3A — STI5 105 — Projet en Programmation — 2018–19

Projet Ô Automates !

Vincent Hugot — `vincent.hugot@insa-cvl.fr` — Bureau CRI 09

9 septembre 2018

---

0 Table des matières

1 Vue d'ensemble

Le but du projet est de réaliser une application – avec interface graphique – pour la construction de graphes orientés et de diagrammes sagittaux [(a)] d'automates d'états finis, avec des facilités automatiques et semi-automatiques pour produire des diagrammes naturels, plaisants, et lisibles, exportés au format LaTeX/Ti*k*Z. Voyez les figures émaillant ce document pour quelques exemples. [(b)]

Cette application a pour vocation de venir en aide à la minorité peu considérée des professeurs de théorie des langages, qui ont très envie d'illustrer leurs diapositives et polycopiés avec de nombreux et beaux diagrammes, mais qui n'ont pas le luxe de passer à chaque fois une heure à se reconnecter avec leur artiste intérieur afin de d'imaginer comment l'automate devrait être disposé pour être "joli", et encore une heure à se battre avec LaTeX et Ti*k*Z pour coder une version approximative de cette vision.

Ce document donne des pistes quant à ce qui est attendu, mais ne doit pas être abordé comme une spécification exhaustive. Posez des questions !

Ceci est à réaliser en Python 3 ; le choix est ouvert pour la partie graphique, PyQt5 étant le choix par défaut [(c)].

Un prototype, réalisé dans le cadre du projet d'application en fin d'année par vos prédécesseurs, est mis à votre disposition pour démarrer — mais rien ne vous empêche de recommencer à zéro. C'est même fortement recommandé.

(a). i.e. diagrammes avec des ronds et des flèches. Même racine latine, *sagitta*, "flêche", que le signe Sagittaire... ici dans un contexte plus scientifique.

(b). Sources : mes polys, `http://www.texample.net/tikz/examples/feature/automata-and-petri-nets/`, `https://www3.nd.edu/~kogge/courses/cse30151-sp18/Public/Assignments/tikz_tutorial.pdf`, `https://tex.stackexchange.com/questions/148158/make-a-tikz-automata-edge-pass-outside-the-automata`,...

(c). cf. `http://doc.qt.io/qt-5/examples-graphicsview.html` pour de la documentation C++. C'est à adapter à la version Python, car PyQt5 est juste une bibliothèque de liens (bindings) vers Qt5.

2 Qu'est-ce qu'un automate ?

On n'abordera le module de théorie des langages en seconde période. Pendant la première moitié du projet, on verra donc les automates purement comme des diagrammes, et je ne donnerai (à l'oral) qu'une vague intuition de leur sémantique – ce qu'ils "veulent dire".

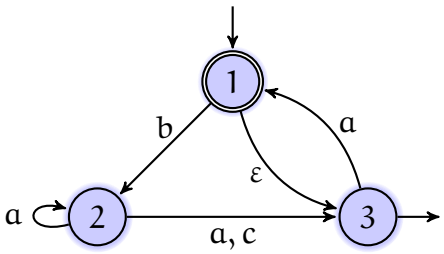
On aura malgré tout besoin de la définition formelle de la structure, afin d'avoir le vocabulaire nécessaire pour communiquer.

Un **automate fini non déterministe** est un 5-uplet $A = (\Sigma, Q, I, F, \delta)$ où :

- Q : ensemble fini d'états
- Σ : alphabet fini
- $I \subseteq Q$: états initiaux
- $F \subseteq Q$: états terminaux
- $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$: relation de transition

$$(p, a, q) \in \delta \overset{\text{notation}}{\equiv} p \xrightarrow{a} q \qquad \delta(p, a) = \{q \mid p \xrightarrow{a} q\}$$

Voici un petit diagramme sagittal d'automate, illustrant les différentes conventions de représentation :



L'alphabet est l'ensemble des étiquettes de transition, i.e. les lettres lues par l'automate. Le symbole ε ne fait pas partie de Σ, il est utilisé pour une transition qui se déclenche sans rien lire. On a

$$\Sigma = \{a, b, c\}$$

Les états sont les ronds :

$$Q = \{1, 2, 3\}$$

Les états initiaux sont indiqués par une flèche entrante déconnectée :

$$I = \{1\}$$

Les états finaux sont indiqués par un double cercle ou par une flèche sortante déconnectée :

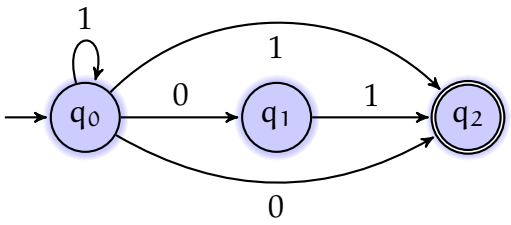
$$F = \{1, 3\}$$

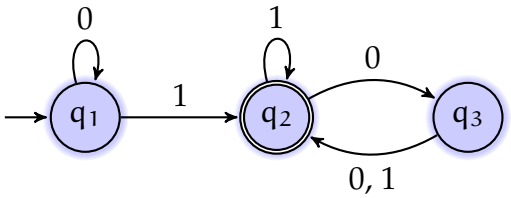
Notons qu'il n'est pas *du tout* classique d'utiliser ces deux conventions dans le même automate ; ou même dans le même document. L'automate ci-dessous commet donc une faute de goût.

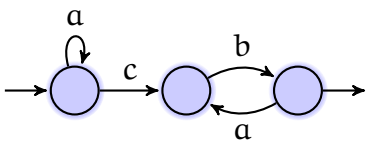
Restent les transitions :

$$\delta = \{(1, b, 2), (1, \varepsilon, 3), (2, a, 2), (2, a, 3), (2, c, 3)\}$$

Faites le même exercice avec les automates suivants :







3 Partie graphique / manuelle

Difficulté modérée ; progression incrémentale ; programmation objet ; algorithmique simple ; travail de documentation bibliothèques Python, LaTeX, TikZ. Groupe de 3 recommandé.

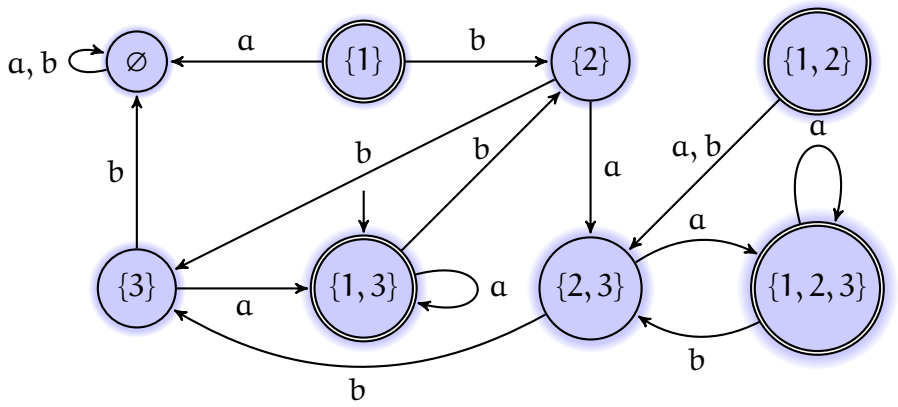
Réaliser un diagramme sagittal d'automate est largement une activité de dessin, régie en partie par des considérations purement esthétiques et en partie par des conventions et habitudes liées au domaine scientifique et favorisant la lisibilité des diagrammes.

Le point de départ de l'application est donc *in fine* un logiciel de dessin vectoriel assez ordinaire, grandement simplifié par le faible nombre de figures de base dont on a besoin : flèches, droites ou incurvées, cercles simples et doubles, étiquettes en langage mathématique (entrée en LaTeX), et c'est tout.

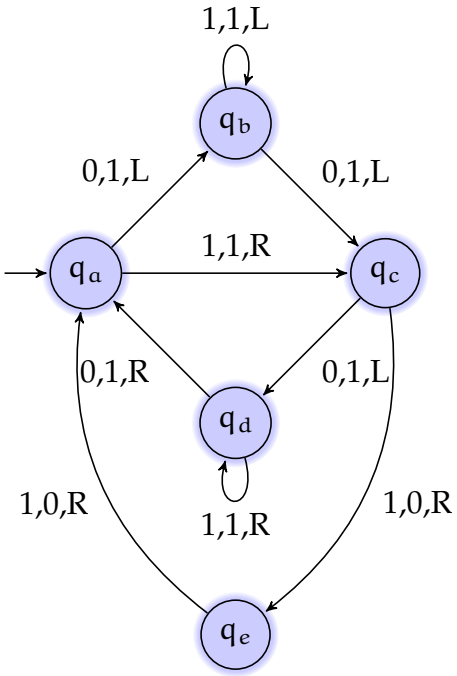
Le logiciel utilisera le vocabulaire "automates" et non le vocabulaire "dessin", par exemple "état" et non "cercle". Si un jour il nous vient l'envie d'avoir des états rectangulaires, cela doit être possible...

Placer des états et définir des transitions doit être aussi simple et rapide que possible pour l'utilisateur .

Notons que, bien que les noms des états soient arbitraires, on voudra parfois les nommer par des formules. Par exemple, l'automate suivant est obtenu via un algorithme de déterminisation, et il est essentiel de pouvoir faire apparaître les sous-ensembles dans les états afin d'illustrer la démarche :



De même, dans l'automate suivant, ce sont les transitions qui portent des étiquettes un peu "compliquées", et pas seulement une seule lettre — pour ceux que ça intéresse il s'agit ici en fait d'une machine de Turing :



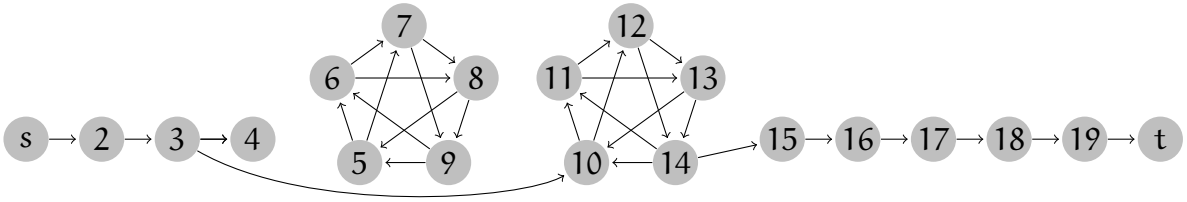
Il n'est pas *totalement nécessaire* que les formules mathématiques soient rendues correctement dans l'interface graphique – le code LaTeX peut être conservé tel-quel jusqu'à l'export vers LaTeX/Ti*k*Z. Ce serait toutefois appréciable si les formules pouvaient être rendues directement dans l'interface. Si c'est le cas, il faut toutefois que l'utilisateur puisse désactiver cette fonctionnalité, car le rendu dépend du contexte d'évaluation (LaTeX est

essentiellement un langage de programmation) et le code de l'utilisateur peut donc ne prendre sens qu'au sein de son document.

Ceci peut poser certains problèmes au moment d'évaluer la taille des états, par exemple. A vous de proposer des solutions pour gérer ces cas de la manière la plus automatique et flexible possible.

Ces briques de base, ronds et flèches, doivent pouvoir être manipulées à la fois finement et semi-automatiquement.

Par "semi-automatiquement", j'entends par exemple l'alignement sur une grille – ou plusieurs grilles – et la disposition de tout ou partie des états selon certains schémas ; par exemple, voici un automate contenant plusieurs sous-figure régulières : deux en disposition linéaire, et deux en cercles (ou pentagones en l'occurrence).

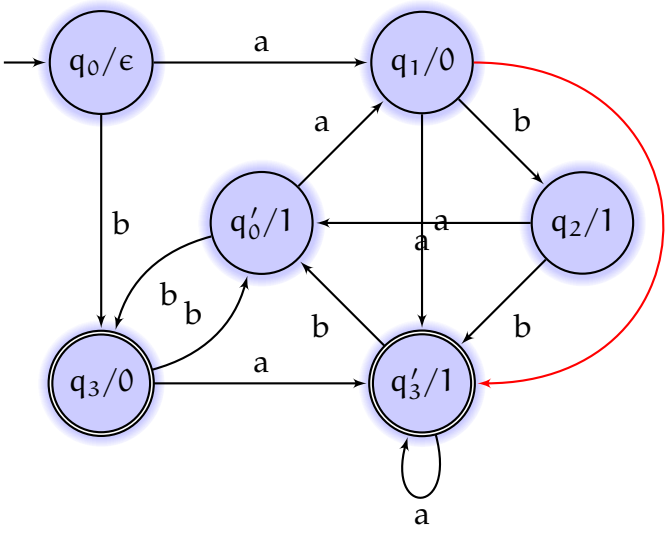


L'interface doit permettre à l'utilisateur de réaliser semi-automatiquement ce type de figure. Spécifiquement l'utilisateur doit être capable de sélectionner des groupes de noeuds et de les arranger selon des figures géométriques régulières telles que des lignes, des cercles, et cetera.

Les arrangements faisant partie d'une telle figure géométrique – que l'on appellera un *groupe* – doivent pouvoir être sélectionnés et déplacés collectivement. On doit aussi pouvoir régler les paramètres de chacun ; par exemple le diamètre d'un groupe déjà défini doit pouvoir être modifié par la suite. De plus, plusieurs groupes doivent aisément pouvoir être uniformisés – par exemple, leur donner le même diamètre, placer leur centre de gravité à des endroits alignés horizontalement ou verticalement, etc.

Les éléments doivent aussi pouvoir être manipulés finement ; par exemple, on a finement déformé la transition de 3 à 10 pour éviter le premier pentagone ; dans l'automate suivant, on doit pouvoir déformer la transition de $q_1/0$ à $q'_3/1$ de manière à éviter le croisement, obtenant ainsi la nouvelle

transition en rouge.

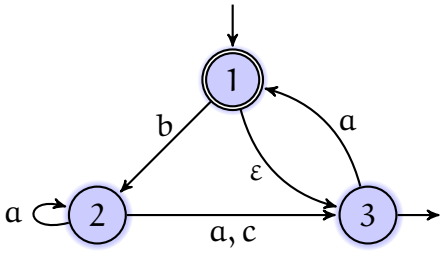


4 Import et export

L'application devra pouvoir sauvegarder et restaurer ses diagrammes dans un format au moins partiellement lisible par l'être humain et proche du formalisme $A = (\Sigma, Q, I, F, \delta)$ – avec bien sur quelques informations supplémentaire d'ordre cosmétique, telles que couleurs, courbure, localisation (x, y), etc.

Au delà de cela, il est absolument essentiel qu'elle soit capable d'exporter au format Ti*k*Z. La qualité de l'export Ti*k*Z est un facteur important, car l'utilisateur aura peut-être envie de retoucher le code généré.

Comme exemple de code Ti*k*Z, l'automate



est codé par

```
\begin{tikzpicture}[fst]
\node[state, initial above, accepting] (1) {$1$};
\node[state, below left of=1] (2) {$2$};
\node[state, below right of=1, accepting right] (3) {$3$};
\draw
  (1) edge[above] node{$b$} (2)
  (1) edge[below, bend right, left=0.3] node{$\eps$} (3)
  (2) edge[loop left] node{$a$} (2)
  (2) edge[below] node{$a, c$} (3)
  (3) edge[above, bend right, right=0.3] node{$a$} (1);
\end{tikzpicture}
```

qui est assez lisible et descriptif. Le paramêtre `fst` (pour Finite-State Transducer) est un style personnalisé défini dans ce document.

Il y a bien d'autres façons de coder cette figure, telle qu'un matrice, ou des positions (x, y) dans le plan explicites. Une grande flexibilité au niveau de la génération du code final est attendue.

À vous de vous documenter sur LaTeX et Ti*k*Z pour voir les différentes possibilités offertes par ces langages. L'application doit en tirer parti autant que faire se peut, afin que le diagramme final puisse être ajusté le plus finement possible avant l'export, et minimiser les besoins en retouches au niveau du code généré.

Au delà de la figure, on doit aussi pouvoir exporter du code LaTeX pour la définition formelle de l'automate sous forme $A = (\Sigma, Q, I, F, \delta)$. Les transitions doivent pouvoir être générées sous forme d'ensemble, comme plus haut, ou sous forme de tableau de transition.

5 Partie automatique

Difficulté élevée (pour obtenir de bons résultats) ; algorithmique poussée ; recherche documentaire ; réflexion poussée. Groupe de 3 recommandé.

Même avec une bonne interface graphique, tous ces choix et ajustements esthétiques prennent énormément de temps. Du point de vue de l'utilisateur final, la partie "dessin manuel" de l'application est donc à réserver pour les figures importantes, qui doivent être réalisées avec une qualité parfaite.

Dans la plupart des cas, l'utilisateur veut juste définir son automate "en vrac" et en quelques clics, appuyer sur un bouton et obtenir instantanément une (ou plusieurs) proposition de diagramme, calculées programmatiquement, qui soient d'une qualité passable. Ceci implique de minimiser les croisements (pas toujours possible à éviter entièrement – on verra au second semestre la notion de graphe planaire, et le fait que tous les graphes ne sont pas planaires), de disposer les états à intervales réguliers de façon à assurer un "niveau de gris" homogène dans la figure, etc.

Il existe de nombreuses techniques spécifiques au dessin automatique de graphes, ainsi que des métaheuristiques générales potentiellement applicables, telles que les algorithmes révolutionnaires / génétiques, le recuit simulé, et les colonies de fourmis, pour n'en citer que quelques-unes.

Mettre en oeuvre ces méthodes nécessitera un gros travail de recherche documentaire et de réflexion poussée pour comprendre des concepts difficiles au niveau 3A.

À ceci s'ajoute une difficulté supplémentaire : il ne s'agit pas de n'importe-quels graphes, mais d'automates ; les conventions de dessin ne sont pas les mêmes ; utiliser un algorithme de dessin arborescent ou fractal donnerait généralement des résultats peu appropriés, même s'ils peuvent être "jolis". On voudra généralement partir de l'état initial, aller plutôt de gauche à droite et de haut en bas, disposer les états sur une grille, sauf certains cycles et autres motifs à détecter et à mettre en exergue, et ainsi de suite.

Il n'y a pas (ou peu) de littérature disponible sur le sujet spécifique "comment dessiner des automates". Les dessins sont faits par les scientifiques "au feeling", influencés par l'habitude et le mimétisme sans que les conventions soient verbalisées.

Dans cette partie du projet, il vous appartiendra d'inférer une partie de ces conventions à partir d'exemples, de les verbaliser, et de les programmer.

Un mode interactif, permettant à l'utilisateur de fournir certaines informations sémantiques à l'application pour guider ses choix (par exemple, ce

groupe d'états va ensemble, cet état est important, etc), ou bien de choisir entre plusieurs alternatives (algorithme évolutionnaire avec sélection (semi-)manuelle, est à considérer.

De plus, ces outils doivent pouvoir être appliqués semi-automatiquement au cours de l'utilisation de l'interface pour faciliter la vie de l'utilisateur – par exemple, en détectant automatiquement quand une nouvelle transition est créée si elle croise quelque-chose, auquel cas elle sera créée courbe et non droite, ou bien en offrant des suggestions d'agencements dans la marge, applicables en un clic, etc.

La partie du rapport consacrée à ces réflexions doit être assez détaillée.

6 Répartition des tâches

Il est conseillé de procéder par groupes de 6, *par exemple* en deux trinômes, un pour la partie manuelle et l'import / export, et l'autre pour la partie automatique.

Dans ce cas, les trinômes doivent évidemment communiquer et intégrer leur code au projet commun de façon très régulière – les tâches ne sont pas indépendantes et la répartition ci-dessus n'est qu'une suggestion.

Il a été noté que l es deux partie sont de difficultés et de natures différentes, la partie manuelle étant colorée "développeur" et l'autre "recherche".

Choisissez bien la partie sur laquelle vous travaillerez en fonction de vos goûts et de vos capacités.

La partie automatique peut potentiellement déboucher sur une très bonne solution (et donc une note stratosphérique) en deux semaines de réflexion et 200 lignes de code,. . . mais vous pouvez tout aussi bien passer un semestre à pondre 10 000 lignes de code produisant une bouillie infâme et inexploitable, de valeur 0.

En revanche, la partie manuelle est difficile à rater si on travaille régulièrement, (mais il faudra pondre beaucoup de lignes que l'on ait les bonnes idées ou pas).

Pour minimiser le risque d'avoir une mauvaise note individuelle sur la partie automatique en cas d'échec, il serait une bonne idée de contribuer un peu à l'interface graphique. Les cas simples comme la détection de croisement sont idéaux pour cela.

Il est bon également que tout membre du groupe ait une vue d'ensemble de l'application, même des parties écrites par les autres.

7 Rapport, évaluation soutenances, et diapos

Les modalités sont sujettes à changement en cours de semestre. Les informations ci-dessous sont non-contractuelles :-P

Il faudra rendre un rapport expliquant vos choix de modélisation, d'implémentation etc, donnant un aperçu des difficultés rencontrées et résolues, et offrant un sommaire de l'investissement personnel de chaque membre du groupe, validé par tout le groupe.

Les compétences et l'investissement individuelles seront testés dans une certaines mesure dans l'examen de Programmation Python.

Les questions posées en soutenance joueront également un rôle.

Dans le rapport, il vous sera demandé de pondérer la quantité de travail (utile) de chacun par consensus du groupe. Par exemple : tout le monde a fourni le même travail, sauf X qui a travaillé 2 fois plus. Ces pondérations affecteront la note individuelle.

Si un consensus ne peut pas être atteint au sein du groupe, nous en discuterons.

Il y aura une journée de soutenances à la fin du semestre, où chaque groupe présentera très rapidement ses travaux, et en fera une brève démonstration. Les modalités exactes sont comme suit :

7.1 Horaires de passage

Le planning sera en ligne sur Celene.

Deux salles seront réservées pendant les soutenances : c'est le jury qui passe d'une salle à l'autre.

De cette manière chaque groupe peut s'installer tranquillement pendant que le groupe précédent soutient dans l'autre salle.

7.2 Timing

Globalement, le principe est le suivant.

Une soutenance est généralement composée de trois types d'interventions :

— Une présentation du travail, préférablement à l'aide de transparents ou 'slides' (Powerpoint, OpenOfficeImpress, Keynote, Beamer (Latex),...).
(P minutes au total)

— Une démonstration doit avoir lieu : elle doit présenter et illustrer votre travail et montrer que ce que vous avez présenté existe bel et bien, et fonctionne !
(D minutes)

— Des questions et de discussion avec le jury.
(Q minutes)

Ces modalités peuvent être monolithiques (on fait toutes les présentations, puis toute la démo, puis toutes les questions) ou être découpées en 2 ou 3 (max) morceaux et alternées les unes avec les autres. Par exemple, chaque binôme présente 4min, puis démontre 5min, puis 5 mins de questions, le tout répété 3 fois, puis quelques questions pour boucler.

Il est aussi possible de faire une présentation/démonstration combinant les deux aspects, i.e. une démo s'aidant de slides.

Dans tous les cas, il importe de bien équilibrer le temps entre les binômes, et entre les individus. **Chacun doit s'exprimer au cours de la présentation / démonstration !**

Vous êtes libres de découper votre temps comme vous le voulez ; les contraintes suivantes doivent être respectées :

$$40 \leqslant P + D + Q < 45$$
$$25 \leqslant P + D \leqslant 30$$

Dans le cas de présentations et démonstrations distinctes, il serait également bon que

$$P \approx D,$$

avec tout de même un peu de flexibilité sur ce point.

Vous pouvez accepter les questions au fur et à mesure (plus agréable, mais plus compliqué à gérer et demande davantage d'aisance à l'oral) ou uniquement à la fin de chaque séquence (plus classique et "scolaire"). Si vous ne précisez pas au début que vous acceptez les interruptions, on restera dans le schéma classique.

Les interventions seront dans tous les cas chronométrées afin d'éviter que les questions ne prennent trop de temps.

Notez que Q couvre à la fois votre temps de réponse et le temps qu'il faut au jury pour exprimer ses questions / remarques. Soyez donc clairs et précis dans vos réponses, et utilisez votre temps de parole efficacement.

7.3 La soutenance

7.3.1 Contenu

L'objectif global est de réaliser un bilan final. Quelques points généraux à aborder :

Quels sont les choix techniques et algorithmiques ? Qu'est ce que le projet à permis de produire ? Quels sont les résultats obtenus ? Insister sur les éléments complexes ou astucieux de votre travail. Expliquer aussi les initiatives prises et les voies explorées pour ce sujet très ouvert, avec les recherches et compréhension des phénomènes complexes associées. Présenter les objectifs atteints et les objectifs à atteindre, les problèmes rencontrés. Une étude critique du travail peut-être formulée : pourquoi le résultat est satisfaisant ou non ?

Bien entendu, il faut s'aider des objectifs exprimés et des éléments de réflexion donnés dans le sujet du projet.

7.3.2 Démonstration

Les démonstrations peuvent se dérouler soit sur l'ordinateur de la salle (celui connecté au vidéo projecteur), soit sur votre ordinateur portable personnel.

Dans tous les cas, évitez les temps morts dus à des contraintes techniques. Utilisez plusieurs ordinateurs si besoin.

Vérifiez aussi la connexion de l'ordinateur servant à la démo avec le vidéo-projecteur avant le jour J.

7.3.3 Conseils pratiques divers

Nous vous conseillons d'utiliser vos ordinateurs portables (si vous en possédez un) pour projeter vos slides et démonstrations. Comme vous n'avez sans doute jamais essayé de projeter avec votre ordinateur, il est impératif que vous fassiez des tests avant le jour J (pour vérifier que vous y arrivez) et pas 5 minutes avant. Si vous avez des problèmes de projection lors de la soutenance, ce sera votre faute ! Le jury n'aime pas attendre à cause d'un problème technique et vous risquez de vous pénaliser face à l'auditoire qui aura l'impression de perdre son temps et celui de vos camarades qui attendent leur tour. (Le système des deux salles réduit la gravité de ce type de problèmes, mais il vaut mieux ne pas en dépendre)

Aucun dépassement de temps ne sera toléré (sinon, tout le scheduling tombe). Le jury vous interrompra si vous dépassez le temps imparti. Evitez aussi de faire une présentation trop courte. Si vous n'utilisez pas tout le temps imparti, le jury risque de rester sur sa faim.

Préparez vos démonstrations à l'avance ! Il ne sera plus temps de recompiler ceci ou cela pendant la démo. Tout doit être compilé et prêt à être exécuté. Prévoyez par exemple plusieurs scripts de démonstration pour vous faciliter la tâche. La démonstration doit être efficace et rapide. Pensez à montrer différents exemples concrets de vos développements pour bien illustrer la soutenance. Surtout, n'essayez pas de commenter le code : le jury attend une démonstration d'un produit fini, pas les détails des lignes de code, même si cet aspect sera pris en compte pour la suite et pour la notation. Ne passez pas de temps à présenter du code à l'écran. C'est particulièrement indigeste, même en étant pédagogue et surtout, vous serez bien plus compréhensible efficace et à l'aise pour présenter un algorithme en langage pseudo-algorithmique que du code. Surtout si vous utilisez des librairies spécifiques. . .

Répétez votre présentation/démo au moins une fois et répartissez-vous correctement vos temps de paroles. Essayez de vous donner la parole de manière naturelle. Évitez un déséquilibre de temps de parole. Pour les questions, évitez qu'il n'y ait qu'un étudiant qui ne réponde à toutes les questions.

7.4 Dépôt du projet sur le serveur Celene

Il est impératif de déposer les versions numériques sur le serveur Celene **au plus tard le jour de la soutenance** :

- Les slides de la soutenance au format PDF

- Les sources de votre projet (dans une archive compressée)

Comme ce travail est un travail de groupe, vous devez déposer ces éléments au nom du groupe.

7.5 Critères d'évaluation des projets

De manière classique, les projets sont évalués en utilisant les critères suivants donnés à titre indicatif et sans prétention à l'exhaustivité :

Qualité technique du projet :

- Quantité et qualité du travail fourni
- Evaluation technique des résultats obtenus par rapport à la difficulté du sujet
- Savoir-faire acquis par les étudiants
- Prise d'autonomie des étudiants
- Qualité technique/algorithmique
- Fonctionnalités de l'outil

Qualité de la présentation :

- Pédagogie de l'exposé
- Prise de recul et expertise face aux questions
- Respect du timing
- Qualité des slides
- Qualité des prises de paroles et explications, aisance orale, répartition des temps de paroles et sujets traités individuellement

Tous ces critères permettent au jurys d'évaluer le projet à l'issue de la soutenance.

Bon courage à tous.