listing 4.2– Code de la vue d'une mission

```
%h1= @title
-if @mission.youtube?
  %iframe#ytplayer.center-block{:type=>"text/html",
                                :width=>854,
                                :height=>480,
                                :src=>"https://www.youtube.com/embed/
                                #{@mission.youtube}?
                                autoplay=1&hd=1&vq=hd720",
                                :frameborder=>0}
%p
  = raw @mission.description
%p
  =button "Realiser cette mission", new_initialization_program_mission_path(
                                    @course.nil? ? {:mission_id=>@mission} :
                                    {:mission_id=>@mission,
                                    :chapter_id=>@chapter,
                                    :course_id=>@course}),
                                    :type=>:success, :id=>"mission-program"
%p
  -if user_signed_in? and (current_user.try(:has_role?,:admin)
                           or @mission.teacher == current_user)
    =button "Code source", @mission.source_code.url, :type=>:primary
    =button "modifier", edit_mission_path(@mission), :type=>:success
    =button "modifier le programme", mission_program_path(@mission),
                                     :type=>:success
```

Ce code est traduit en html avec les informations d'une mission pour donner une page web comme la figure 4.1.

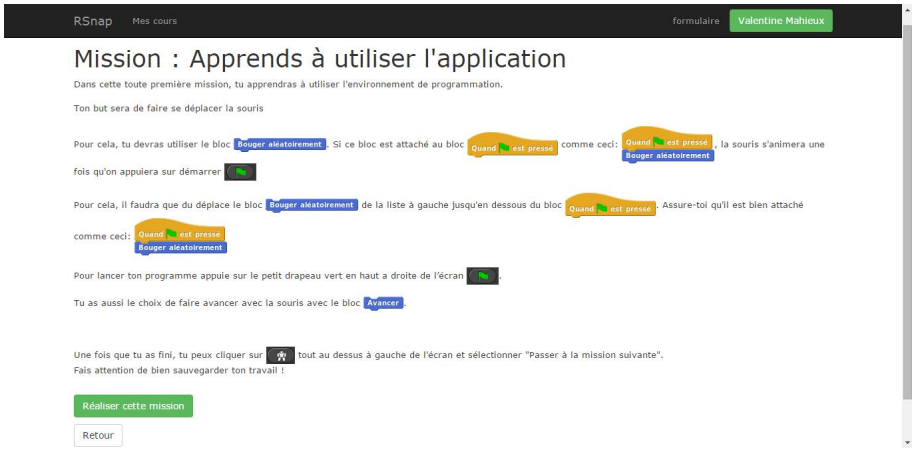


FIGURE 4.1 – Description d'une mission

La partie Controller

La partie *Controller* est le cerveau du MVC. Cette partie se charge de récupérer les requêtes HTTP envoyées par l'explorateur internet, de modifier l'un ou l'autre *Modèle* (utilisateur, mission, ...), de récupérer les informations demandées, de traduire les *Vues* avec ces informations et ensuite les renvoyer l'explorateur internet. La figure 4.2 nous montre l'architecture avec les liens entre chacune de ses composantes.

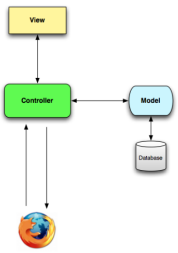


FIGURE 4.2 – Modèle du MVC de Rails

4.2 Le modèle de RSnap

Le modèle utilisé dans RSnap a été modifié pour y intégrer les données requises pour les fonctionnalités ajoutées. L'annexe B contient les diagrammes du modèle de RSnap. La figure B.1 nous montre le modèle avant les modifications apportées par ce mémoire et la figure B.2 montre le modèle actuel de RSnap.

Les chapitres

Le premier ajout au modèle fut le modèle de chapitre (Chapter) qui est composé, comme celui d'une mission, d'un titre et d'une description. Le second a été le manifeste chapitre-mission (ChapterMissionManifest) qui permet de faire le lien entre les missions et les chapitres tout en faisant en sorte que les missions d'un chapitre soit ordonnées. Ces manifestes chapitre-mission nous permettent de donner un ordre dans les missions d'un chapitre et de savoir quelle est la prochaine mission que l'élève doit réaliser.

Les cours

Les deux modifications suivantes sont semblables : l'ajout du modèle de cours (Course) et de manifeste cours-chapitre (CourseChapterManifest). La différence à ce niveau est qu'un cours n'a qu'une simple description (affichée dans la liste des cours) alors que les chapitres et les missions ont des descriptions longues destinées à enseigner les notions aux élèves. Le modèle de cours a été créé pour permettre aux élèves d'avoir accès à plusieurs cursus en même temps.

Les rôles

Ensuite, le modèle de rôle a été retiré pour laisser place aux modèles de professeur (Teacher), d'élève (Student) et d'administrateur (Admin). Ces trois modèles ont été créés via une STI (Single Table Inheritence) qui permet d'avoir un héritage de modèle tout en ne gardant qu'une seule table en base de données mais en ajoutant un attribut "type". Le listing 4.3 nous montre un exemple simple de STI en action.

Listing 4.3– Exemple simple de STI

```
class User < ActiveRecord::Base; end
class Teacher < User; end
class Student < User; end
class Admin < Teacher; end

User.all # => returns all users, teachers, students and admins
Teacher.all # => returns all teachers and admins
```

Les programmes

Le modèle de programme s'est vu modifié afin de permettre plusieurs étapes de réalisation du programme. Les programmes rédigés par les élèves peuvent être dans 4 états différents :

- En cours de réalisation
- En attente de correction
- Corrigé
- Correct

Le professeur ayant créé la mission peut choisir de rendre la validation du programme automatique ou non-automatique. Si le professeur choisit la version non-automatique, l'élève devra attendre que le professeur aille vérifier son programme et le note comme correct pour qu'il puisse passer à la mission suivante. Le professeur a aussi le choix de mettre le programme corrigé et incorrect dans l'état "corrigé" pour ajouter un commentaire au programme de l'élève sans lui permettre de passer à la mission suivante. L'élève ayant une mission actuellement corrigée aura

un message sur le haut de sa page lui montrant qu'il a une mission corrigée par un professeur et qu'il peut la refaire.

Le contenu et les professeurs

Enfin, les modèles de cours, de chapitre et de mission se sont vu attribuer une modification pour permettre d'appartenir à un professeur : l'ajout d'une référence vers le professeur l'ayant créé. Étant donné qu'un administrateur est aussi un professeur, une mission (resp. chapitre, cours) peut lui appartenir.

4.3 Les modifications de l'environnement de programmation

L'environnement de programmation, c'est-à-dire Snap !, a vu quelques modifications à son tour pour apporter les nouvelles fonctionnalités à l'application. Pour rappel, l'écran de Snap ! (voir figure 4.3) est divisé en 4 parties :

1. La zone de blocs qui contient les blocs qui peuvent être utilisés
2. La zone de scripts dans laquelle l'élève créera son programme grâce aux blocs disponibles
3. La scène dans laquelle l'élève pourra voir l'animation créée par son programme
4. La zone d'objet dans laquelle tous les personnages ainsi que la scène sont répertoriés

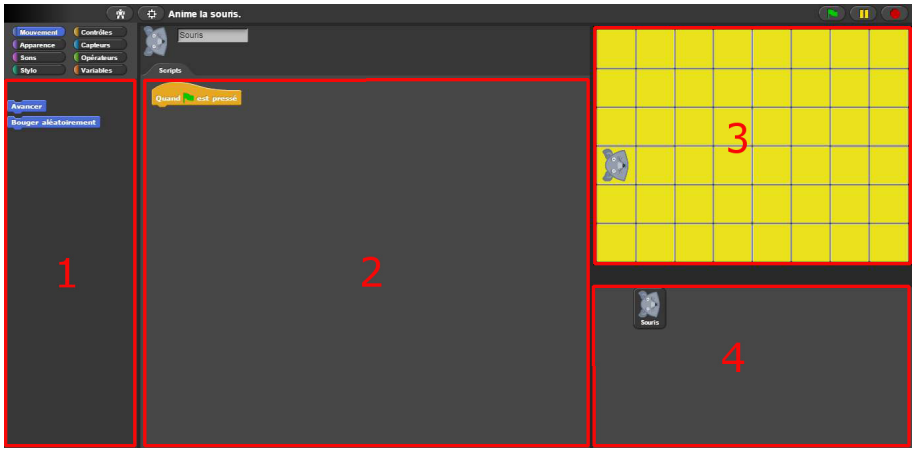


FIGURE 4.3 – Snap, l'environnement de programmation

Les nouveaux blocs

La première modification de l'environnement a été la création de 3 nouveaux blocs. Les deux premiers blocs sont reliés entre eux de par leur fonctionnement. Ces deux blocs sont des blocs d'en-tête, qui seront activés lors d'événements. L'événement déclencheur du premier bloc est le démarrage de l'animation, c'est-à-dire au moment où le bouton contenant un drapeau vert au dessus à droite de l'écran est cliqué. C'est à la suite de ce bloc que l'élève devra créer son programme. Le second bloc est activé lorsque le programme créé par l'élève, à la suite du

premier bloc, a fini de s'exécuter. Ce second bloc donne la possibilité aux professeurs de créer un programme qui sera exécuté après celui de l'élève et qui peut faire une vérification du résultat du programme de l'élève ou simplement faire un retour visuel.

Le troisième bloc créé envoie un message de réussite de la mission au serveur. Ce bloc permet aux professeurs de faire une correction automatique des programmes des étudiants et de leur laisser la possibilité de continuer les exercices. Avant l'apparition de ce bloc, une mission était considérée comme réussie à partir du moment où l'élève avait cliqué sur "Réaliser la mission", ce qui ne permettait pas aux professeurs de s'assurer que l'élève avait bien réussit la mission. Ce bloc est réservé aux professeurs et ne doit pas être accessible aux étudiants.

Un retour visuel

En parallèle à ce dernier bloc, lorsque l'élève réussit une mission, un message de réussite de l'élève a été mis en place pour le féliciter. Ce message permet également à l'élève de passer directement à la mission suivante sans devoir repasser par la liste de ses cours. Ce message est déclenché quand le bloc "La mission est réussie" est activé.

Le retrait des distractions

La modification suivante de l'environnement fut de masquer tous les boutons qui ne sont pas utiles à la réalisation de la mission par les élèves. Ces boutons permettaient, entre autres, de montrer ou non le schéma de logique du professeur aux étudiants, de créer ou supprimer un personnage ou d'en modifier le costume ou le nom. Ce retrait des distractions rejoint le principe de cohérence expliquée dans la section 3.1.1

La langue

La dernière modification a été de forcer la langue française au démarrage de l'environnement pour la première fois. Ceci permet aux élèves de ne pas devoir aller faire une modification des préférences et ainsi de se concentrer sur la réalisation du programme.

Enfin, l'ensemble des traductions françaises ont été adaptées à toutes les modifications ci-dessus.

4.4 La mise en place d'un serveur pour le Printemps des Sciences

Durant les activités du Printemps des Sciences, nous avons proposé aux élèves entre la 5ème primaire et la 2ème secondaire une initiation à la programmation en utilisant RSnap. Pour cela, il a fallu mettre en place un serveur web permettant aux élèves d'y accéder. Le serveur a été installé dans les locaux de l'UCL sur une machine dédiée aux mémorants. La machine a les caractéristiques suivantes :

— OS : Ubuntu 14.04

— CPU : Intel(R) Pentium(R) 4 CPU 3.20GHz

— RAM : 1 Gb

— HDD : 80Go

L'accès au serveur a été restreint au réseau de l'UCL et les étudiants ayant participé aux activités ont eu accès aux salles informatiques réservées aux étudiants en informatique de l'UCL. Les classes participant aux activités comprenaient 20 à 29 élèves et étaient regroupés par 2 ou 3 par ordinateur.

Chapitre 5

Les tests réalisés

Les tests ont été réalisés en deux phases. La première grosse batterie de tests qui a été réalisée s'est déroulée pendant le Printemps des Sciences par 5 classes d'élèves allant de la cinquième primaire à la deuxième secondaire et la seconde batterie de tests a été faite via le site PullReview qui propose des révisions du code Ruby et Rails.

5.1 Le Printemps des Sciences

Le Printemps des Sciences est un événement organisé par Scienceinfuse, Inforsciences, Réjouisciences, Atout Sciences et SciTech[2] qui se déroule dans toute la Wallonie et à Bruxelles. Dans le cadre de cet événement, un ensemble d'animations pour tout public se déroulent chaque année sur les sites de l'UCL à Louvain-la-Neuve et à Woluwe ainsi que sur le site de l'ULB et d'autres.

Cet événement fut l'occasion de tester l'application en situation réelle avec des élèves entrant dans la tranche d'âge ciblée par la plate-forme. Une animation a été proposée aux organisateurs du Printemps des Sciences pour des élèves de cinquième primaire à la deuxième secondaire. Au total, cinq classes se sont inscrites : deux de cinquième primaire de l'école fondamentale Martin V, une de première secondaire du collège Da Vinci et deux de deuxième secondaire dont une du collège technique Saint-Jean et une de l'institut Saint-Albert.

5.1.1 Le déroulement des activités

Après l'accueil des élèves au hall des Sciences, nous avons rejoint les locaux informatiques réservés aux étudiants en informatique de l'UCL. Une fois arrivés sur place, quelques consignes concernant le respect des autres étudiants et du matériel ont été données et nous sommes entrés dans la salle qui nous était réservée pour la durée de l'activité. Nous avons d'abord projeté une vidéo d'explication du déroulement de l'activité et le fonctionnement de la plate-forme, ensuite, les élèves se sont vus assigner un ordinateur et ont commencé à utiliser RSnap.

Tous les élèves ainsi que les professeurs se sont pris au jeu. Dans chacune des classes, les élèves étaient autonomes. Ils ont commencé à évoluer de chapitre en chapitre par groupe de 2 à 3.

Les professeurs ont pris part à l'activité en aidant les élèves lorsque ces derniers avaient des questions. Les élèves restaient sous la responsabilité de leur professeur et ils ont respecté les consignes concernant le matériel utilisé tout en restant calmes et concentrés.

5.1.2 Analyse des formulaires

Afin d'évaluer le ressenti des élèves face à l'utilisation de l'application, un questionnaire leur a été soumis une fois l'activité terminée. Le questionnaire était adapté à l'état de l'application au moment où les élèves l'ont utilisée. Il comprenait un partie administrative avec le lieu et niveau scolaire et des questions orientées sur la perception ludique de l'adolescent face au logiciel implémenté. Le questionnaire demandait une évaluation de 1 à 5 pour les points suivants :

1. Le niveau de difficulté des missions.
2. Le niveau de réussite des missions.
3. L'amusement durant la réalisation du chapitre.
4. La compréhension de la notion enseignée.

La figure 5.1 nous montre les moyennes des résultats toutes classes confondues. Il y avait environ 60% de garçons et 40% de filles, ce qui est une moyenne équilibrée et un public faisant bien partie du public ciblé.

La difficulté

D'après les élèves, la difficulté augmentait au fur et à mesure de leur progression dans les chapitres. Cette difficulté croissante est due au fait que les notions apprises sont de plus en plus complexes. Nous pouvons donc considérer cette accroissement de la difficulté comme étant normale.

La réussite

Les étudiants ont répondu qu'ils avaient réussi les missions à 90% pour les deux premiers chapitres, à 63% et 56% pour les deux derniers chapitres. Ces valeurs correspondent à une diminution de la réussite au fur et à mesure qu'ils avancent dans la matière. Après avoir personnellement ouvert les programmes de chaque étudiant, les missions d'introduction des notions étaient souvent résolues d'une mauvaise manière (en utilisant trop de blocs, en utilisant plus de boucles qu'il n'en fallait, ...) mais à la fin de chaque chapitre, chaque élève réussissant la mission le faisait en utilisant la notion comme il le fallait. Pour rappel, chaque chapitre proposé aux élèves enseigne une notion de programmation.

L'amusement

Comme expliqué dans le chapitre 3 de ce mémoire, l'amusement est une composante importante d'un cours car il permet de garder l'attention des élèves sur la matière. Le recensement des

réponses des étudiants a montré que l'amusement reste, dans la plupart des cas, constant et de bon niveau. Les élèves de l'institut Saint-Albert de Jodoigne ont donné une note d'amusement en moyenne autour de 4 sur 5, ce qui représente un "Je me suis amusé" sur l'échelle utilisée (1 : "Je ne me suis pas du tout amusé", 2 : "Je ne me suis pas amusé", 3 : "C'était banal", 4 : "Je me suis amusé", 5 : "Je me suis beaucoup amusé").

La compréhension

La compréhension des notions de programmation est le but même de l'application, il est donc extrêmement important que les élèves arrivent au bout de chaque chapitre en ayant compris la notion que le chapitre vise à enseigner. Les étudiants ont répondu en moyenne qu'ils avaient compris à plus de 75% les notions. Comme expliqué pour la réussite, après avoir vérifié la validité des réponses, nous constatons que les missions d'introductions des notions, c'est-à-dire les quelques premières missions de chaque chapitre, sont sujettes à des erreurs d'utilisation de la notion. Dans la majorité des cas, ces erreurs ne surviennent plus après une ou deux missions. Au final, les dernières missions sont réussies en utilisant correctement la nouvelle notion, le but est atteint.

La difficulté et l'amusement

Nous pouvons constater, par le biais de la figure 5.1, que malgré l'augmentation de la difficulté des chapitres au fur et à mesure de l'avancement dans l'apprentissage des notions, l'amusement ressenti par les élèves n'est pas affecté.

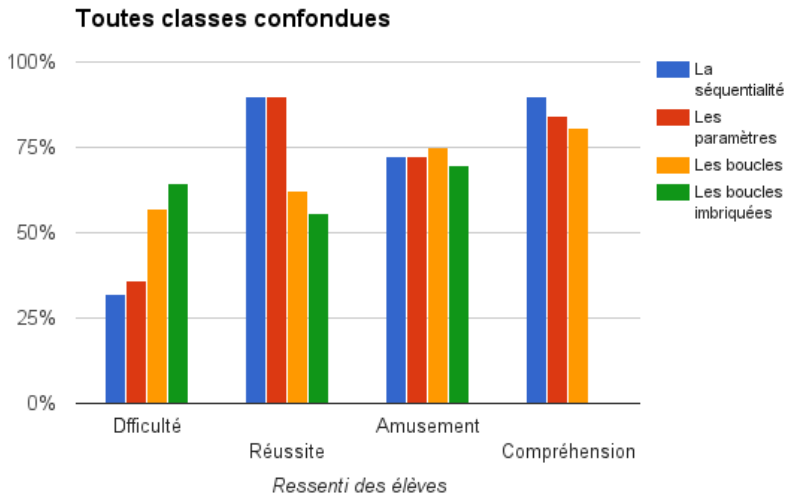


FIGURE 5.1 – Statistiques sur les questionnaires des élèves toutes années confondues

En général

Une question supplémentaire a été posée aux étudiants : "Continuerais-tu à utiliser RSnap à la maison ?". Plus de la moitié des étudiants auraient apprécié continuer l'activité chez eux ou dans leur école. La figure 5.2 nous montre le pourcentage de réponse des élèves.

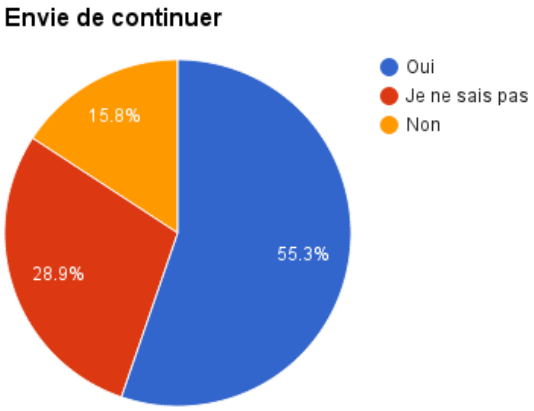


FIGURE 5.2 – Souhait de poursuivre l'utilisation de RSnap

Conclusion

D'après l'analyse des résultats, nous concluons que les activités ont été terminées avec succès. Toutefois, nous constatons que les notions ont été assimilées entre 70 et 80 %, ce qui nous semble un peu faible. Pour cette raison, nous avons pris la décision de réécrire les descriptions des chapitres et des missions.

5.2 PullReview

PullReview est un site web proposant des révisions de code Ruby sur les répertoires en-ligne du type GitHub et BitBucket.

PullReview détecte les différents problèmes du code en allant de la simple information au niveau critique d'erreur. Ces erreurs détectées sont des vérifications du bon usage des fonctionnalités de Ruby, des possibles failles de sécurité et du style d'écriture du code.

Lors de l'implémentation des extensions de l'application, le site PullReview a été consulté plusieurs fois pour s'assurer qu'il n'y avait aucune faille de sécurité et que le style d'écriture du code était cohérent avec le style de Ruby.

Au début du mémoire, le site montrait 268 erreurs, autant de documentation que de style d'écriture. L'ensemble du code a été révisé et il ne reste actuellement que 47 erreurs alors que la quantité de code a augmenté.

Chapitre 6

Conclusion

6.1 La problématique

L'application implémentée par Simon Claessens et Gaëtan Collart dans leur mémoire en 2014-2015 proposait un ensemble fixe de missions ludiques visant à accompagner l'apprentissage de la programmation aux jeunes de 10 à 14 ans. Cette application n'offrait pas à un professeur la possibilité de créer ses propres missions pour élaborer son cours car le contenu de l'application était commun à tous les utilisateurs.

6.2 La solution proposée

Le but de ce mémoire est de rendre l'application proposée par Gaëtan Collart et Simon Claessens accessible aux professeurs de primaire et secondaire ainsi que de rendre l'application plus agréable à l'utilisation pour les élèves.

La première modification a été le changement de structure du contenu proposé aux élèves. Celui-ci est passé d'une simple liste de missions à un ensemble de chapitres qui regroupaient eux-mêmes des missions afin de pouvoir les assembler par thématique et par notion de programmation enseignée.

Ensuite, les blocs d'introspections ont été développés dans la plate-forme Snap. Ces blocs permettent aux professeurs de tester la d'une mission automatiquement, ce qui facilite la tâche du professeur en ce qui concerne la correction des travaux réalisés par les étudiants. En parallèle à ce changement, le contrôle de l'avancement d'un élève tout au long des chapitres a été automatisé, c'est-à-dire que l'élève a automatiquement accès à une mission lorsqu'il a réussi toutes les missions précédentes. En plus de la correction automatique, la mission peut être marquée comme devant faire l'objet d'une correction non-automatique par un professeur. Dans ce cas, l'élève devra attendre que sa mission soit corrigée par le professeur et considérée comme réussie pour pouvoir passer à la suivante.

Après les activités du printemps des sciences, la plate-forme a encore vu un changement majeur : l'apparition du statut de professeur permettant à celui-ci de créer et modifier ses propres

missions et chapitres, ce qui concorde avec les caractéristiques d'un LMS (voir section 1.3). Enfin, les cours ont fait leur apparition. Ceux-ci reprennent un ensemble de chapitres et peuvent être comparés à ceux proposés dans l'Enseignement obligatoire pour les matières scientifiques, c'est-à-dire des chapitres visant chacun l'enseignement d'un ensemble de connaissances.

L'apparition du statut de professeur s'est accompagnée de celui d'étudiant. Les élèves, une fois leur compte créé avec ce statut, ont accès à une liste des cours proposés par les professeurs. A l'inverse, les professeurs ont accès à une liste des étudiants inscrits à au moins un de leurs cours. Ceci permet aux professeurs de vérifier la validité des réponses de leurs étudiants, ce qui répond au besoin d'un système de vérification de connaissances d'un LMS (voir section 1.3.4).

6.3 Les résultats

Comme expliqué dans le chapitre sur les tests réalisés, le Printemps des Sciences a été l'occasion de tester l'application en situation réelle. Nous avons amené des classes d'élèves allant de la 5ème primaire à la 2ème secondaire à utiliser l'application et à découvrir le contenu proposé. Ces étudiants ont ensuite répondu à un questionnaire demandant une appréciation de différents points.

Après analyse des réponses des étudiants au formulaire, nous constatons que la difficulté pour résoudre les chapitres était croissante. Ceci nous montre que l'ordre choisi pour l'apprentissage des notions reprenait celles-ci de la plus simple à la plus compliquée. De plus, les réponses des étudiants ont montré que ces notions enseignées dans le cadre de chapitres ont été assimilées par les élèves, ce qui nous confirme que la stratégie didactique (voir section 1.3.2) utilisée est fonctionnelle.

Des statistiques sur le questionnaire ont révélé que les élèves se sont amusés en réalisant les chapitres proposés au Printemps des Sciences et que la plupart d'entre eux souhaitaient continuer à utiliser l'application une fois l'activité terminée.

6.4 Le bilan

La problématique de départ était l'absence de support pour les professeurs voulant enseigner la programmation aux étudiants entre 10 et 14 ans. L'application de Simon Claessens et Gaëtan Collart proposait un ensemble commun de missions à tous les utilisateurs, ce qui ne permettait pas à un professeur de créer son propre cours et d'avoir accès aux réponses de ses propres étudiants. L'application modifiée durant la réalisation de ce mémoire permet à chaque enseignant la création de cours personnalisé sur les bases de la programmation. L'évaluation des élèves est directement intégrée et accessible dans les modifications apportées. De plus, l'application contient déjà un cours de base proposant un enseignement de quelques notions de base de la programmation.

Nous pouvons conclure que l'application est prête à être mise à la disposition des enseignants des écoles primaires et secondaires. Cet outils doit servir de support aux professeurs dans leur

enseignement à l'apprentissage de la programmation. Ceci permettra d'éveiller l'intérêt des jeunes de 10 à 14 ans pour l'informatique et ainsi amener plus d'étudiants à s'orienter vers l'informatique, ce qui comblera le manque de techniciens en informatique.

6.5 Perspectives

Au terme de ce mémoire, l'application est fonctionnelle et contient et contient déjà un cours proposant un enseignement de notions de base de la programmation. Quelques fonctionnalités qui pourraient être utiles ont été imaginées pour améliorer l'expérience des utilisateurs.

6.5.1 L'import et l'export de contenu

La première extension qui serait utile aux professeurs est l'import et l'export du contenu. Il serait bénéfique pour l'ensemble des professeurs de pouvoir partager entre eux différents cours, chapitres ou missions. Ceci permettrait aux professeurs d'apprendre des autres professeurs et ainsi de proposer des cours de meilleur qualité.

6.5.2 La charte graphique du LMS

Comme expliqué dans le mémoire rédigé par Simon Claessens et Gaëtan Collart [6], la partie RSnap de l'application, c'est à dire le LMS (Learning Management System, Système d'accompagnement de l'apprentissage), utilise Bootstrap de manière brute, ce qui pourrait choquer certains utilisateurs. Cette utilisation de Bootstrap rend la charte graphique de l'application banale. C'est pourquoi un changement de toute la partie visuelle pourrait faire partie d'un travail de plus ou moins grande envergure.

6.5.3 Un système de messagerie

Une autre extension qui pourrait être bénéfique à tous les utilisateurs serait d'ajouter un système de messagerie, personnelle ou via des forums. Cela permettrait aux utilisateurs de partager leurs difficultés ou leurs réussites par rapport aux exercices ou aux professeurs d'ajouter des précisions si une mission est trop difficile. En parallèle de ce système de messagerie, une modification par rapport à la création de compte utilisateur "Professeur" serait de le rendre non-accessible au grand public mais seulement via une requête aux administrateurs qui créeraient le compte avec les autorisations de professeur.

6.5.4 La protection par un mot de passe pour l'inscription aux cours

La dernière extension serait une option choisie par les professeurs lors de la création d'un cours : le verrouillage des inscriptions au cours par mot de passe. Cette extension permettrait aux professeurs de créer des cours auxquels les élèves n'auraient accès qu'après que le professeur l'ai décidé.

Bibliographie

[1] https://github.com/simonhock/rsnap.

[2] https://github.com/simonhock/Snap--Build-Your-Own-Blocks.

[3] ACM and IE. Report of the joint informatics europe & acm europe working group on informatics education. apr 2013.

[4] ADEPS. Cours généraux du moniteur sportif "initiateur". http://www.sport.cfwb.be/index.php?id=7011.

[5] Andrus Ansip. Digital skills, jobs and the need to get more europeans online. mar 2015.

[6] Simon Claessens and Gaëtan Collart. Plateforme web pour accompagner l'apprentissage de la programmation aux 10 - 14 ans.

[7] Fédération Wallonie-Bruxelles FWB. Socles de compétences.

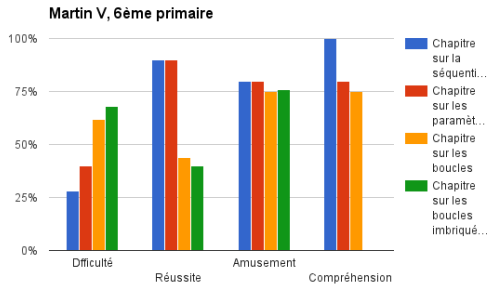
[8] Rails. Rails : Web development that doesn't hurt. http://rubyonrails.org/.

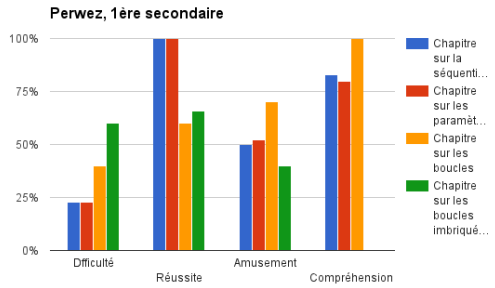
[9] David Vellut. 10 principes pédagogiques à prendre en compte pour concevoir des environnements d'apprentissage multimédia. http://www.formavox.com/principes-pedagogiques-environnements-apprentissage-multimedia.

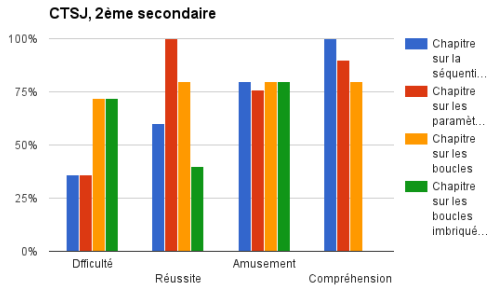
[10] Wiki. Learning management system. https://fr.wikipedia.org/wiki/Learning_management_system#Fonctions_d.27une_plateforme_d.27apprentissage.

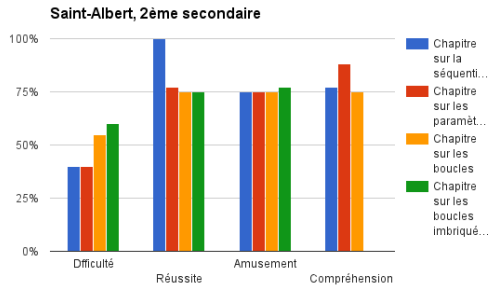
Annexe A

Statistiques sur le Printemps des Sciences.









Annexe B

Diagrammes

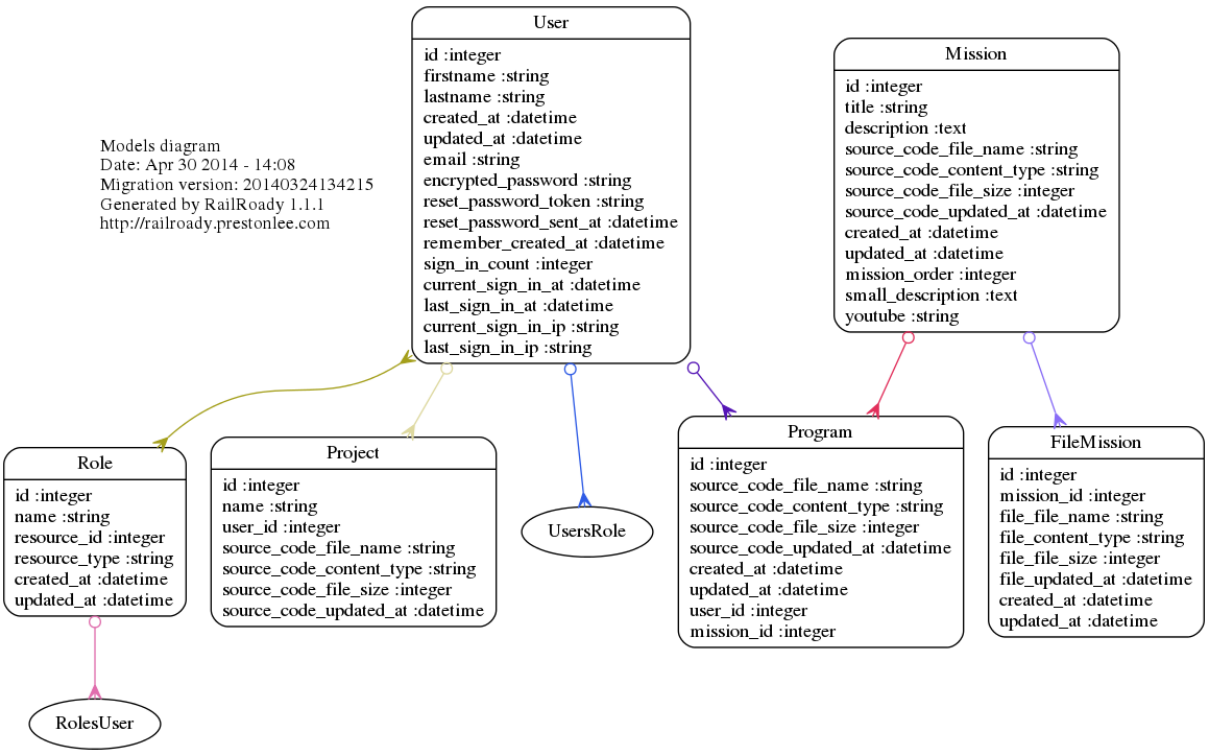


FIGURE B.1 – Modèle de RSnap avant modifications

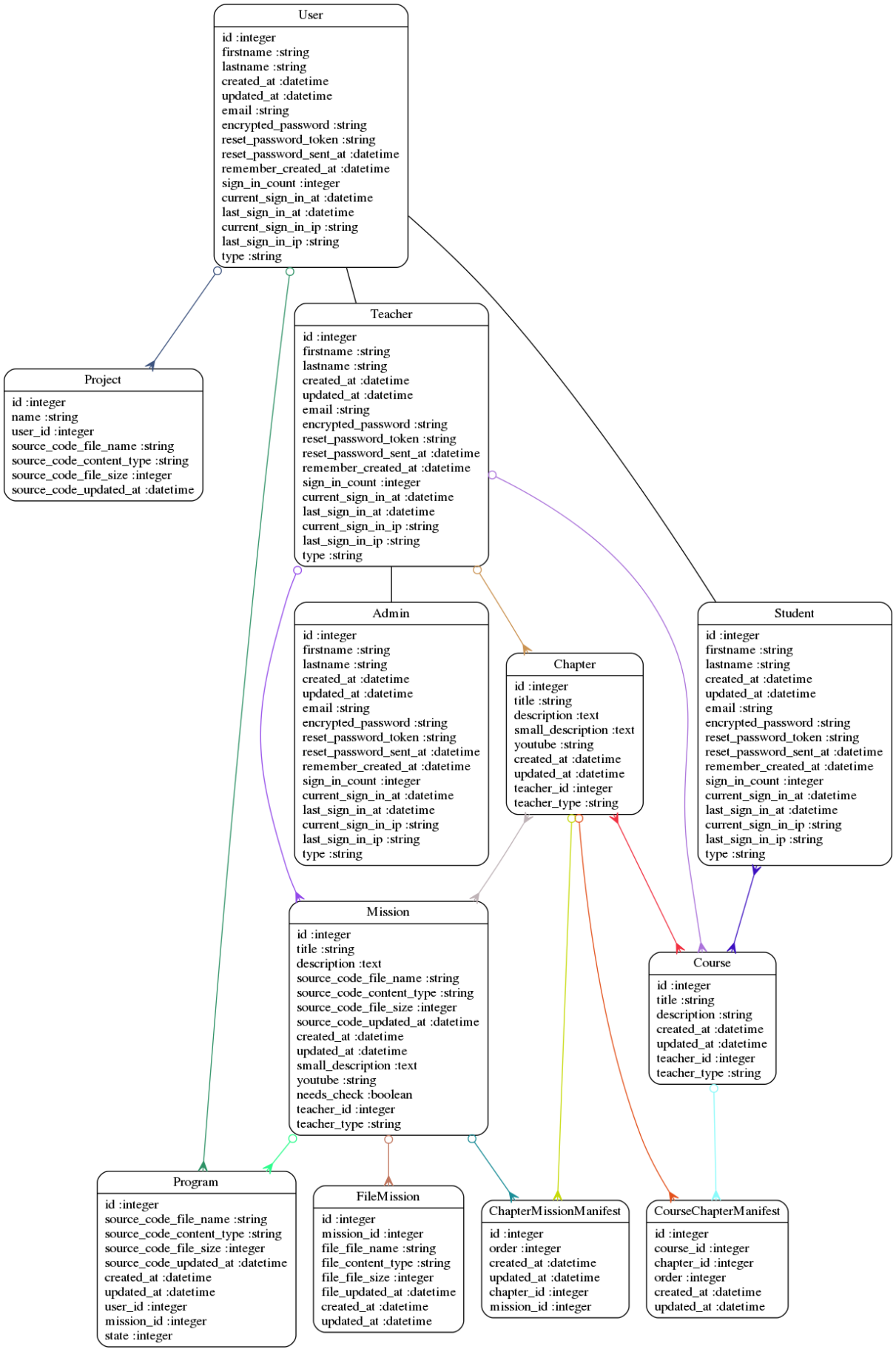


FIGURE B.2 – Modèle de RSnap après modifications

Annexe C

Le guide du professeur

Cette annexe est destinée aux professeurs voulant créer leur propre cours sur la plate-forme afin d'amener leurs élèves à apprendre la programmation. Nous allons détailler la structure de l'application et ensuite la construction d'un programme et enfin expliquer les fonctionnalités utiles à la gestion de l'avancement des élèves dans le cours créé.

C.1 La structure de l'application

Nous allons commencer par expliquer les rôles de professeur et d'étudiant. Nous allons ensuite détailler la structure de la plate-forme en ce qui concerne les cours proposés par les professeurs pour les étudiants. Pour ceci, nous expliquerons ce qu'est une mission et son programme, la structure des chapitres, la structure des cours et enfin détailler les fonctionnalités de gestion des élèves.

C.1.1 Les rôles

Les utilisateurs peuvent jouer 3 rôles différents : celui d'administrateur, de professeur ou d'élève. Chacun d'eux a des autorisations bien spécifiques.

Les administrateurs

Les administrateurs ont tous les droits sur l'application, ce sont eux qui iront faire les modifications diverses en cas de problème ou de requêtes diverses. Ils peuvent créer, modifier ou supprimer les comptes utilisateurs, les cours, les chapitres, les missions et les réponses des étudiants. Les administrateurs sont nommés par des administrateurs déjà existants.

Les professeurs

Les professeurs sont les utilisateurs qui proposent des cours aux élèves. Ils vont créer un ou plusieurs cours pour lesquels ils auront les droits d'édition et de suppression. Ces utilisateurs ont l'autorisation de créer, modifier ou supprimer les cours, chapitres et missions leur appartenant

et n'ont accès qu'à leurs créations. Ils ont aussi accès à une liste des élèves inscrits à leurs cours et peuvent voir les réponses de ces élèves pour les cours leur appartenant, ce qui leur permet de corriger les réponses de ces mêmes élèves. Le rôle de professeur est choisi par l'utilisateur au moment de la création du compte.

Les élèves

Les élèves sont les utilisateurs avec le moins de droits. Ceux-ci, une fois leur compte créé, ont accès à la liste des cours développés par les professeurs et peuvent s'y inscrire. Une fois inscrit à un cours, l'élève peut commencer à évoluer dans les chapitres de ce cours. Il peut ensuite voir tous les chapitres proposés dans le cadre du cours mais ne peut les réaliser que un par un et ne sait passer au chapitre suivant qu'une fois le précédent réussi.

C.1.2 Les missions

Une mission est comparable à un exercice. La figure C.1 nous montre la description d'une mission proposée pour apprendre le fonctionnement de l'application.

FIGURE C.1 – Description d'une mission

Les professeurs peuvent, comme décrit dans la section sur les rôles, créer un ensemble de missions. La partie description de la mission est en quelque sorte l'énoncé du problème que l'élève doit résoudre. Une mission contient un titre, une description, une petite description s'affichant dans la liste des missions et un lien YouTube pour une vidéo intégrée.

Le programme d'une mission

Le programme d'une mission est ce que l'élève complètera pour résoudre le problème posé dans la description de la mission. L'élève arrivera devant le programme une fois qu'il aura cliqué sur "Réaliser la mission". La figure C.2 nous montre le programme "vierge" correspondant à la description de la figure C.1.

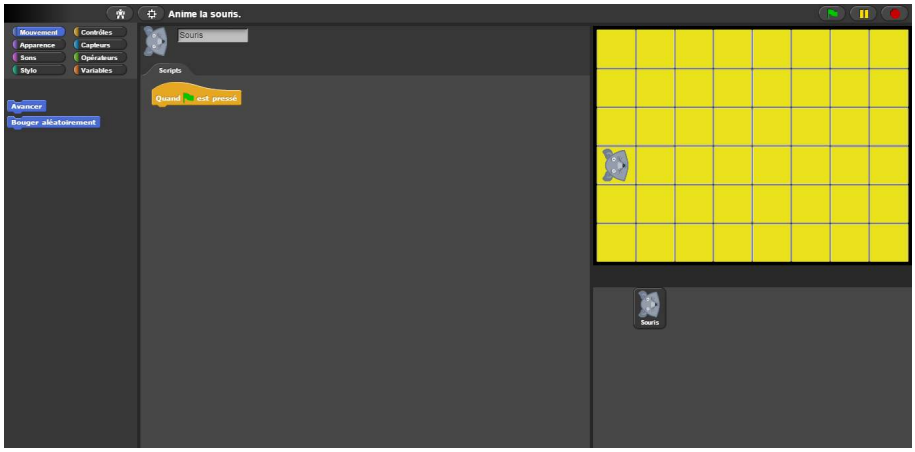


FIGURE C.2 – Programme "vierge" d'une mission

Chaque mission a un programme "vierge" qui lui est associé. Au moment où l'élève cliquera sur "Réaliser la mission", une copie de ce modèle sera créée à partir de laquelle l'élève aura le droit à l'édition afin de pouvoir le compléter. À la création d'une mission par un professeur, il faut tout d'abord passer par le choix du titre, de la description, de la petite description qui s'affichera dans la liste des missions et d'une option expliquée plus tard avant d'être amené à créer le programme "vierge" que les élèves compléteront.

C.1.3 Les chapitres

Les chapitres sont composés, comme les missions, d'un titre, d'une description, d'une description courte s'affichant dans les listes de chapitres et d'un lien YouTube. La figure C.3 nous montre la description du chapitre contenant la mission décrite ci-dessus.

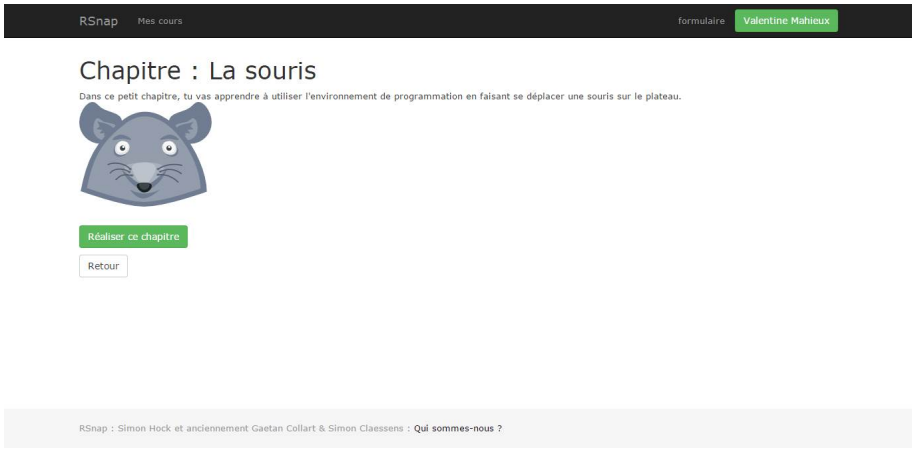


FIGURE C.3 – Description d'un chapitre

La différence avec les missions est que le bouton "Réaliser ce chapitre" n'amène pas l'élève à un programme mais à une liste de missions. Cette liste de missions est la liste développée par le professeur ayant créé le chapitre et décidé des missions le composant. La figure C.4 nous montre une liste de missions faisant partie d'un chapitre.

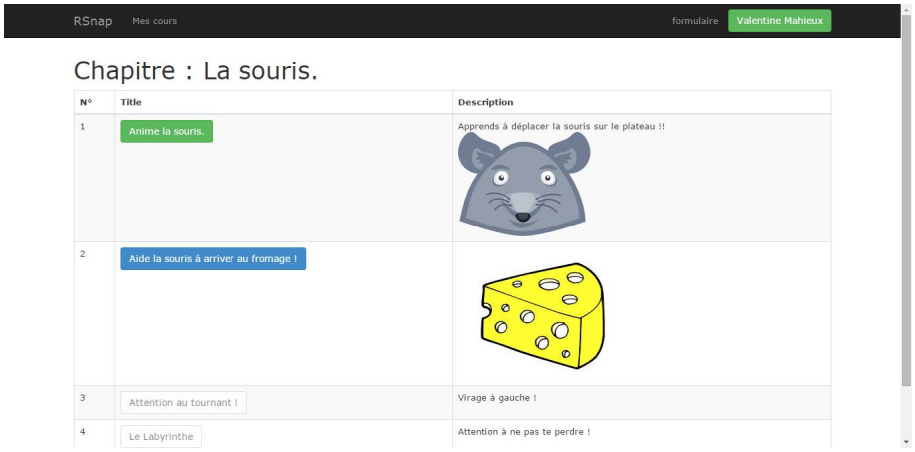


FIGURE C.4 – Liste des missions d'un chapitre

Comme on peut le voir dans cette figure, les titres des missions sont en vert, en bleu et en gris. Le screenshot a été pris en étant sur un compte étudiant qui avait déjà réussi la première mission notée en vert, il est en train de réaliser la seconde dont le titre est en bleu, avant de pouvoir passer aux suivantes, grisées car inaccessibles.

C.1.4 Les cours

Les cours sont créés par les professeurs ou les administrateurs, ils sont visibles par les élèves et ces derniers peuvent s'y inscrire. Un cours est composé d'un titre et d'une simple description qui est affichée dans la liste des cours. Dans la même optique qu'un chapitre contient des missions, un cours est un regroupement de chapitres que l'élève peut réaliser les uns après les autres. La figure C.5 nous montre la liste des chapitres d'un cours. Le système d'accès aux chapitres est le même que pour les missions avec le code couleurs (vert, bleu, gris), c'est-à-dire que l'élève pourra réaliser un chapitre seulement si le précédent a été réussi, et donc que toutes les missions le composant ont aussi été réussies.

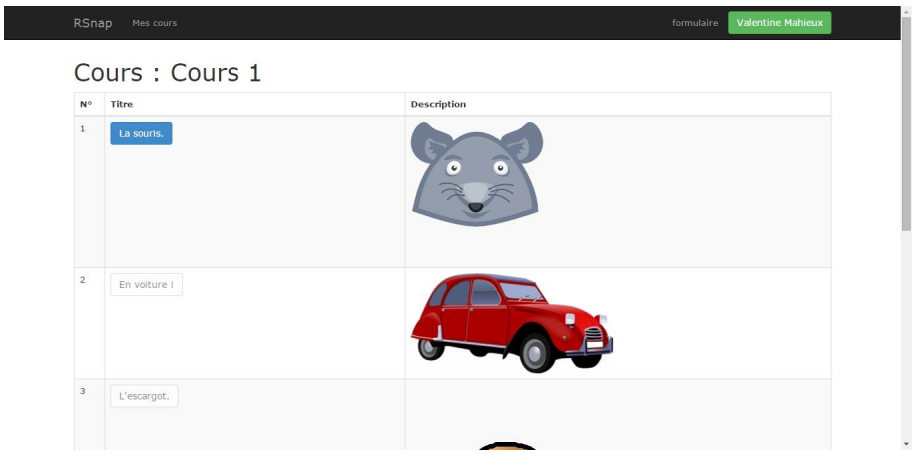


FIGURE C.5 – Liste des cours auxquels l'étudiant est inscrit

La figure C.6 nous montre la liste des cours auxquels un élève est inscrit. Le bouton "S'inscrire

à un cours" permet de voir la liste des cours disponibles et de s'y inscrire. Le bouton "Se désinscrire" permet de faire l'inverse, c'est-à-dire retirer un cours de sa liste d'inscription.

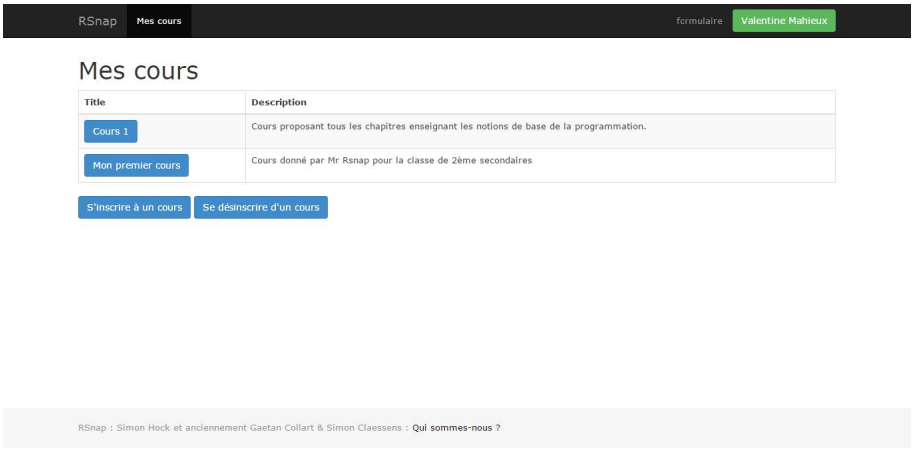


FIGURE C.6 – Liste des cours auxquels l'étudiant est inscrit

C.2 La création d'une mission

Concevoir le programme d'une mission est la partie la plus complexe de la création d'un cours. Cette partie va décrire le fonctionnement d'un programme et ensuite expliquer comment a été créée la première mission.

C.2.1 Les différentes parties d'un programme

La figure C.7 nous montre la page où un programme est créé, édité ou exécuté.

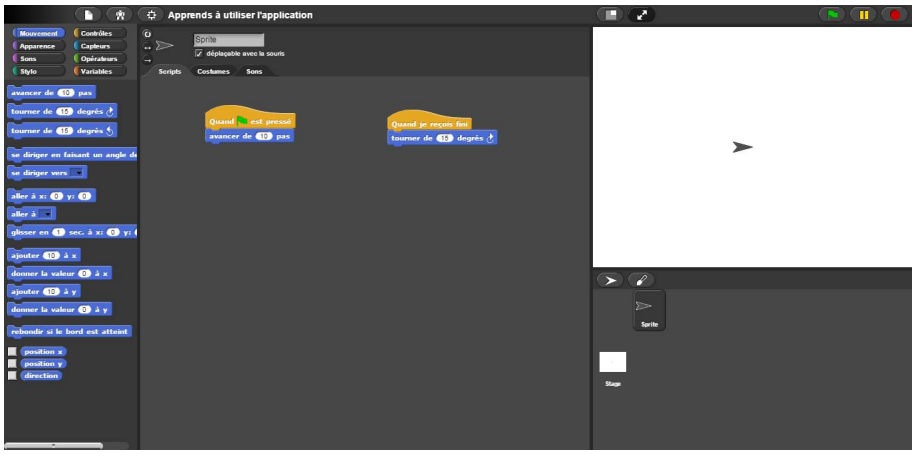


FIGURE C.7 – Vue d'un programme

Les sprites

Un programme dans RSnap est une animation de plusieurs personnages sur une scène. Sprite est un mot anglais désignant un lutin, ce sont les personnages de l'animation. La zone en dessous

à droite de l'écran (voir fig C.7) montre les sprites existants et le sprite actuellement sélectionné. Dans la figure C.7, il n'existe qu'un sprite en forme de flèche et la scène.

Les blocs

RSnap permet d'enseigner la programmation via une interface visuelle et propose de la programmation par blocs. Ces blocs sont comparables à des lignes de code dans ce sens que chaque bloc aura une répercussion sur le fonctionnement du programme. Il existe plusieurs types de blocs : les blocs d'événement, les blocs d'action, les blocs de contrôle et les blocs de fonction.

Les blocs d'événement sont des blocs d'en-tête comme celui de la figure C.8a. Ces blocs sont activés lors de certains événements et dans ce cas-ci, quand on a cliqué sur le bouton avec le drapeau vert.

La figure C.8b nous montre un bloc d'action. Ces blocs auront chacun un effet différent sur l'animation et dans ce cas-ci, le personnage avancera de 10 pas.

Les blocs de fonction sont des blocs qui ne vont pas avoir d'effet sur l'animation mais vont changer l'effet d'un autre bloc. La figure C.8c nous montre un bloc qui donnera un nombre aléatoire. Par exemple, la figure C.9a nous montre une composition du bloc d'action et du bloc de fonction qui feront avancer le personnage d'une distance aléatoire entre 1 et 10 pas.

Finalement, la figure C.8d nous montre un bloc de contrôle. Les blocs de contrôle sont des blocs qui si ils sont seuls, ne vont avoir aucune utilité alors qu'une fois combiné avec d'autres blocs, ils influenceront leur effet. Dans l'exemple de la figure C.9b, le bloc "avancer" sera répété 5 fois.

(a) Un bloc d'événement

(b) Un bloc d'action

(c) Un bloc de fonction

(d) Un bloc de contrôle

FIGURE C.8 – Les types de blocs

(a) Un bloc de fonction et un bloc d'action

(b) Un bloc de contrôle et un bloc d'action

FIGURE C.9 – Les compositions de blocs

La partie à l'extrême gauche de l'écran (voir fig C.7) contient tous les blocs qui peuvent être

utilisés. On peut comparer cette zone avec la palette de couleurs qu'un peintre utilise pour créer sa toile.

La zone de script

La zone de script est la zone dans laquelle seront composés les blocs pour créer l'animation. Dans la figure C.7, nous pouvons voir qu'il y a déjà 2 blocs d'en-tête et 2 blocs d'actions attachés 2 à 2. La composition de gauche fera avancer le personnage une fois que le bouton avec le drapeau sera sélectionné. La composition de droite est particulière, le bloc d'en-tête "Quand je reçois fini" a comme événement déclencheur que l'ensemble des blocs attachés sur le bloc d'en-tête "Quand le drapeau est pressé" ont fini leur exécution. En d'autres termes, tout ce qui se passera dans la composition de droite sera exécutée après la composition de gauche. Si le drapeau est sélectionné, le personnage avancera de 10 pas en avant et ensuite tournera de 15 degrés vers la droite. Ces deux blocs d'en-tête sont donc liés. Il existe un autre bloc d'en-tête ayant pour événement le clic sur le drapeau permettant de ne pas exécuter le second bloc d'en-tête, nommé "Quand le drapeau est pressé (sans envoyer de "fini")". Ce bloc d'en-tête sera utile pour faire toute la logique interne au programme qui sera cachée aux étudiants. Par exemple : vérifier que le personnage ne touche jamais le bord, faire parler le personnage en fonction de son déplacement, ...

La zone de script est propre à chaque personnage. Dans la figure C.7, nous pouvons voir que dans la zone de sprites, la flèche est sélectionnée. La zone de scripts contient donc les scripts de cette flèche et les blocs "avancer" la feront progresser.

Les costumes

La zone de costumes dépendra, comme pour la zone de scripts, du personnage sélectionné. La figure C.10 nous montre les costumes du personnage. Ces costumes peuvent être modifiés grâce au programme et aux blocs d'actions du type "Apparence". Par exemple, on pourrait faire danser un personnage en alternant ses costumes grâce à la composition des blocs de la figure C.11.

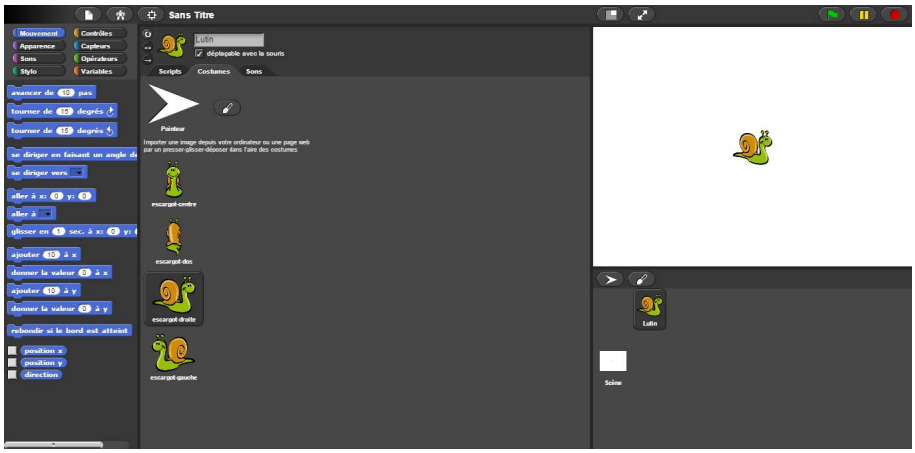


FIGURE C.10 – Les costumes d'un personnage

FIGURE C.11 – Composition "danse"

C.2.2 Les fonctionnalités utiles

Cette partie va expliquer quelques fonctionnalités utiles pour créer un programme.

Cacher des blocs

Il est possible de cacher des blocs aux élèves, aussi bien dans la zone des blocs à gauche que dans la zone de scripts. Les élèves ne peuvent pas cacher des blocs ou afficher ceux qui étaient cachés au préalable. Dans la zone des blocs, en faisant un clic droit, nous pouvons afficher un menu déroulant dans lequel un bouton "Hide primitives". Ce bouton nous permet de cacher tous les blocs primitifs, c'est-à-dire les blocs de base, de la section actuelle (mouvement, contrôle, apparence,...). Si un bloc primitif est déjà caché, un bouton "Show primitives" permet d'afficher tous les blocs qui étaient cachés dans la section.

Dans la zone de scripts, chaque bloc peut être aussi caché en faisant un clic droit dessus et en sélectionnant "cacher". Les blocs cachés peuvent être affichés grâce au bouton "Tout afficher" du menu déroulant s'affichant en faisant un clic droit dans la zone de scripts.

Le fait de cacher des blocs aux élèves va permettre de ne pas les surcharger d'informations dès le départ et de limiter les blocs auxquels ils ont accès pour résoudre la mission. Cela permet aussi de créer une logique interne à votre animation sans que l'élève ne puisse la voir ou la modifier.

L'import d'un programme

Un programme peut être exporté via le menu déroulant du bouton "Fichier" en haut à gauche de l'écran. Une fois le bouton "Exporter le projet" sélectionné, une nouvelle page de l'explorateur Internet s'ouvre avec le programme sous forme de texte. En choisissant "fichier" et puis "sauver" ou "ctrl"+"s", il est possible de créer un fichier dans lequel sauver le programme. L'extension du fichier doit être ".xml". Une fois le programme exporté, il peut être importé dans un autre programme, toujours via le menu "ficher" et ensuite "importer", ce qui le remplacera. Cette fonctionnalité vous permettra de sauvegarder un programme sur votre ordinateur pour le réutiliser dans une autre mission.

La création de nouveaux blocs

Les blocs disponibles dans la zone des blocs sur la gauche de l'écran (voir figure C.7) sont des blocs de base. Il est possible de créer de nouveaux blocs ayant le même effet qu'une composition de blocs de base. La figure C.12b nous montre la définition du nouveau bloc "danser" de la figure C.12a. Ces nouveaux blocs créés ne peuvent pas être cachés mais leur définition n'est pas accessible aux étudiants. Le menu déroulant s'affichant par un clic droit dans la zone de script contient un bouton "Nouveau bloc".

(a) Le nouveau bloc

(b) La définition du nouveau bloc

FIGURE C.12 – Le nouveau bloc et sa définition

La réussite de la mission

Dans les blocs d'action, il existe un bloc "La mission est réussie". Ce bloc, une fois exécuté, enverra un message vers RSnap disant que la mission a été réussie, ce qui permettra à l'élève de passer automatiquement à la mission suivante. En général, ce bloc sera utilisé dans un bloc de contrôle conditionnel du type "si". La figure C.13 nous montre la vérification automatique du programme "Anime la souris". Pour éviter tout problème avec d'autres scripts encore en exécution, la première chose à faire est d'arrêter tous les autres scripts. Il faut ensuite vérifier si l'élève a bien déplacé la souris, le contrôle se fait sur la position et la direction de la souris. Une fois le contrôle effectué, le programme envoie un retour à l'étudiant, le message dépendra de la réussite ou non de la programmation.

FIGURE C.13 – Vérification de la réussite de la première mission