Licence 2 Maths-Info
Université de Marne la Vallée

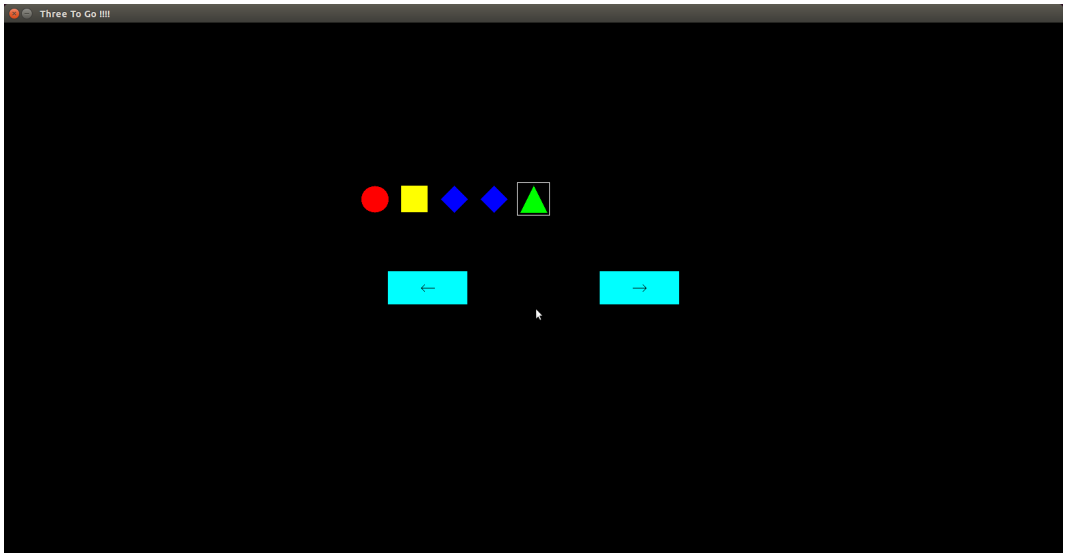
Programmation C

Projet de programmation C : Three to go !!!

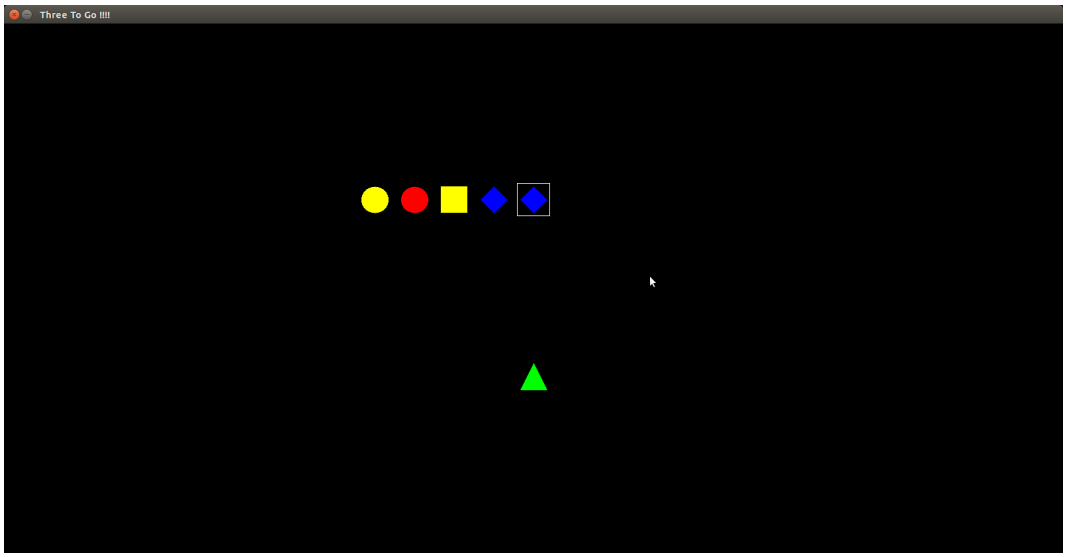
L'objectif de ce projet est de développer une application graphique d'un jeu où l'utilisateur doit produire des motifs répétés avec des petits tokens caractérisés par une forme et une couleur.

Description et déroulement du programme

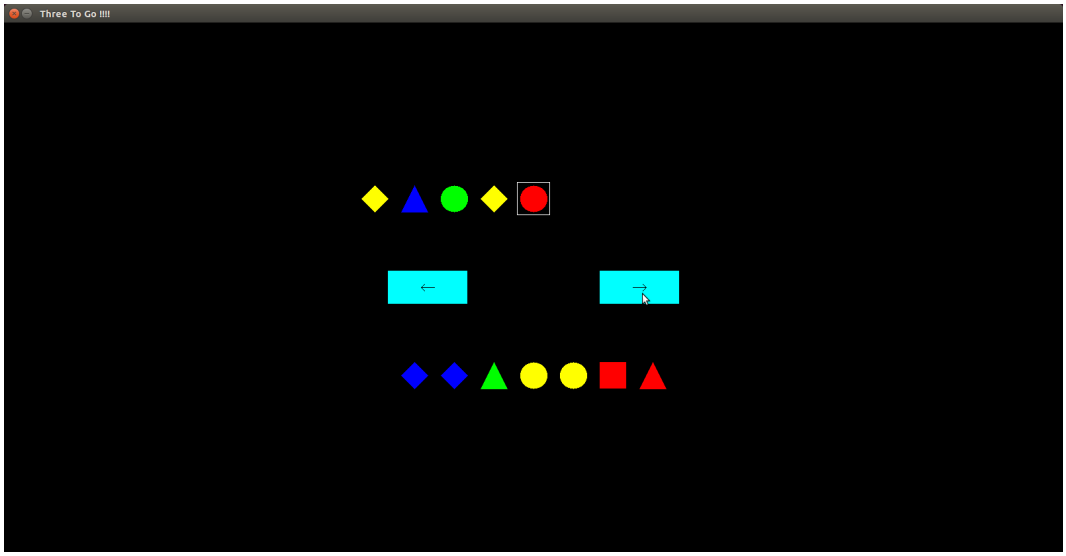
Choisissons quatre couleurs différentes : rouge, vert, bleu et jaune ainsi que quatre formes : rond, carré, triangle et diamant. Un token est un élément ayant une forme et une couleur, on peut ainsi construire seize tokens différents. Lorsque le jeu se lance, le joueur voit une image qui ressemble à l'image qui suit :



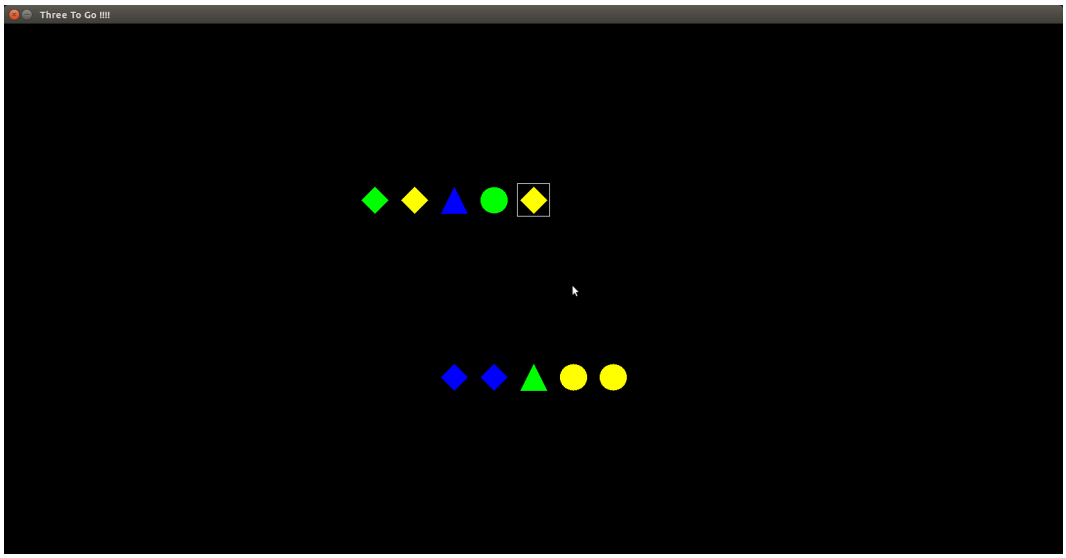
On voit cinq tokens qui ont été générés aléatoirement par le programme dont le dernier est entouré. Ces tokens sont les prochains à insérer, le token entouré est celui que l'utilisateur doit insérer en premier. L'utilisateur doit choisir entre insérer le token au début (bouton flèche gauche) ou l'insérer à la fin (bouton flèche droite). Pour le premier token, les deux cotés reviennent au même.



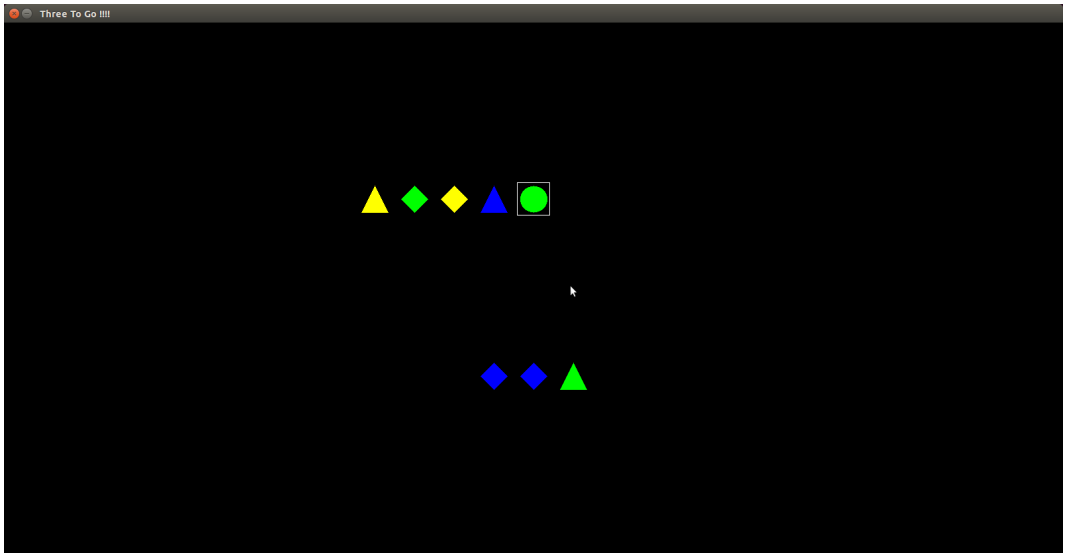
Après avoir insérer quelques tokens, on peut par exemple arriver dans la configuration suivante pour laquelle une insertion à droite va produire trois tokens rouges consécutifs...



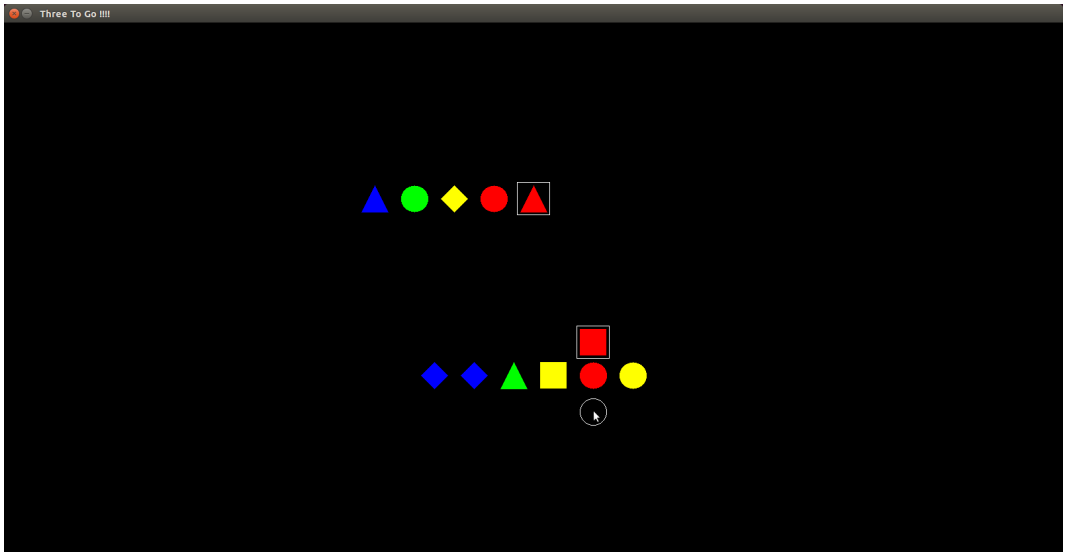
Les trois tokens rouges à droite ont disparu. Encore une fois, l'insertion à droite du prochain token qui est jaune produira encore un motif de trois tokens ayant un caractère en commun (ici la couleur jaune).



Ainsi, la liste de tokens se raccourcit encore et le joueur marque des points.



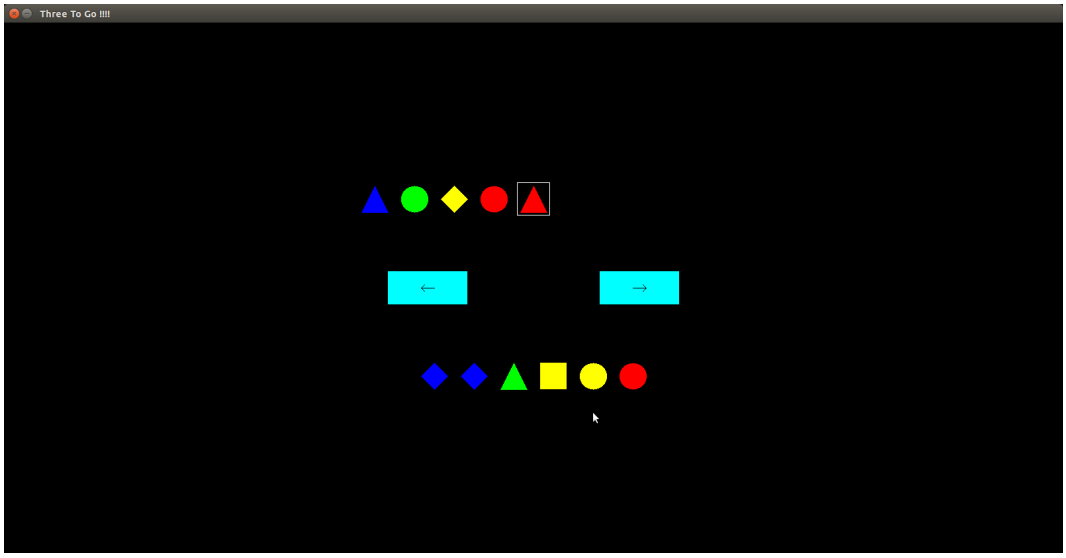
Voilà pour les insertions. Toutefois, le joueur peut créer des motifs disparaissant d'une autre manière plus complexe. On peut cliquer sur un token pour faire une opération de décalage sur les autres tokens ayant la même forme ou la même couleur que celui sélectionné. Prenons la liste suivante lorsque l'utilisateur a cliqué sur l'avant dernier token:



Une fois ce token cercle rouge sélectionné, l'utilisateur peut choisir entre :

- Garder la couleur rouge (bouton au dessus du token choisi) et décaler circulairement les formes des tokens rouges d'un cran vers la gauche.
- Garder la forme cercle (bouton au dessous du token choisi) et décaler circulairement les couleurs des tokens cercles d'un cran vers la gauche.

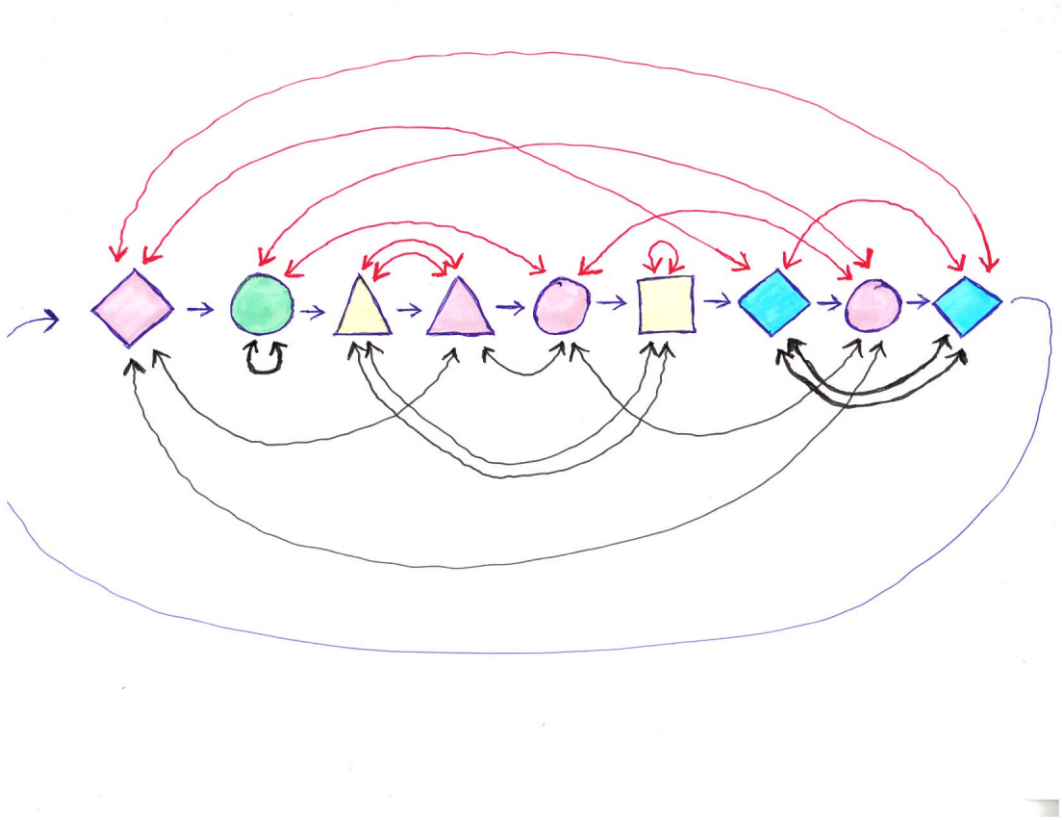
Comme notre liste dans cette exemple ne comporte qu'un seul token rouge, le décalage circulaire des formes ne produirait rien. Par contre, la liste contenant deux cercles, si l'utilisateur garde la forme cercle, la couleur jaune du dernier token va monter d'un cran et la couleur rouge du token selectionné va repartir vers le dernier token cercle de notre liste. Ce qui va produire:



Plus il y a de tokens de même couleur ou de même formes, plus cette opération est difficile à pressentir pour le joueur. Ainsi, les triplets de tokens ayant un caractère commun apparaissant après cette action doivent rapporter plus de points que les insertions classiques.

Chaînages multiples des tokens

Le schéma suivant illustre les chaînages multiples des tokens.



La liste est ici composé en premier d'un diamant rouge, puis d'un cercle vert, puis de deux triangles jaune puis rouge, puis d'un cercle rouge, d'un carré jaune, d'un diamant bleu, d'un cercle rouge et encore d'un dernier diamant bleu.

Le chaînage simple est bleu; c' est l'ordre des tokens dans la liste fabriqué par ajouts et insertions multiples par le joueur.

On voit en rouge le chaînage des formes: les tokens ayant la même forme sont ainsi doublement chaînées circulairement.
En noir, on a le chaînage double des tokens par leur couleur.

Le chaînage simple circulaire avec pointeur vers le dernier facilite les insertions en début ou fin. Les deux chaînages circulaires doubles facilitent les décalages circulaires vers la gauche.

Ainsi, voilà la structure C imposée pour modéliser les tokens:

```
1 typedef struct token{
2   int couleur;
3   int forme;
4   struct token* suivant;
5   struct token* suivant_couleur;
6   struct token* precedant_couleur;
7   struct token* suivant_forme;
8   struct token* precedant_forme;
9 }Tokens, *Liste;
```

Avec comme choix raisonnables pour formes et couleurs:

- Couleur 0 : rouge
- Couleur 1 : vert
- Couleur 2 : bleu
- Couleur 3 : jaune
- Forme 0 : cercle
- Forme 1 : carré
- Forme 2 : triangle
- Forme 3 : diamant

Interface graphique

L'interface graphique est une partie importante du projet. Cette dernière devra être développée avec la bibliothèque graphique C de l'université : la libMLV. Une large documentation à propos de cette bibliothèque est accessible à l'URL :

`http://www-igm.univ-mlv.fr/~boussica/mlv/index.html`

Les machines de l'université en sont équipées.

Ce choix de la libMLV est une obligation. Il y a mieux, il y a aussi moins bien... On attend ici de vous que vous vous familiarisez avec cette bibliothèque écrite par autrui, et que via sa documentation seulement, vous puissiez construire une véritable application graphique. Un projet qui fonctionne est un projet qui compile et fonctionne correctement sur les machines de l'université. La déclaration suivante : "Monsieur, ça marchait chez moi et ça marche pas sur les machines de l'université !!!" ne constitue ni une excuse, ni un argument, ni une justification. L'évaluation de votre projet sera alors très fortement impactée.

Niveaux de rendus

Niveau 1:

- Votre programme fonctionne en console sans interface graphique en utilisant de l'ASCII art pour dessiner les listes:

  ```
  --> D r --> O v --> T j --> T r --> O r --> C j --> D b --> O r --> D b -->>
  ```

  pour la liste correspondant au schéma des chaînages.

- Les insertions en début et fin fonctionnent correctement.
- Les délétions des triplets de tokens consécutifs ayant un caractère en commun fonctionnent correctement.
- Vous allouez et désallouez correctement la mémoire.

Niveau 2:

- Toutes les conditions du niveau 1
- Vous proposez une interface graphique fonctionnelle avec la souris seulement.
- Vous gérez les décalages circulaires via les deux chaînages doubles.

Niveau 3:

- Toutes les conditions du niveau 2
- Vous affichez le score du joueur à l'écran.
- Vous arrêtez la partie à partir de 2 minutes en vérifiant le temps écoulé après chaque clic du joueur, vous affichez le score du joueur en fin de partie.
- Votre jeu est agréable à jouer.

Pour un étudiant en L2, le niveau correct attendu (possiblement 12/20) est le niveau 2 complètement correct et fonctionnel (compilation sans warning avec le flag `-Wall`, gestion de la mémoire ok avec `valgrind` qui doit donner sur votre programme : `All heap blocks were freed -- no leaks are possible.`, ...).

Pour les plus coriaces d'entre vous ayant de l'amour propre, nous vous proposons quelques améliorations raisonnables et sympathiques (Après le niveau 3...):

- (faisable) La partie s'arrête après 2 minutes même si l'utilisateur ne clique sur rien (demande d'utiliser une fonction de la libMLV qui gère la souris et le temps conjointement).
- (faisable) Calculer un score réaliste qui prend en compte la difficulté de faire des combos. Un combo est l'apparition de multiples triplets de tokens qui disparaissent en une action ou bien l'apparition de quadruplets, quintuplets, ...
- (faisable mais long) Faire un tableau des 10 highs scores, sauvegardé dans un fichier, qui se met à jour quand un utilisateur fait un nouveau score parmi les 10 plus grands, l'utilisateur entre alors son nom dans une boite de l'interface graphique.

- (moyen) Votre interface graphique exploite au mieux l'écran de l'ordinateur (taille des tokens affichés, centrage, ...).
- (plus difficile) Faire des niveaux de difficultés où le joueur choisit en début de partie le nombre de formes et le nombre de couleurs.
- Tout ce qui peut améliorer ce projet mais qui reste raisonnable. Lorsque l'on a un doute sur le caractère intéressant d'une amélioration, le mieux est d'en discuter avec les enseignants.

Remarques importantes :

- L'intégralité de votre application doit être développée exclusivement en langage C. Toute utilisation de code dans un autre langage vaudra 0 pour l'intégralité du projet concerné.
- En dehors des bibliothèques standards du langage C et de la libMLV, il est interdit d'utiliser du code externe : vous devrez tout coder vous-même.
  Toute utilisation de code non développé par vous-même vaudra 0 pour l'intégralité du projet concerné.
- Tout code commun à plusieurs projets vaudra 0 pour l'intégralité des projets concernés.

Conditions de rendu :

Vous travaillerez en binôme si possible du même tp. Il faudra rendre au final une archive `tar.gz` de tout votre projet contenant le rapport, les sources de votre application et ses moyens de compilation.

Votre archive devra donc contenir :

- Un fichier `makefile` (récupérer le makefile de la libMLV par exemple) contenant les règles de compilation pour votre application ainsi que tout autre petit bout de code nécessitant compilation (comme les tests par exemple).
- Un dossier `doc` contenant la documentation technique de votre projet ainsi qu'un fichier `rapport.pdf` contenant votre rapport qui devra décrire votre travail. Si votre projet ne fonctionne pas complètement, vous devrez en décrire les bugs. On attend aussi de voir comment le binôme s'est réparti le travail, quels ont été les choix importants durant le développement de votre projet ... Il ne sert à rien de copier-coller votre code dans votre rapport, les correcteurs savent lire vos sources.
- Un dossier `src` contenant les sources de votre application.
- Aucun fichier polluant du type `bla.c~` ou `.%smurf.h%` généré par les éditeurs. Votre dossier doit être propre !
- Si possible, la partie html générée par l'utilitaire `doxygen` à partir de votre application de visualisation. Ceci est optionnel mais tellement plus propre.