Projet de programmation L2
Octopunks

Flavien Breuvart

13 novembre 2023

Principe Le projet de cette année consiste à implémenter votre propre version de Exapunks. Il s'agit d'un jeu de programmation, ce sont des jeux de puzzles que l'on doit résoudre en programmant des robot/machines vituelles dans un langage dédié. Dans Exapunks, vous programmez, dans un assembleur adapté, de petits robots capables de se dupliquer et d'interagir pour accomplir des tâches divers.

Langage Le projet est en *Java + Swing* (bibliothèque graphique de *Java*).

Taille des groupes Le projet est gros et vous devez faire des groupes de 4 personnes ou plus.
Un canal *mattermost* sera ajouté pour les annonces de recherche de groupe/membres.
Chaque membre du groupe doit être responsable d'une tâche du projet, ce qui ne signifie pas que le responsable ne travaille que sur cet aspect, car ceux-ci sont inégaux en terme de travail à fournir. Chaque personne supplémentaire dans le groupe rajoutera une tâche, mais les tâches supplémentaires sont plus simples que les précédentes.

Remerciement Le projet a pu se faire grâce à la compagnie Zachtronics, qui nous autorise à utiliser leur concept et fournit des licences académiques gratuites.

1 Tâches

Les tâches suivantes doivent être effectuées, avec un responsable pour chaque phase. Les 4 premières sont obligatoires pour tous, les trois suivantes le sont que pour des groupes de taille 5, 6 voire 7 :

- 4 et + : coordination et vision globale,
- 4 et + : l'interface graphique avec les contrôles demandés,
- 4 et + : le modèle mémoire autorisant les actions assembleurs possibles uniquement,
- 4 et + : l'ouverture de niveau format imposé, ainsi que la lecture de l'assembleur,
- 5 et + : la gestion des évènements asynchrones (kill, send...),
- 6 et + : la gestion des méta-fichiers à l'intérieur du jeu,
- 7 : La "Gamification" (gestion des conditions de victoire et de la gestion du classement).

Ces tâches sont détaillées dans les annexes dédiées, le responsable de chaque tâche doit connaître son annexe. Mais avant ça, vous devez bien avoir en tête que personne ne doit se cantonner à ses propre tâches : la répartition du travail décrite ci-dessus est à peu près équilibrée pour un travail valant une note de 10. Lorsque vous irez plus loin, les rôles 2, 4 et 6 auront plus de travail. Il vous faudra alors travailler tous ensemble.

2 Évaluation

Rendu d'initialisation le 10 janvier Vous devrez, avant le 10 janvier :

- avoir trouvé les membres de votre groupe et avoir fait une première réunion,
- avoir créé un canal mattermost privé en invitant les membres de votre groupe et votre enseignant,
- y avoir déposé un "diagramme des classes" prévues et un "diagramme de Gantt" de ce que vous envisagez de faire.

Les diagrammes n'auront pas à être respectés, le but est de structurer vos idées. Par contre il faudra expliquer tous les changements apportés dans le rendu final.

Rendu final en mars À la fin du projet, il vous faudra rendre le code produit (toutes leurs versions, qu'elles soient finies ou pas), un *makefile* et/ou un *readme* pour le compiler et le lancer le jeu qui soient utilisable sur *linux*. On demande aussi un rapport **c**ommun au format `.txt` ou `.pdf` contenant :

- la répartition des rôle et l'organisation du travail au sein du groupe,
- le "diagramme des classes" final, en précisant les changements sur l'initial,
- le "diagramme de Gantt" effectifs, en précisant les changements sur l'initial et les versions intermédiaires,
- un listing des bugs encore présents (un bug connu est moins grave qu'un bug ignoré),
- une auto-évaluation des point forts et points faibles de votre travail,
- un listing des difficultés principales rencontrées par le groupe,
- la description d'une solution (même bancale) utilisée pour l'une des difficultés,

et un rapport **i**ndividuel au format `.txt` ou `.pdf` d'une ou deux pages chacun contenant :

- une description de votre contribution et de vos interactions avec vos camarades,
- une réflexion libre sur ce que vous auriez fait différemment si vous aviez recommencé le projet au moment de la rédaction du rapport,
- une dernière partie libre sur votre retour d'expérience.

Le rendu du projet se fait via un dépôt git sur lequel seront invités les enseignants et tuteurs et dont le lien sera donné dans le canal mattermost. Le rapport commun sera donné sur le canal mattermost. Les rapports individuels seront envoyé via mattermost mais en privé.

soutenances Vous aurez une à deux soutenances en groupes : la première, optionnelle, aura lieu au début du semestre, la seconde, obligatoire, aura lieux après les partiels de mi-semestre. J'y poserai des question sur votre avancement et vous présenterez brièvement ce que vous avez fait. Il vous sera aussi demandé un rapport joint et un rapport par personne. Ce dernier est très important (peut être plus que le rendu technique) et vous interrogera sur votre implication dans le projet.

2.1 Annexe : Coordinateur

C'est un rôle qui nécessite des capacités humaines et d'organisation. C'est aussi le rôle avec les tâches les plus diverses, s'il doit comprendre ce que chacun fait, par contre, la technique impliquée est moindre. Il lui faudra mettre en place tout ce qui permettra aux membres du groupe de travailler en synergie.

En particulier il devra créer un groupe privé *Mattermost* pour interagir avec les enseignants (vous faites ce que vous voulez entre étudiants) et un dépôt *Git* pour le rendu du projet. Ce dernier sera utilisé tout au long du semestre et son historique sera consulté pendant la soutenance.

Il lui faudra organiser les réunions régulières (physique, virtuelles et partielles) et récupérer des comptes rendus de celles-ci. S'il n'a pas pour vocation d'imposer les dead-line, il doit vérifier que chacun en a et qu'elles soient cohérentes entre elles.

Le coordinateur doit comprendre le code abstrait (méthodes publiques et structure choisies) de chacun affin de pouvoir aider à l'intégration des codes entre eux. Il sera aussi en charge d'écrire des tests et les faire tourner régulièrement pour repérer les bugs au plus tôt possible.

Attention, être coordinateur ne signifie pas être le chef, au contraire : le coordinateur doit essayer de répondre aux exigences de chacun. Il est aussi invité à participer, de manière secondaire, à chacun des codes des autres membres pour avoir une vision global de l'état du projet.

Remarque : Si vous pensiez avoir moins à lire en étant coordinateur, détrompez-vous, il vous faut maintenant lire le rôle de chacun des membres de votre groupe pour pouvoir les accompagner ;-)

3 Annexe : Responsable Graphique et contrôles

C'est un rôle qui nécessite un peu d'aisance en programmation objet et une capacité à fouiller dans la documentations des bibliothèques de Swing pour trouver comment faire chaque petites choses.

La qualité esthétique n'est pas très importante (on n'est pas graphistes), mais sera prise en compte.

3.1 Objectif 10

Dans un premier temps, on doit pouvoir afficher :

- une zone de code où le joueur pourra écrire le code du ou des robots disponibles, [1] avec une barre de déplacement (de bas en haut) dans le code,
- les deux zones de mémoire de votre robot,
- une zone avec les boutons avec
  - un bouton "pas" qui lance le robot sur une instruction assembleur et verrouille la zone de code,
  - un bouton "stop" qui réinitialise le robot et réactive la zone de code.
- une zone de jeu qui contient les plateformes sur lesquelles vous allez jouer et où vont s'exécuter les déplacements du robot.

3.2 Fonctionnalités graphiques supplémentaires

Pour avoir plus, voici les comportements à implémenter, dans l'ordre de votre choix. Certaines tâches sont à faire en interaction avec vos camarades, et sont prioritaires lorsque ceux-ci en sont à ces points, elles sont indiqués par une ou plusieurs lettres entre parenthèses ('G' pour graphique, 'M' pour métier, 'T' pour textuel, 'C' pour communication, 'F' pour fichier et 'E' pour enregistrement), le coordinateur n'est pas indiqué car il est toujours présent dès qu'il y a interaction entre des membres, si des membres non-présents sont évoqués (car votre groupe est petit), ces tâches ne sont pas demandées :

- La zone de boutons doit contenir les boutons suivants (vous pouvez utiliser des icônes) :
  - "pas" qui lance le robot sur une instruction assembleur
  - (M) "automatique" qui lance le robot sur une instruction automatiquement jusqu'à résolution du puzzle,
  - (M) "pause" qui stoppe l'exécution automatique,
  - (M) "stop" qui réinitialise le plateau et replace en mode d'écriture,
  - "+" et "-" qui augmentent ou diminuent la vitesse d'exécution automatique.
- (T,M) Un fichier de niveau peut être chargé et analyse au lancement du jeu.
- Les boutons de la zone doivent avoir leur raccourci clavier.
- Le jeu doit s'ouvrir sur un menu pour choisir le niveau.

1. à partir de 5 il faut 2 robots

— On doit avoir un moyen de sauvegarder un code assembleur pour le retrouver en relançant le programme. Vous pouvez aussi enregistrer les meilleurs scores/solutions.
— Un évènement doit indiquer que le puzzle a été résolu.
— (M) Le robot peut se dupliquer et créer une seconde zone de code.
— (F,M) S'il y a des meta-fichiers, leur contenu s'affiche dans la zone avec les codes des robots. Les méta-fichiers différents doivent s'afficher différemment.
— On doit pouvoir bouger les zones de codes et de méta-fichiers.
— Pendant l'exécution, on doit pouvoir cliquer sur un robot, ce qui nous place sur le code de ce robot.
— (T) Les meta-instructions sont remplacés par les vrais instructions lorsque vous exécutez le code (pas à pas ou automatiquement).
— (M) Le code de vos camarades doit pouvoir stopper l'exécution automatique ('!' dans l'assembleur ou zone bloquante).
— (M) Il doit être possible de double-cliquer sur une zone pour la rendre bloquante, ce qui interromprait l'exécution quand un robot entre dans la zone.
— On doit pouvoir accélérer au delà du rafraîchissement de l'ordinateur et n'affichant qu'un mouvement sur 10, sur 20, sur 40, etc... (attention à bien interrompre quand il faut)
— On doit avoir des musiques.
— (C) Lancement de tests aléatoires sur les niveaux.

4 Annexe : Responsable du code métier

C'est le rôle qui nécessite le plus de rigueur et d'aisance en terme de code Java, il vous faudra faire plein de choix de design et bien manipuler vos structures. Le code résultant est critique car central, mais ce n'est pas pour autant le plus difficile. [2]

C'est un rôle qui fera le lien entre la partie textuel et la partie graphique, d'où les nombreuses interactions, c'est pour ça qu'il est central. À 5 ou plus, le responsable métier devra collaborer de manière privilégiée avec le responsable des commandes asynchrone (on peut considérer qu'il s'agit d'un binôme).

4.1 Objectif 10

Attention, l'objectif change légèrement en fonction du nombre de membres dans le groupe.

- Si vous êtes 4 : On considère qu'il y a un robot tenant un fichier (avec juste un entier) qui se déplace sur une grille carrée 5x5 de territoires avec une place. Aucune condition de victoire à vérifier.
  Les instructions à implémenter sont : `COPY`, `ADDI`, `MULI`, `SUBI`, `JUMP N`, `FJMP N` et `LINK`. Pour les instruction de jump (`JUMP`, et `FJMP`) on demande un entier plutôt qu'un label, il faut sauter autant de lignes qu'indiqué par cet entier (qui peut être négatif), c'est plus simple à implémenter que le label. La lettre `M` peut être utilisée en lecture uniquement pour lire l'unique entier du fichier.
- Si vous êtes 5 : On considère qu'il y a deux robots avec chacun un fichier chacun (avec un entier) qui se déplacent sur une grille carrée 5x5 de territoires avec une place. Condition de victoire simple à vérifier (robot sur la case opposé).
  Les instructions à implémenter sont les mêmes qu'à 4, il n'y a pas d'instruction de communication car on n'utilise que le mode global et la lettre `M` pour indiquer que l'on transmet/reçoit un message envoyé.
- Si vous êtes 6 ou plus : On considère qu'il y a deux robots qui se déplacent sur une grille carrée 5x5 de territoires avec deux places. Condition de victoire simple à vérifier (robot sur la case opposé).
  Les instructions à implémenter sont les mêmes qu'à 5 plus `TEST EOF`, `GRAB` et `DROP` ; Il devra aussi être possible d'utiliser la lettre `F` pour lire ou écrire dans un fichier. Remarquez que le test est juste sur `EOF` pour savoir si on est à la fin d'un fichier.
- Si vous êtes 7 : On considère qu'il y a deux robots qui se déplacent sur une grille carrée 5x5 de territoires avec deux places. Vérifier des conditions de victoire complexe et transmettre les données de la partie à votre camarade chargé des enregistrements.
  Les instructions à implémenter sont les mêmes qu'à 6.

Dans chaque cas, il ne vous sera demander que d'utiliser une carte unique préchargée (directement dans le code ou autrement) consistant en une grille 5x5

2. Contrairement au projet de l'an dernier.

avec des zones accessibles et des zones occupées, et des portes numérotées de 0 à 3. Le responsable de la partie graphique, via de bouton d'action pour lancer le robot, va appeler le parser de votre camarade responsable des entrées-sorties, puis votre programme d'exécution qui mettra en place le robot. Ensuite, à chaque pas, il vous faudra calculer la possibilité du mouvement et l'exécuter. Remarque : un robot qui fait un mouvement impossible meurt.

4.2 Fonctionnalités métiers supplémentaires

Pour avoir plus, voici les comportements à implémenter, dans l'ordre de votre choix. Certaines tâches sont à faire en interaction avec vos camarades, et sont prioritaires lorsque ceux-ci en sont à ces points, elles sont indiqués par une ou plusieurs lettres entre parenthèses ('G' pour graphique, 'M' pour métier, 'T' pour textuel, 'C' pour communication, 'F' pour fichier et 'E' pour enregistrement), le coordinateur n'est pas indiqué car il est toujours présent dès qu'il y a interaction entre des membres, si des membres non-présents sont évoqués (car votre groupe est petit), ces tâches ne sont pas demandées :

- (T) Un fichier de niveau peut être chargé et analysé au lancement du jeu.
- (T,G,E) Celui-ci peut contenir une condition de victoire à vérifier.
- (G) Le robot peut se dupliquer sur l'action `REPL` et se détruire sur l'action `HALT`.
- Les commandes arithmétique : `NOOP`, `DIVI`, `MODI`, `SWIZ` et `TEST`.
- (C) Les commandes de communication : `KILL`, `MODE`, `TEST MRD` et `VOID M`.
- (F) Les commandes de fichiers : `SEEK`, `MAKE`, `WIPE` et `VOID F`.
- (G) Le redémarrage en position initiale.
- (G) L'exécution automatique du code en boucle.
- (G,T) L'arrêt de l'exécution automatique en cas d'intervention (presser 'C' ou le bouton d'arrêt), ou de balise dans le code ('!') ou sur une région (clique de l'utilisateur)
- La possibilité de niveaux plus divers (plus de place dans une zone/*host*, graphe plutôt qu'une grille, etc).

5 Annexe : Responsable entrées-sorties textuelles

C'est un rôle qui nécessite de bien maîtriser le concept de langage assembleur (vu en architecture et système). Il vous faudra aussi comprendre comment lire des strings depuis plusieurs sources et les convertir.

5.1 Objectif 10

Il y a deux tâches à faire :

- Une version textuelle du jeu (pour le tester avant la version graphique), [3] dans laquelle on entre un code assembleur puis qui exécute un pas du robot à chaque fois que l'on appui sur entré (en affichant le niveau à chaque fois).
- Le "parsing" du code assembleur que l'on découpe en ligne (fonction `split(\n)`), dont on récupère le nom et les arguments de chaque instruction, pour transmettre au code métier une liste d'objets représentant les instruction (utilisez une classe/enum "instruction").

Attention : Ce rôle est relativement simple, mais urgent, cette tâche doit être effectuée très vite afin de permettre a chacun de tester son code. Le responsable ne doit donc pas hésiter à débaucher ses camarades au début en leur demandant de l'aider. En contrepartie, cette tâche finie, le responsable pourra aider les autres.

5.2 Fonctionnalités I/O supplémentaires

Pour avoir plus, voici les comportements à implémenter, dans l'ordre de votre choix. Certaines tâches sont à faire en interaction avec vos camarades, et sont prioritaires lorsque ceux-ci en sont à ces points, elles sont indiqués par une ou plusieurs lettres entre parenthèses ('G' pour graphique, 'M' pour métier, 'T' pour textuel, 'C' pour communication, 'F' pour fichier et 'E' pour enregistrement), le coordinateur n'est pas indiqué car il est toujours présent dès qu'il y a interaction entre des membres, si des membres non-présents sont évoqués (car votre groupe est petit), ces tâches ne sont pas demandées :

- (M) Un fichier de niveau peut être chargé et analysé au lancement du jeu.
- (M,G,E) Celui-ci peut contenir une condition de victoire à vérifier.
- (C) Le fichier de niveau peut être paramétré de manière à générer des niveaux aléatoires.
- (G,M) La détection de la balise '!' en début d'instruction signifiant que l'exécution automatique doit être interrompu lorsque l'on arrive à ce point.
- La gestion des méta/macro-instructions :

3. Voir même après car vous pourrez alors afficher des infos supplémentaires pour le débugguage

— (G,M) La méta-instruction `MARK` permet de définir un label dans le code, ce n'est pas une instruction, mais plutôt un paramètre de l'instruction suivante ou un numéro de ligne permettant de faire un jump vers ce label.
— (G) La méta-instruction `NOTE` permet de faire un commentaire, ce n'est pas une instruction et elle ne doit pas être transmise au code métier (elle peut éventuellement être transmise au code graphique).
— (M) les macro-instructions `@REP N`, `@END` et `@{M,N}` permettent d'écrire plein de lignes de manière plus dense. Ces macro doivent être étendues pour dupliquer du code dès l'étape de parsing, le code métier ne doit voir que la séquence final de vrais instructions.

6 Annexe : Responsable des (simulations de) commandes asynchrones

Le responsable de ce rôle va devoir explorer les différentes notions de hasard et de concurrence en informatique. On y trouvera des liens avec les cours de SDA mais surtout de S&R. Comme pour la partie graphique, vous allez explorer quelques bibliothèques mais avec des concepts plus théoriques et moins de détails techniques.

Il vous faut d'abord apprivoiser la classe `Random` de Java. Celle-ci permet de créer un générateur aléatoire à partir d'une graine. L'idée est de partir d'un entier quelconque pour initialiser une fonction qui fait "n'importe quoi"[4] et fournit des valeurs successives qui ont l'air d'être aléatoires ; on parle de psedo-aléatoire lorsqu'il n'est pas possible de le distinguer d'un vrais aléa. On utilise un tel générateur pour deux raisons : 1) l'aléatoire réel n'existe pas en terme algorithmique, on se contente donc toujours du psedo-aléatoire, et 2) cela permet de reproduire exactement une trace d'exécution même si elle contenait des choix aléatoire, ce qui est utilisé pour déboguer ou, comme dans notre cas, pour faire passer des tests systématiques et comparables.

6.1 Objectif 10

On dispose de deux robots dont l'ordre d'exécution doit être (psedo-)aléatoire, c'est à dire qu'à chaque pas d'exécution on tire au sort celui qui va agir en premier. C'est très important car leurs actions peuvent ne pas être compatibles.

Pour le projet minimal, il n'est rien demandé de plus, par contre, il sera demandé au responsable des communications asynchrone de se mettre en binôme avec le responsable métier, notamment pour le code de tout ce qui est interaction entre les robots. On parle ici de binôme, car, contrairement aux autres rôles, ces deux rôles vont a priori travailler sur les mêmes classes et donc le même code.

6.2 Fonctionnalités concurrentes supplémentaires

Pour avoir plus, voici les comportements à implémenter, dans l'ordre de votre choix. Certaines tâches sont à faire en interaction avec vos camarades, et sont prioritaires lorsque ceux-ci en sont à ces points, elles sont indiqués par une ou plusieurs lettres entre parenthèses ('G' pour graphique, 'M' pour métier, 'T' pour textuel, 'C' pour communication, 'F' pour fichier et 'E' pour enregistrement), le coordinateur n'est pas indiqué car il est toujours présent dès qu'il y a interaction entre des membres, si des membres non-présents sont évoqués (car votre groupe est petit), ces tâches ne sont pas demandées :

— (M) Les commandes de communication : `KILL`, `MODE`, `TEST MRD` et `VOID M`.

4. Ce "n'importe quoi" est en fait bien choisi pour être uniforme et non réversible, comme du vrais aléa. Vous aurez un cours en M1 qui traite en détail ces générateurs.

— (G,T) Niveaux tests générés aléatoirement selon une définition stricte pour être lancé sur un même code. [5] Mieux encore : permettre d'écrire dans un fichier un niveau avec des variables aléatoires et une graine, ce qui génère toujours les mêmes variantes.
— Le jeu est implémente une asynchrone avec synchronisation régulière, c'est à dire que tout le monde s'attend à chaque pas. Ce n'est pas un comportement réaliste, en réalité les robots/processeurs sont plus asynchrone que ça, on va essayer d'avoir d'autres modes plus proche ou plus éloignés de l'asynchronie :
  — Mode synchrone : prédéfinissez un ordre d'exécution à l'aide d'une réimplémentation du pid d'un robot ; chaque robot commence avec un PID, et chaque duplication fournit un pid fils unique et déterminé déterministiquement.
  — Mode asynchrone discret simultané : à chaque pas, l'ordre entre les robots est déterminisé psedo-aléatoirement, et, en plus, chaque robot a une chance fixe (par exemple 1/2) de ne pas faire de pas (car il n'aurait "pas eu le temps").
  — Mode asynchrone discret : un seul robot, tiré psedo-aléatoirement, peut agir à chaque pas.
  — Mode asynchrone discret biaisé : idem mais les chance d'être tiré au sort augmentent quand on n'est pas tiré et diminue si on l'est.
  — Mode asynchrone réel : chaque robot est assigné à un *thred* différent, il vous faut alors gérer les conflits avec de la synchronisation... Cette version est dure, il faut importer les connaissances de S&R et se taper la documentation de Java ! Remarquez que l'on perd la reproductibilité, car l'aléatoire de la concurrence entre *threads* est différemment "aléatoire" que le générateur psedo-aléatoire au sens où il est aussi influencé par ce qu'il tout se passe sur l'ordinateur en même temps.

5. Dans Exapunks, pour chaque niveau, il y a 100 versions, celles-ci ne sont pas écrites à la main, mais générées aléatoirement.

7 Annexe : Responsable du code des méta-fichiers

C'est le rôle le plus algorithmique. Vous allez devoir implémenter différents types de méta-fichiers correspondant à différentes approches algorithmiques du texte puis des données.

Dans EXAPUNKS, il n'y a qu'un type de fichiers : une liste d'entiers et de mots plus un pointeur, que l'on peut étendre vers la droite (si le pointeur est à la fin), consulter à l'endroit du pointeur, déplacer le pointeur (SEEK), mais aussi où l'on peut contracter la liste en enlevant la valeur pointé. Cette dernière contrainte complique les choses.

Dans OCTOPUNKS, nous allons proposer plusieurs structures de données. Chaque "fichier" aura alors un comportement différent en fonction de son type, la commande `MAKE` pourra aussi avoir des variantes pour créer des fichiers de type différents.

7.1 Objectif 10

Pour la version minimal, on demande 3 types de fichiers :

- Des fichiers "Pile", créés avec `MAKELIFO`, qui implémente le fichier avec une pile :
  - on pointe toujours sur la fin du fichier,
  - la commande `SEEK` ne fait rien dessus,
  - on lit la dernière case du fichier en l'effaçant (opération "pop"),
  - on écrit en empilant (opération "push"),
  - la commande `VOID F` efface la dernière case (opération "pop"),
  - la commande `TEST EOF` test alors si la pile est vide.
- Des fichiers "File", créés avec `MAKEFIFO`, qui implémente le fichier avec une pile.
- Des fichiers "Tableau Dynamique", créés avec `MAKE` (c'est le défaut de EXAPUNKS), qui consiste en une `ArrayList` qui se comporte comme les fichier originaux, attention à la commande `VOID F`.

7.2 Fonctionnalités supplémentaires

Pour avoir plus, voici les comportements à implémenter, dans l'ordre de votre choix. Certaines tâches sont à faire en interaction avec vos camarades, et sont prioritaires lorsque ceux-ci en sont à ces points, elles sont indiqués par une ou plusieurs lettres entre parenthèses ('G' pour graphique, 'M' pour métier, 'T' pour textuel, 'C' pour communication, 'F' pour fichier et 'E' pour enregistrement), le coordinateur n'est pas indiqué car il est toujours présent dès qu'il y a interaction entre des membres, si des membres non-présents sont évoqués (car votre groupe est petit), ces tâches ne sont pas demandées :

- Faire de nouveaux types de fichiers :
  - Des fichiers "Double Pile", créés avec `MAKEZIPER`, qui se comportent comme les fichiers originaux de EXAPUNKS, sauf que que l'on va utiliser deux piles : les caractères avant le pointeur et ceux après.

Cela permet une opération `VOID F` en temps constant, par contre, c'est `SEEK` qui devient en temps linéaire.
  - Des fichiers "Arbres", créés avec `MAKETREE`, qui se comportent comme les fichiers originaux, sauf qu'ils sont implémentés avec des arbres binaire, ce qui permet `VOID F` et `SEEK` en temps logarithmique (randomisé) :
    - chaque noeud contient une lettre et une taille,
    - la lecture de gauche à droite donne la liste du fichier (comme dans un ABR),
    - la racine contient la lettre pointée,
    - lire une lettre fait avancer à droite en rebalançant l'arbre,
    - `VOID F` consiste à aller chercher la lettre la plus ç gauche de l'arbre droit et la placer à la racine,
    - `SEEK n` consiste à trouver le sous-arbre à la bonne position, en faire la racine, et rebalancer le tout.
  - (F) Des fichiers "Tubes", créés avec `MAKEPIPE` ce qui crée deux fichiers, encodés avec des files, représentant un contenu partagé.
- (G) Afficher différemment les différents types de fichiers,
- (I) Permettre des conditions de victoire dépendant du contenu des fichiers à la fin,
- (T) Permettre de créer des niveau contenant un fichier au départ,
- (G,T) Permettre de la génération aléatoire maligne de fichiers en début de niveau,

8 Annexe : Responsables “gamification”

Pour les touche-à-tout qui veulent manipuler des fichiers, des conditions algorithmiques et même faire un peu de réseau.

Il s'agit ici d'implémenter la “gamification”, c'est à dire ce qu'il manque pour faire un vrais jeu : permettre des conditions de victoire divers et bien rédigé, permettre de lancer des niveau sur des instances aléatoires, mesurer les paramètres (taille, temps et activité), enregistrer les solutions, et comparer les solutions.

8.1 Objectif 10

Pour la version minimal, on vous demande :

— De faire un menu de choix de niveaux avec au moins 3 niveaux aux conditions de victoires différents,
— pour chacun, de lancer les test sur 5 variantes,
— d'enregistrer les solutions et les résultats localement,
— de comparer avec les résultats précédents et de charger une solution préalablement enregistrée.

8.2 Fonctionnalités supplémentaires

— Mettre en place un mini-serveur sur votre conteneur (voir cours d'administration système) qui interagira avec votre programme de jeu affin de comparer vos résultats en ligne,
— l'étendre pour permettre les batailles de robots (comme sur EXAPUNKS),
— permettre plus de conditions de victoire (fichiers, nombre de robots, etc...),
— permettre des tests plus expansifs (100 tests par niveaux),
— tout autre améliorations qui en feront un peu plus un “jeu”.