SAÉ S2.02 : Exploration algorithmique d'un problème
Transformations d'objets dans un jeu vidéo

Thématique générale : Modéliser un réseau de transformation d'objets dans un jeu vidéo de différentes façons, résoudre des problèmes d'accessibilité, comparer différentes implémentations.

Sujet : On considère un jeu vidéo de type « bac à sable » où le joueur peut récolter des ressources ou objets, et ceux-ci peuvent être transformés eux-mêmes en d'autres objets. Il se peut qu'on puisse transformer un objet A en un objet B, cette transformation est parfois réversible mais pas toujours. Voir la Figure 1 pour un exemple.

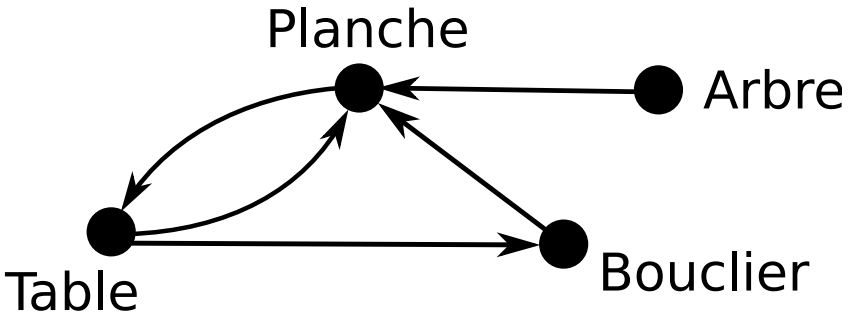

Figure 1: Exemple de réseau de transformation d'objets

On souhaite disposer d'un programme permettant de répondre à la question suivante :

- Est-ce qu'un joueur disposant d'un objet A peut le transformer en objet B (par un nombre quelconque de transformations successives) ?

De plus, on souhaite modéliser le réseau de transformation d'objets de différentes façons afin de comparer l'efficacité de ces implémentations. Pour ce faire, on peut représenter le réseau de plusieurs façons, voici quelques exemples :

1. Une liste d'objets. Chaque objet contient la liste des objets accessibles via une seule transformation directe.

   Avec l'exemple de la Figure 1:
   Objets : {A,B,P,T}
   Objets atteignables en une transformation :
   A : {P}
   B : {P}
   P : {T}
   T : {B,P}

2. Une liste d'objets, et une liste séparée contenant les transformations directes – une transformation directe étant un couple (A,B) d'objets.

   Avec l'exemple de la Figure 1:
   Objets : {A,B,P,T}
   Transformations directes : {(A,P), (B,P), (P,T), (T,B), (T,P)}

3. Les n objets sont numérotés de 0 à n-1, et les transformations directes sont stockées dans une matrice (tableau à deux dimensions) où la case (i,j) contient 1 s'il y a une transformation directe de l'objet n°i à l'objet n°j, et 0 sinon.

   Avec l'exemple de la Figure 1:
   Objets : {A=0, B=1, P=2, T=3}

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

Travail demandé :

- ***Modélisation des réseaux.*** Modéliser les réseaux de transformation en C++ et en utilisant les paradigmes de la programmation orientée objets. Il faudra le faire d'au moins deux façons différentes parmi celles proposées ci-dessus (ou d'autres vous semblant pertinentes) afin de pouvoir les comparer par la suite. Votre code sera modulaire, pour que votre algorithme fonctionne indifféremment avec les deux implémentations de réseau choisies. Pour cela, vous créerez une classe représentant un réseau. Chaque implémentation de réseau disposera d'une fonction permettant d'obtenir la liste des objets pouvant être créés par une transformation directe depuis un objet A donné.
  Pour chaque implémentation de vos réseaux, vous devrez concevoir les structures de données adaptées. Vous pourrez utiliser des listes, des tableaux... : à vous de choisir.
- ***Algorithme d'accessibilité.*** Développer l'algorithme d'accessibilité demandé. Attention, comme le réseau peut avoir des cycles, pour éviter de boucler indéfiniment, il faudra trouver un moyen pour que l'algorithme sache quelle partie du réseau il a déjà explorée.
- ***Tests.*** Tester votre programme sur des exemples de réseau suffisamment complexes (entre 10 et 20 objets par exemple).
- ***Rapport.*** Rédiger un rapport décrivant les choix d'implémentation et répondant aux questions détaillées plus bas (voir la section « Livrables »).

Organisation : Vous travaillerez par groupes de quatre étudiants. Les groupes ne sont pas libres mais formés par l'équipe enseignante.

Livrables : À l'issue de cette SAÉ, les différents livrables que vous devez fournir sont :

- Le code de l'application avec sa documentation. Vous détaillerez le travail effectué par chaque membre du groupe. Vous écrirez aussi dans les commentaires du code, qui a implémenté telle ou telle classe/méthode.
- Un rapport au format PDF contenant :
  1. Une description des implémentations du réseau que vous avez choisies, avec le détail des structures de données utilisées.
  2. Une explication des algorithmes développés pour résoudre le problème d'accessibilité. Cette explication doit être compréhensible par un non-informaticien. Vous pouvez illustrer cette explication par un schéma représentant un exemple de petit réseau, et dérouler les étapes de l'algorithme sur cet exemple.
  3. Une comparaison de la complexité de l'algorithme, en fonction des deux méthodes de représentation du réseau que vous avez choisies. Par complexité on entend le nombre d'opérations effectuées, au pire des cas, en fonction de la taille du réseau (la taille est le nombre d'objets et de transformations directes). A votre avis, laquelle des deux méthodes choisies est plus pertinente ? Pourquoi ?
  4. A votre avis, l'algorithme que vous avez fourni peut-il résoudre un ou plusieurs autres problème(s) similaire(s) dans un autre contexte ? Si oui lequel ?

  Attention, il est important de justifier et d'expliquer toutes vos réponses.

Évaluation :

Qualité et complexité du code et de la documentation : 6 points
Compte-rendu, partie 1 : 4 points
Compte-rendu, partie 2 : 5 points
Compte-rendu, partie 3 : 4 points
Compte-rendu, partie 4 : 1 point

Modalités de rendu : Vous rendrez une archive intitulée NOM1-NOM2-NOM3-NOM4.zip contenant le code et le rapport au format PDF, nommé NOM1-NOM2-NOM3-NOM4.pdf, avant le vendredi 4 mars 2022 à minuit :

- sur Karuta (chaque étudiant dépose l'archive dans son portfolio),
- ainsi que par e-mail à votre enseignant du cours R201 - Développement orienté objets (un seul envoi par groupe). Précisez vos noms dans l'objet de l'e-mail.

Compétence et apprentissages critiques concernés : Compétence 2 (Optimiser des applications informatiques / Appréhender et construire des algorithmes)

- Apprentissage Critique (AC1) : Analyser un problème avec méthode (découpage en éléments algorithmiques simples, structures de données...)
- Apprentissage Critique (AC2) : Comparer des algorithmes pour des problèmes classiques (tris simples, recherche…)
- Apprentissage Critique (AC3) : Expérimenter la notion de compilation et les représentations bas niveau des données
- Apprentissage Critique (AC4) : Formaliser et mettre en œuvre des outils mathématiques pour l'informatique