RIGARD Lobsang
RIVIERE—JOMBART Diego

COMPTE RENDU SAE 1.2

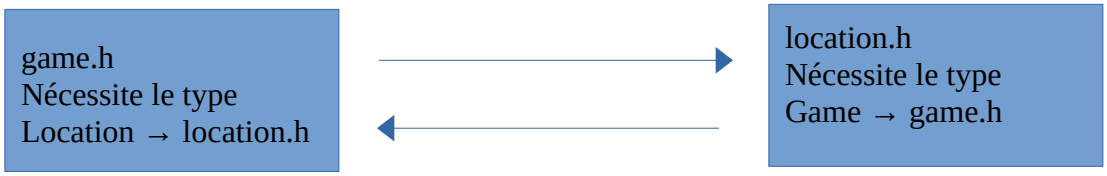
WORLD OF IUT

RIGARD Lobsang
RIVIERE—JOMBART Diego

Problèmes Rencontrés

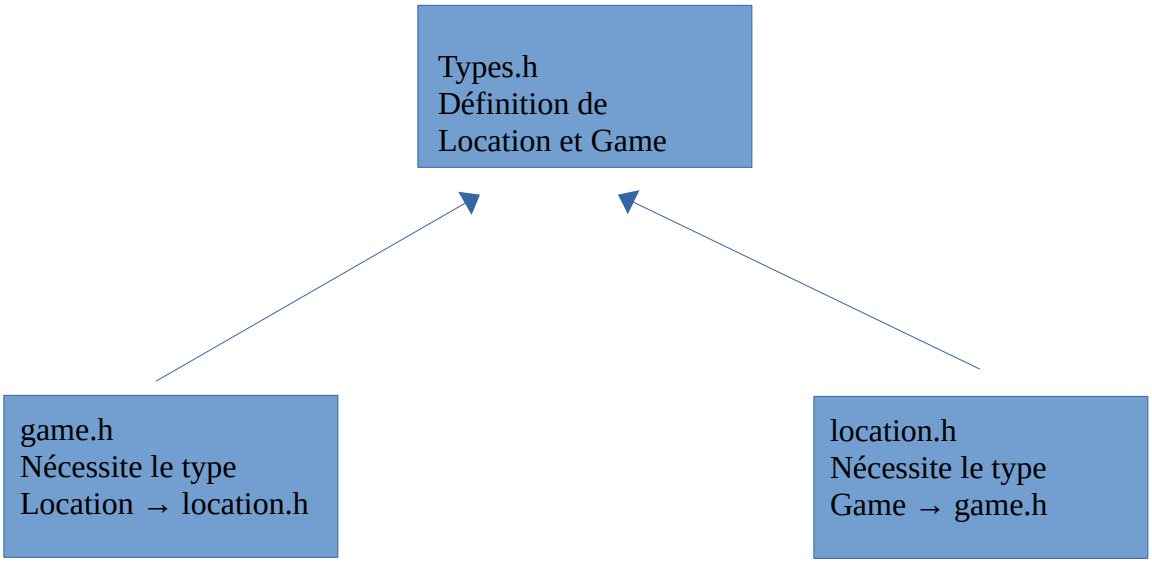
Importations Circulaires

Lors de l'implémentation de certains header files, il est nécessaire d'importer d'autre headers contenant la défin,ition e certains types or il est possible que ces header aient également besoins du type définitions contenues dans le header l'appelant, créant donc une boucle d'importation représentable par le schéma suivant



Solution

Une solution possible et celle que nous avons adopté est d'extraire les types problématiques dans un autre fichier header (types.h dans notre cas) On se retrouve donc avec le schéma suivant



RIGARD Lobsang
RIVIERE—JOMBART Diego

Lecture de fichier & Parsing des arguments

Dans notre programme, nous avons réalisé la fonction readFile prenant en paramétre un fichier et retournant une liste de location peuplée ou non d'objets, la fonction parseAndExecute ne prenant en compte que le premier argument et mettant de coté le reste de la ligne, pour des objets à nom composées d'un espace, cela pose un problème car la seconde partie du nom de l'objet est traité comme un argument.

Solution

Pour répondre à ce problème, nous avons modifé le code de parseAndExecute pour que les commandes get et drop recoivent la totalité de la ligne et pas seulement le premier argument.

Allocation / Désallocation des Objets

La solution la plus évidente aurait été de a l'instar des locations d'ajouter tout les objets dans un stack puis de tous les désallouers, cependant nous avons choisis une approche différente, les objets étant forcément soit dans un inventaire soit dans une location et n'étant pas duplicables, il ne nous semblait pas nécessaire de recréer une structure stack entiére uniquement pour cela ou d'adapter la précedente. En effet, au moment de désallouer le joueur et les pièces, nous désallouont les objets y étant liées.

Limitation du lecteur de fichier

Dans le cas ou @objet ou @room n'aurait pas un identifiant, ou qu'il ne serait pas unique ou qu'il manquerai n'importe quel partie de la data le programme, entrerait en erreur, le rendant inutilisable.
Il est donc obligatoir de vérifier la justesse et la completion des données afin de garantir le bon fonctionnement du programme.