Université de Paris **Langages à objets avancés**
M1 **Projet 2021 – 2022**

Projets

Consignes à lire attentivement

Trois sujets sont proposés. Quelque soit le sujet, la livraison doit être sous forme d'une archive au format `tar` ou `zip` portant votre nom (par exemple : `Jack-DALTON.tar`) contenant :

1. un répertoire avec vos sources sous la forme d'un code source en langage C++ compilable *via* un `Makefile` [1],
2. un diagramme UML décrivant l'architecture de classe de votre framework,
3. un rapport de quelques pages présentant ce qui est important,
4. un README facilitant l'installation, sans supposer que nous utilisions tel ou tel IDE.

De plus les programmes doivent être compilables et exécutables sur les machines de l'UFR, bien prendre soin de le vérifier régulièrement. C'est un point attendu.

Le travail est à réaliser en binôme et à déposer sur Moodle avant le xxxxxx. Les soutenances auront lieu la semaine du xxxxxx.

Sujet n°1

Objectif

Obtenir un cadre permettant de mesurer au plus fin la complexité des algorithmes

1. Respecter impérativement cette consigne ! Vous pouvez utiliser n'importe quel outil pour développer mais la livraison doit contenir de quoi générer le projet avec la commande `make`. Une bonne façon de procéder est de simuler la livraison auprès de camarades lesquels devraient pouvoir générer les applications par simple appel à `make` (éventuellement en modifiant le nom ou l'emplacement du compilateur et des bibliothèques). Et si cela fonctionne, de livrer le tout comme il sera demandé ensuite...

Description

Il s'agit d'écrire un ensemble types, types génériques, fonctions génériques, etc, permettant d'écrire des algorithmes pour lesquels une collecte de mesure peut être effectuée durant leurs exécutions. Par exemple (attention cet exemple - simple - n'est pas contractuel, il n'est là que pour l'illustration) :

```
template <typename E>
E max(Array<E> array) {
   E max = array[0];
   for (int i=1; i<array.size(); i++) {
       if (array[i]>max) max = array[i];
   }
}
...
Array<Int> a(10);
...
cout << max(a) << endl;
cout << Int::measures() << endl;
cout << Array<Int>::measures() << endl;
```

dont le résultat pourrait être quelque chose comme :

```
Type Int :
  '=' : 123
  '<' : 12
  'suffix␣++' : 45
Type Array<Int> :
  '[]' : 23
```

De façon plus générale on pourrait imaginer un type générique `Collect<E>` permettant d'obtenir un collecteur de mesure sur le type `E`.

On pourrait aussi imaginer pouvoir collecter des statistiques sur plusieurs exécutions d'un même algorithme afin d'obtenir des complexité mesurées en moyenne.

Sujet n°2

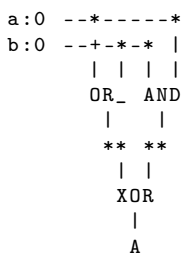
Objectif

Obtenir un simulateur de circuit combinatoire.

Description

Il s'agit d'écrire un ensemble de types/fonctions permettant de créer un circuit combinatoire, c'est-à-dire un circuit sans boucle comprenant des portes

logiques (AND, NAND, OR, NOR, XOR, NXOR, NEGATE, etc) reliées ensembles par des « fils ». L'entrée du circuit est représentée par un ensemble de portes chacune représentant une variable booléenne nommée (par une lettre de l'alphabet minuscule) et contrôlable. La sortie du circuit est un ensemble de porte chacune représentant la sortie d'une porte et est nommée par une lettre majuscule. On prendra soin de pouvoir afficher le circuit sous une forme agréable de type (attention **à ne pas utiliser** d'interface graphique, le mode texte est bien suffisant) :



qui pourrait être obtenu par un code ressemblant à :

```
InputGate *a = new InputGate('a');
InputGate *b = new InputGate('a');
Gate *or = new OrGate(a,b);
Gate *and1 = new AndGate(a,b);
Gate *and2 = new XorGate(or,and1);
OutputGate *A = new OutputGate(and2);
```

La simulation s'effectuera en mode pas à pas, c'est-à-dire qu'à chaque pas l'information franchit au plus une porte, et on doit pouvoir voir dans l'affichage la progression de l'information. On doit pouvoir changer les valeurs des portes d'entrée, etc.

De plus, il devrait être possible d'afficher sous forme textuelle les fonctions de sortie à l'aide des variables d'entrée : $A = xor(or(a, b), and(a, b)$.

Encore mieux, offrir la possibilité de synthétiser le circuit à partir de son expression textuelle.

Autres fonctionnalités : sauver un circuit dans un fichier, le relire, etc. (facile si l'on sait écrire et lire l'expression textuelle d'un circuit).

Sujet n°3

Objectif

Le but de ce projet est totalement révolutionnaire : programmer un jeu. Il s'agit d'un jeu de rôle interactif, ou le joueur contrôle un personnage qui se promène dans un château ou habitent d'autres personnages et qui contient plusieurs objets.

Description

Dans un château il y a plusieurs pièces carrées, avec quatre portes (au plus) dans les quatre directions cardinales. Le château est habité par quatre types de personnages : Moines, Guerriers, Sorcières et Amazones. Le joueur contrôle un des personnages, les autre étant contrôlés par le programme. Chaque personnage a :

- un nom,
- un indice de santé,
- un indice d'habileté,
- un sac qui peut contenir 4 objets au plus.

Les objets sont de types différents parmi lequels :

- bouteilles de médicament et de poison,
- armes et boucliers,
- clés de téléportation.

Le jeu procède en tours. À chaque tour, le joueur choisit de déplacer son personnage a travers une des portes (ou bien de ne pas bouger). Le programme déplace tous les autres personnages de façon aléatoire. Le joueur est présenté avec un description de la pièce : les personnages et les objets présents. Le joueur doit s'engager en combat avec chacun des personnages présents. Le résultat du combat dépend :

- de l'habileté des personnages – des armes et des boucliers
- des types des personnages
- du hasard

Le résultat est un changement des indices d'habileté et de santé, selon des règles a inventer. Après le combat le joueur peut :

- utiliser les objets de son sac,
- poser dans la pièce des objets de son sac,
- prendre de la pièce des objets et les mettre dans son sac.

Le tour se termine quand le joueur choisit où aller.

[Points importants et extensions.] Il faudra faire attention au cas particuliers (les pièces d'angle, la mort d'un personnage quand son indice de santé tombe à 0, la naissance de personnages. Par contre, il ne faudra en aucun cas réaliser d'interface graphique. L'interaction devra être purement textuelle. Il est important de réfléchir à une structure de classes raisonnablement riche. Un schéma UML de cette structure devra être également produit. Plusieurs extension sont possibles, et bienvenues. En particulier, on aimerait rendre possible la sauvegarde sur fichier de l'état d'un jeu, pour pouvoir la recharger en un deuxième temps. On pourrait également inventer des règles plus complexes, ajouter des types de personnage ou animaux etc.