Projet informatique 2019-2020

Rattrapage

Attention ! Ce sujet n'a pas vocation à être imprimé. (D'ailleurs, il n'a probablement pas de vocation du tout puisqu'il y a peu de chance que ce sujet soit conscient.)

1 Jeux à programmer

1.1 Environment line

On souhaite coder un jeux de gestion de ligne de production, sous forme d'un idle game.

Résumé des épisodes précédents : À force de jouer avec le supercalculateur de l'école, les élèves du parcours CIDM ont ouvert une porte vers un univers parallèle, dévoilant ainsi l'existence d'une seconde école ENSIIE et de tout un tas d'autres écoles d'ingénieur. Avec effroi, les deux ENSIIE virent leur classement parmi les écoles d'ingénieur les plus investies dans le développement durable chuter, naturellement englouties sous un flot d'énergie démentiel pour maintenir la porte ouverte entre les mondes. Après une lutte acharné chacune des écoles est retournée dans son univers avec comme seul objectif de refermer la porte.

Ce jeu se joue à un seul joueur. L'objectif du joueur est de gérer et développer sa ligne de production. Pour cela, il dispose d'une carte contenant des ressources qu'il va pouvoir collecter et transporter jusqu'à la porte transuniverselle. En faisant cela, le joueur devra maîtriser son énergie et son environnement pour ne pas faire sombrer la planète dans le chaos le plus total.

1.2 La carte

La carte du jeu est un plateau de taille $n \times n$ cases. Au début du jeu, le joueur choisi parmi 3 types de cartes, *facile* (10×10), *moyenne* (20×20), *grande* (30×30). Chaque case peut contenir 4 types d'éléments :

- un élément de la ligne de production (collecteur, tapis roulant, déchetterie et centre de recyclage)
- une source de ressource
- rien, la case est vide
- la porte transuniverselle à refermer

1.3 L'énergie, l'environnement et les élèves

Tout au long du jeu, le joueur doit maîtriser deux quantités :

- Son énergie, qui lui permet de faire fonctionner et de modifier sa ligne de production. On notera dans la suite E la quantité d'énergie du joueur.
- Son environnement, qui mesure la santé générale de la planète. On notera dans la suite DD cette valeur. Plus elle est grande meilleure est l'environnement de la planète.

Au début du jeu, le joueur démarre avec `100 E` et `100 DD`.

Une manière d'augmenter ces deux valeurs est d'utiliser les élèves de l'ENSIIE. Le joueur recrute régulièrement de nouveaux élèves, des `FISE` ou des `FISA`. A chaque tour, chaque élève FISE produit `1 E` et `1 DD`. Les FISA étant alternant, ils ne produisent pas à chaque tour, mais, en contrepartie, ils sont plus flexibles. Tous les deux tours, chaque élève FISA peut produire soit `4 E`, soit `4 DD` selon le choix du joueur. Tous les FISA produisent la même ressource en même temps. Le joueur peut modifier ce choix à chaque tour à volonté.

Le joueur commence la partie avec `5 FISE` et `5 FISA`. Le joueur peut recruter d'autres élèves au cours de la partie. Un élève (`FISE` ou `FISA`) coûte `50 E` et `20 DD`.

1.4 Les ressources, la porte et les déchets

L'objectif du joueur est de fermer la porte transuniverselle. Pour cela, il doit amener des ressources depuis une source vers la porte. Au début du jeu, la source produit trois ressources tous les 10 tours en partant du tour 0. Cette ressource peut être transportée avec des tapis roulant (voir ci-dessous) jusqu'à la porte, ce qui ferme peu à peu cette dernière. Il est possible d'augmenter la fréquence de production des ressources. Si une ressource est produite mais n'est pas transportée dans le tour par un collecteur (voir plus bas) alors la ressource est perdue, elle n'est pas stockée sur la source.

Pour terminer le jeu, le joueur doit déplacer 10000 ressources jusqu'à la porte.

Chaque ressource amenée à la porte produit un déchet. La porte peut contenir autant de déchet que souhaite mais, à chaque tour, chaque déchet présent sur la carte diminue le nombre de `DD` du joueur de `1 DD`. Ces déchets peuvent être traités (voir plus bas).

1.5 Les machines

Le joueur a besoin de machines pour collecter et transporter les ressources jusqu'à la porte ou pour déplacer et traiter les déchets. Il existe plusieurs types de machine. Chaque machine a une taille d'une case. Les machines ont des fonctions qui s'activent à chaque tour [1].

Chaque machine a une fonction (ce qu'elle fait quand elle s'active), une orientation (gauche, droite, haut ou bas), des sorties et des entrées (possiblement 0) et un coût. Si une ressource est envoyée sur une machine mais par une case autre qu'une de ses entrées, la ressource disparait. De même, si la ressource est envoyée sur une case vide, elle disparaît. De même pour les déchets, mais au lieu de disparaître, ils sont renvoyés à la porte.

1.5.1 Collecteur de ressource ou de déchet

Un collecteur est une machine qui doit être placée sur une case adjacente à une source. Au début du jeu, à chaque tour, le collecteur envoie une ressource produites par la source sur la case voisine indiquée par son orientation. Si la source n'a rien produit, alors le collecteur n'envoie rien. Le collecteur n'a pas d'entrée. On ne peut donc lui envoyer de ressource ou de déchet. Au début du jeu, puisqu'une ressource est produite tous les 10 tours, le collecteur envoie une ressource tous les 10 tours. Il est possible d'améliorer le nombre de ressources envoyées par un collecteur.

Un collecteur agit de manière similaire si on le place à côté de la porte. A chaque tour, il envoie alors un déchet produit par la porte sur la case voisine indiquée par son orientation.

Un collecteur qui est placé sur une case non adjacente aux sources ou à la porte ne fait rien.

Les collecteurs ne s'activent pas en même temps. On active les collecteurs ligne par ligne, de haut en bas. Sur une même ligne, on les actives de gauche à droite. Si deux sources ou plus sont adjacentes au collecteur, le collecteur sélectionne au hasard la ou les ressources qu'il transmet.

1.5.2 Tapis roulant

Un tapis roulant permet de transporter les ressources/déchets en ligne droite. Elles sortent sur la case voisine indiquée par son orientation. Elles entrent par n'importe quelle autre case. Un tapis a donc une sortie et trois entrées. La ressource ou le déchet sort 1 tour après être entrée sur le tapis roulant. Par exemple, si on a, dans cet ordre, de gauche à droite, un collecteur construit au tour 15 orienté vers la droite, un tapis roulant orienté vers la droite et la porte, alors au tour 20, une ressource produite par la source est envoyée du collecteur au tapis, et au tour 21 elle est transportée du tapis à la porte, produisant ainsi un déchet.

Un tapis peut transporter autant de ressources/déchets qu'on veut. Ainsi si un tapis est orienté vers le bas et reçoit une ressource par l'entrée gauche et un déchet depuis le haut au même tour, alors au tour suivant, la ressource et le déchet sont envoyés en même temps vers la case du bas.

Attention, tous les tapis s'activent en même temps. Il n'y a pas d'ordre d'activation. Ainsi, supposons par exemple qu'on a dans cet ordre, un tapis orienté vers la droite avec une ressource et, à sa droite, un tapis orienté vers le bas, avec une ressource. Alors pendant l'activation, le tapis de gauche envoie sa ressource à droite, et, en même temps, le tapis de droite envoie sa ressource en bas. Ainsi, à la fin de l'activation, il y a une ressource sur le tapis de droite et une ressource en dessous de ce tapis.

1. C'est le joueur qui passe les tours quand il en a envie (voir la section sur le joueur)

1.5.3 Croix

Une croix est un ensemble de deux tapis roulant juxtaposés, permettant de croiser deux lignes de tapis roulants. La croix a 2 entrées et 2 sorties qui dépendent de l'orientation de la croix :

- Orientation haut : une entrée à droite envoyant les ressources vers la sortie à gauche ; une entrée en bas envoyant les ressources vers la sortie en haut.
- Orientation droite : une entrée à gauche envoyant les ressources vers la sortie à droite ; une entrée en bas envoyant les ressources vers la sortie en haut.
- Orientation bas : une entrée à gauche envoyant les ressources vers la sortie à droite ; une entrée en haut envoyant les ressources vers la sortie en bas.
- Orientation gauche : une entrée à droite envoyant les ressources vers la sortie à gauche ; une entrée en haut envoyant les ressources vers la sortie en bas.

Le fonctionnement est similaire à celui du tapis roulant.

1.5.4 Déchetterie

La déchetterie permet de stocker des déchets. La déchetterie possède 4 entrées et 0 sortie. Au début du jeu, une déchetterie peut stocker jusqu'à 50 déchets. Les déchets stockés n'influencent plus le nombre de `DD` du joueur. Si une ressource est envoyée à la déchetterie, elle disparaît. Si un déchet est envoyé à une déchetterie pleine, il est renvoyé à la porte. Il est possible d'augmenter la capacité des déchetteries.

1.5.5 Centre de recyclage

La centre de recyclage permet de transformer les déchets en ressource. Le centre possède comme le tapis roulant 3 entrées et une sortie. Le centre peut stocker jusqu'à 100 déchets. Il est possible d'augmenter cette capacité. Contrairement à la déchetterie, les déchets stockés continuent de diminuer le nombre de `DD` du joueur. Au début du jeu, à chaque tour, pour chaque paquet de 10 déchets qu'il possède, le centre de recyclage produit une ressource qu'il envoie vers sa sortie.

1.5.6 Détruire une machine

Il est possible de détruire une machine. Si des ressources sont présentes sur la machine, elles disparaissent. Si des déchets sont présents sur la machines, ils sont renvoyés à la porte. Ce peut être pratique pour améliorer sa ligne de production ou changer l'orientation d'une machine.

1.5.7 Coût et amélioration des machines

Chaque machine coûte des `E` et `DD` pour être construite, utilisée et détruite. Chaque machine peut aussi être améliorée à l'exception des tapis et des croix. Le détail des coût est indiqué ci-dessous.

Nom	Construction		Amélioration		Destruction		Effet des améliorations
	E	DD	E	DD	E	DD	
Collecteur	200	20	500	100	60	200	Le collecteur peut collecter une ressource/un déchet de plus en un tour.
Tapis	60	20	-	-	60	200	-
Croix	160	20	-	-	60	200	-
Centre de recyclage	500	40	1500	100	100	500	Le stockage du recyclage augmente de 10.
Déchetterie	100	100	200	600	100	200	Le stockage de la déchetterie augmente de 20.

1.6 Utiliser le personnel de l'école

Le joueur peut utiliser le personnel de l'école pour acquérir des bonus. Chaque personnel coûte mais rapporte des avantages. Un personnel peut être acheté plusieurs fois pour cumuler plusieurs fois les avantages. Le détail des coûts et des bonus est indiqué ci-dessous.

N°	Nom	Coût E	Coût DD	Effet
1	Fetia Bannour	100	30	Le coût de construction des collecteurs diminue de 10E et 1DD (minimum 10E et 1DD).
2	Kevin Goilard	100	30	Le coût de construction des tapis diminue de 3E et 1DD (minimum 3E et 1DD).
3	Vincent Jeannas	100	30	Le coût de construction des croix diminue de 8E et 1DD (minimum 8E et 1DD).
4	Thomas Laurent	100	30	Le coût de construction des centres de recyclage diminue de 25E et 2DD (minimum 25E et 2DD).
5	Massinissa Merabet	100	30	Le coût de construction des déchetterie diminue de 5E et 5DD (minimum 5 et 5DD).
6	Stefi Nouleho	200	100	Le coût d'amélioration des collecteurs diminue de 25E et 5DD (minimum 25E et 5DD).
7	Vitera Y	200	100	Le coût d'amélioration des centres de recyclage diminue de 75E et 5DD (minimum 75E et 5DD).
8	Laurence Bourard	200	100	Le coût d'amélioration des déchetterie diminue de 10E et 30DD (minimum 20E et 30DD).
9	Nicolas Brunel	100	200	Le coût de destruction des collecteurs diminue de 3E et 10DD (minimum 3E et 10DD).
10	Anastase Charantonis	100	200	Le coût de destruction des tapis diminue de 3E et 10DD (minimum 3E et 10DD).
11	Catherine Dubois	100	200	Le coût de destruction des croix diminue de 3E et 10DD (minimum 3E et 10DD).
12	Stefania Dumbrava	100	200	Le coût de destruction des centres de recyclage diminue de 5E et 25DD (minimum 5E et 25DD)
13	Alain Faye	100	200	Le coût de destruction des déchetterie diminue de 5E et 10DD (minimum 5E et 10DD)
14	Anne-Laure Ligozat	1000	10	La moitié des déchets de chaque case disparaissent.
15	Christophe Mouilleron	1000	400	L'école recrute 20 FISE et 10 FISA.
16	Marie Szafranski	1000	400	A chaque fois qu'une ressource est envoyée à la porte, elle compte double. Le nombre de déchet produit est toujours de 1.
17	Gael Thomas	1000	400	Quand un déchet quitte la porte, il a une chance sur 10 de disparaître.
18	Eric Lejeune	1000	200	Le coût des FISE diminue de 5E et 2DD (minimum 5E et 2DD).
19	Christine Mathias	1000	200	Le coût des FISA diminue de 5E et 2DD (minimum 5E et 2DD).
20	Katrin Salhab	1500	300	Les membres du personnel coûtent 50E et 20DD de moins (minimum 10E et 10DD).
21	Julien Forest	2000	500	Les FISE produisent 1E et 1DD de plus chaque tours.
22	Thomas Lim	1000	400	Les sources mettent un tour de moins à produire (minimum 1).
23	Dimitri Watel	2000	500	Les FISA produisent 4E ou 4DD de plus tous les deux tours.
24	Laurent Prével	3000	1000	Un dixième des FISE et des FISA de l'école sont diplômés. Ils quittent l'école mais chaque élève diplômé envoie une ressource à la porte.

FAQ : Plusieurs Gael Thomas : À chaque fois qu'un déchet quitte la porte, pour chaque Gael Thomas achetés, on lance un dé à 10 faces (entre 0 et 9) et s'il tombe au moins une fois sur 0, le déchet disparaît.

Disclaimer : Ce jeu se veut avant tout ludique et bon enfant. S'il est vrai (voire évident) que certains effets ont été associés à des membres du personnel à cause de leurs fonctions, une bonne partie a été positionnée arbitrairement. Le tableau ci-dessus ne veut, en aucun cas, signifier qu'une personne affecte plus positivement ou plus négativement la production ou le développement durable à l'ENSIIE et les étudiants de l'école.

1.7 Début du jeu

Au début du jeu, une carte est générée selon la taille choisie par la joueur. Sont alors placés aléatoirement sur la carte 2 cases de type source et 1 case contenant la porte transuniverselle.

1.8 Tour de jeu

A chaque tour, le joueur peut, dans n'importe quel ordre :

— recruter un ou plusieurs élèves

— construire, améliorer ou détruire une machine

— utiliser un personnel de l'école

— terminer le tour : dans cet ordre, les tapis et croix s'activent, les sources produisent en fonction du tour, la porte produit des déchets si des ressources ont été envoyées, les centres de recyclages s'activent si possible et les collecteurs s'activent,

— quitter le jeu

1.9 Cahier des charges

Votre objectif est de coder le jeu Environment Line. Votre jeu devra permettre, à minima :

— De démarrer une partie et de la jouer jusqu'à son terme ;

— D'afficher l'état du jeu à chaque tour.

2 Travail à réaliser.

Dans ce projet, il vous est demandé de respecter les consignes de chaque lot et d'utiliser les outils qui vous ont été présentés en cours. Le travail est découpé en 3 lots, le Lot A, le Lot B et le Lot C. Les deux premiers lots sont découpés en 12 tâches (4 pour le Lot A et 8 pour le Lot B). Vous êtes libres de découper le Lot C comme bon vous semble.

Chaque étudiant doit effectuer au moins **3 tâches**. Idéalement, un groupe de 4 élèves doit donc effectuer au moins les lots A et B, chacun 1 tâche du lot A et chacun 2 tâches du lot B. Une tâche vous sera attribuée et validée à 2 conditions : vous êtes désigné sur votre fichier GanttProject comme ressource de la tâche ; et vous avez commité la totalité du code relatif à cette tâche ou presque (une autre personne peut vous aider localement, corriger un bug ou une faute d'orthographe, ...). **Afin de faciliter la correction, chaque message de commit doit préciser le nom de la tâche associée au commit. Sans quoi le chargé de projet n'est pas tenu de faire un effort.** Un élève qui n'aurait pas effectué 3 tâches sans excuse valable se verra attribué la note de 0. Aucune tâche n'est sensée être trop longue. N'hésitez pas à demander de l'aide si vous en avez besoin, il vaut mieux demander trop que pas assez.

Attention : on vous demande d'effectuer les tâches, pas d'être parfait, vous avez le droit à l'erreur.

Aucun rapport ni aucune soutenance ne sont exigés pour ce projet avant le lot C. Toutefois, des points réguliers (à chaque séance) sont à faire avec votre chargé de projet. Vous devez être capable de dire où vous en êtes, d'indiquer les problèmes que vous rencontrez, ... La présence en séance est obligatoire (même en distanciel). Une petite présentation de votre projet (sous forme de démonstration, de rapport ou de soutenance) peut être un plus et permet de donner un peu plus de crédit à votre travail, un sentiment d'accomplissement mais cela ne doit pas se faire au détriment du Lot A et du Lot B. Le Lot C étant complètement libre, un rapport ou une soutenance est important pour permettre au chargé de projet de comprendre ce que vous avez fait. Les groupes parvenant à effectuer un lot C crédible seront récompensés d'une meilleure note que ceux n'ayant pu effectuer que le lot A et le lot B.

Les outils Doxygen et CUnit sont facultatifs. Ils sont moins prioritaires que les autres. Toutefois :

— votre code doit être commenté !

— votre code doit compiler avec `gcc -Wall -Wextra -std=c99` sans erreur ni warning !

— votre code doit être testé !

Un code non commenté, un code qui ne compile pas, ou un code dont l'exécution plante au démarrage se verra attribué la note de 0.

2.1 Travail en équipe

Remarque importante : vous êtes 4 dans votre groupe. Répartissez vous les tâches selon vos compétences respectives. Cependant, il est important que chacun d'entre vous connaisse l'avancement du projet.

2.2 Lot A

Le premier lot consiste à réfléchir à la mise en place du projet et à préparer les interfaces de vos modules (fichiers `.h`). Les interfaces ne contiennent que la définition de types (concrets ou abstraits), les signatures des fonctions utiles et des commentaires indiquant leur fonction (et non leur implantation interne). Elle ne contiennent pas de code à proprement parler. Pour les produire, il vous faut donc réfléchir, non pas à comment vous aller les implanter, mais ce que vous voulez que les fonctions fassent. À l'aide de ces signatures, vous pouvez programmer votre fichier principal `main.c` en le précompiler en fichier `main.o`. Ce fichier `main.c` doit contenir le code du logiciel permettant à un joueur de lancer une partie et de la jouer jusqu'à son terme selon le cahier des charges de la section précédente.

Vous ne serez pas en mesure de générer un exécutable à partir de `main.c`, puisque les fonctions décrites dans les interfaces ne sont pas encore implantées. Mais ça ne vous empêche pas de les utiliser dans le fichier `main.c` puisque vous savez ce que ces fonctions sont sensées prendre en entrée et produire en sortie. Votre fichier devra compiler avec la commande `gcc -Wall -Wextra -std=c99 -c main.c`.

Si vos interfaces et votre fichier `main.c` sont corrects, vous n'aurez plus qu'à implanter les fonctions des interfaces pour avoir un programme fonctionnel. Ces tâches sont prévues pour le lot B et le lot C. Le lot B constituera une implantation des fonctions de sorte à jouer en console. Le lot C constituera une évolution du projet (interface graphique, augmentation du nombre de fonctionnalités, ...).

Attribuez chacune des tâches suivantes, à une **(et une seule)** personne du groupe. N'hésitez pas à découper chaque tâche en sous-tâches.

2.2.1 Tâche A.1 – Mise en place du projet et `main.c`

Votre rôle est central pour le Lot A. Lisez les tâches de tous les membres du groupe avant de commencer.

Mettez en place le dépôt git et invitez votre encadrant sur votre dépôt en tant que `rapporter` ainsi que tous les membres du groupe en tant que `developper`. Ce git contiendra votre code. Créez une première branche `lot_a`. Le dernier commit de cette branche contiendra le code du Lot A une fois celui-ci terminé.

Créer un projet avec GanttProject et poussez le sur votre dépot git, dans la branche `lot_a`. Remplissez ce fichier avec les différentes tâches du Lot A. Mettez des durées arbitraires sur vos tâches. Ajoutez chaque membre du groupe comme ressource. Affectez les ressources aux tâches après décision de qui effectue quelle tâche. (Rappel : c'est nécessaire pour qu'une tâche soit attribuée à un membre du groupe). Enfin, indiquez les contraintes de précédences sur les tâches (quelle tâche doit être finie pour pouvoir commencer une nouvelle tâche). Vous pouvez couper les tâches en sous-taches sur vous voulez.

Codez le fichier `main.c`. Ce fichier devra utiliser **toutes les fonctions** et **uniquement les fonctions** qui auront été ajoutées dans les interfaces remplies par les autres membres du groupe. Remarque : ce fichier restera normalement inchangé tout le long du Lot B. Une fois ce dernier terminé, `main.c` pourra être compilé en un exécutable qui fera tourner le projet.

Vérifiez, une fois toutes les interfaces rédigées, que votre fichier `main.c` compile sous forme de fichier objet `main.o`.

Faites valider le Lot A par votre encadrant une fois toutes les tâches terminées.

Vous devez maintenir le dépot git à jour au fur et à mesure de l'avancement de votre travail. Vos commits doivent avoir l'indication *Tâche A.1* (en plus d'un message décrivant le commit).

2.2.2 Tâche A.2 – `carte.h`

L'interface `carte.h` est en charge de la carte de jeu. Elle doit permettre d'accéder à l'ensemble des machines, sources, ressources, déchets, personnels et porte présents sur la carte. Elle doit également permettre de modifier cette carte en fonction des décisions prises par le joueur. C'est aussi cette interface qui gère les différents tours de jeu et les `E` et `DD` disponibles du joueur.

Ajoutez chacun des types et des fonctions suivants, avec les paramètres qui vous semblent adéquats à vous et au responsable de la tâche A.1 pour coder le fichier `main.c`. Documentez ensuite ces fonctions et ces types.

- un type `carte` (possiblement abstrait)
- une fonction pour créer une carte
- une fonction pour libérer la mémoire allouée à une carte
- une fonction pour recruter un FISE
- une fonction pour recruter un FISA
- une fonction pour changer le type de ressource créée par les FISA
- une fonction pour terminer le tour.

D'autres fonctions sont prévues dans `carte.h` dans la tâche A.3.

Attention : aucune de ces fonctions n'échange d'informations avec le joueur humain (pas de `printf`, pas de `scanf`). C'est le rôle de `interface.h`.

On rappelle qu'un fichier `.h` ne contient pas de code, juste des signatures de fonctions.

Vous devez maintenir le dépot git à jour au fur et à mesure de l'avancement de votre travail. Vos commits doivent avoir l'indication *Tâche A.2* (en plus d'un message décrivant le commit).

2.2.3 Tâche A.3 – `machine.h`, `personnel.h` et fin `carte.h`

L'interface `machine.h` est en charge de gérer les informations des machines du jeu, indépendamment de leur existence dans le jeu. De même l'interface `personnel.h` est en charge de gérer les informations du personnel du jeu, indépendamment de leur existence dans le jeu. C'est dans ces interfaces qu'on retrouvera par exemple les informations de coût des machines ou du personnel.

Ajoutez à `machine.h` un type `machine` (possiblement abstrait). Ajoutez à `personnel.h` un type `personnel` (possiblement abstrait). Ajoutez à `carte.h` les fonction suivantes. Documentez ensuite ces fonctions et ces types.

— une fonction pour ajouter une nouvelle machine, si le budget le permet

— une fonction pour améliorer un machine, si le budget le permet

— une fonction pour détruire un machine, si le budget le permet

— une fonction pour acheter un membre du personnel, si le budget le permet

Attention : aucune de ces fonctions n'échange d'informations avec le joueur humain (pas de `printf`, pas de `scanf`). C'est le rôle de `interface.h`.

On rappelle qu'un fichier `.h` ne contient pas de code, juste des signatures de fonctions.

Vous devez maintenir le dépot git à jour au fur et à mesure de l'avancement de votre travail. Vos commits doivent avoir l'indication *Tâche A.3* (en plus d'un message décrivant le commit).

2.2.4 Tâche A.4 – `interface.h`

L'interface `interface.h` est en charge de tous les échanges entre le joueur humain et le jeu : affichage et prise de décision du joueur.

— une fonction pour demander la taille d'une nouvelle carte en début de jeu

— une fonction pour afficher la carte

— une fonction pour demander une action au joueur (quitter le jeu, passer au tour suivant, ajouter une machine, ...)

— une fonction pour demander au joueur des informations nécessaires pour ajouter une nouvelle machine à la carte

— une fonction pour demander au joueur des informations nécessaires pour sélectionner une machine de la carte (pour l'améliorer ou la détruire)

— une fonction pour demander au joueur des informations nécessaires pour sélectionner un membre du personnel à acheter

`Aucune de ces fonctions ne modifie la carte de jeu. C'est le rôle de carte.h.`

Vous devez maintenir le dépot git à jour au fur et à mesure de l'avancement de votre travail. Vos commits doivent avoir l'indication *Tâche A.4* (en plus d'un message décrivant le commit).

2.3 Lot B

Le but du Lot B est de permettre de compiler pleinement le fichier `main.c` en un exécutable fonctionnel. Il faudra ici implanter toutes les fonctions des interfaces créées dans le Lot A.

Vous ne devez plus déplacer la branche `lot_a`. Elle restera définitivement pointée sur la fin de votre travail précédent même si vous remettez en question ce travail ultérieurement.

Attribuez chacune des tâches suivantes, à une personne **(et une seule)** du groupe. N'hésitez pas à découper chaque tâche en sous-taches. Notez que certaines tâches sont plus faciles et courtes que d'autres.

2.3.1 Tâche B.1 – Lier tous les fichiers

Dans le dépot git, créez une deuxième branche `lot_b`. Le dernier commit de cette branche contiendra le code du Lot B une fois celui-ci terminé. Vous pouvez, vous ou les autres membres du groupe, si vous le souhaitez, créer d'autres branches temporaires le temps de la conception du Lot B.

Créer un fichier `makefile` et quatre dossiers `src`, `obj`, `bin` et `headers` pour ranger vos fichiers. Remplissez le makefile pour qu'il permette, à terme, de compiler chacun des fichiers sources en fichier objet (`.o`) et le fichier `main.o` en un exécutable.

Remplissez le fichier GanttProject avec les différentes tâches du Lot B. Mettez des durées arbitraires sur vos tâches. Affectez les ressources aux tâches après décision de qui effectue quelle tâche.

Vous devez maintenir le dépot git à jour au fur et à mesure de l'avancement de votre travail. Vos commits doivent avoir l'indication *Tâche B.1* (en plus d'un message décrivant le commit).

2.3.2 Tâche B.2 – Tester votre code

Créez sur le dépot git une branche `lot_b_test` dans laquelle vous effectuerez vos tests sans affecter le code du `lot_b`. Créez un fichier `test.c` qui vous permettra de tester. N'hésitez par à modifier le code des autres et à commiter, il n'y a pas de risque, vous êtes sur une autre branche. Vous pouvez utiliser `CUnit` si vous le souhaitez.

Effectuez les tests de l'exécutable. Remontez les bugs aux membres du groupe pour déterminer qui doit corriger ce bug. Vous pouvez par exemple commiter sur `lot_b_test` un code qui démontre le bug. Pensez à garder une trace écrite de vos tests, en particulier si le bug est visuel, s'il n'affecte pas le fonctionnement du jeu, s'il n'est pas codable, vous pouvez simplement décrire le bug dans un fichier texte que vous commiterez.

Corrigez ce bug sur la branche `lot_b`. Puis fusionnez les deux branches et recommencez.

Vos tests doivent inclure toutes les règles du jeu. Par exemple :

- démarrage sans problème du jeu
- vérification des fuites mémoires avec valgrind
- placement original des sources et de la porte
- recrutement de FISE ou de FISA
- changement de type de ressource créée par les FISA
- achat d'une machine
- amélioration d'une machine
- destruction d'une machine
- collecte de ressources ou de déchets
- déplacement de ressources ou de déchets avec un tapis ou une croix
- disparition des ressources/déchets (quand ils tombent par terre ou son envoyés sur une sortie d'une machine)
- diminution des `DD` avec les déchets
- stockage en déchetterie
- recyclage des déchets
- achat du personnel
- achat du même personnel plusieurs fois
- tout autre idée de votre part ...

Vous devez maintenir le dépot git à jour au fur et à mesure de l'avancement de votre travail. Vos commits doivent avoir l'indication *Tâche B.2* (en plus d'un message décrivant le commit).

2.3.3 Tâche B.3 – Structures de données – `structures.h`

Attention, on parle bien ici de structures de donnée (tableau, file, pile, ensemble, dictionnaire, ...), pas de structure au sens `struct` en C.

Créez un nouveau fichier `structure.h` qui gérera toutes vos structures de données. Ces structures gèreront le contenu de la carte, la liste des machines ou des personnels achetés, ... Ces structures seront utilisées pour implanter les types `carte`, `machine` et `personnel`.

Il peut s'agit de simples tableaux ou de structures plus évoluées. Ajoutez des types et des fonctions permettant de gérer ces structures (ajout, suppression, lecture du `i-eme` élément, ...). Documentez ensuite ces fonctions et implantez les dans un fichier `structure.c`.

Vous devez maintenir le dépot git à jour au fur et à mesure de l'avancement de votre travail. Vos commits doivent avoir l'indication *Tâche B.3* (en plus d'un message décrivant le commit).

2.3.4 Tâche B.4 – Constantes et variables globales

Les règles du jeu disposent de beaucoup de constantes. Placez dans les fichier `carte.h`, `machine.h` et `personnel.h` des constantes ou des variables globales contenant les différents paramètres du jeu. Vous pouvez aussi créer un fichier annexe contenant ces informations. Par exemple le nombre de FISE au début du jeu, le nom des différents membres du personnel, de combien diminue le coût de construction des machines si Kevin Goilard est acheté, ... Ces constantes vous donneront plus facilement accès à ces informations.

Vous devez maintenir le dépot git à jour au fur et à mesure de l'avancement de votre travail. Vos commits doivent avoir l'indication *Tâche B.4* (en plus d'un message décrivant le commit).

2.3.5 Tâche B.5 – Getters de `machine.h`, `machine.c`, `personnel.h` et `personnel.c`

Implantez le type `machine` dans le fichier `machine.c` et le type `personnel` dans le fichier `personnel.c`.

Ajoutez à `machine.h` des fonctions utiles pour récupérer des informations sur les machines : par exemple récupérer les coûts ou l'orientation de la machine. Implantez ces fonctions dans `machine.c`. Ajoutez à `personnel.h` des fonctions utiles pour récupérer des informations sur le personnel : par exemple récupérer les coûts de la personne. Implantez ces fonctions dans `personnel.c`.

Vous devez maintenir le dépot git à jour au fur et à mesure de l'avancement de votre travail. Vos commits doivent avoir l'indication *Tâche B.5* (en plus d'un message décrivant le commit).

2.3.6 Tâche B.6 – Getters de `carte.h` et `carte.c`

Implantez le type `carte` dans le fichier `carte.c`.

Ajoutez à `carte.h` des fonctions utiles pour récupérer des informations sur la carte : récupérer les ressources et déchets de chaque case, le nombre de FISE ou FISA recrutés, la quantité d'`E` ou de `DD`, ... En d'autres termes, toutes les fonctions qui vous semblent adéquates à vous et à votre référent pour coder le fichier `carte.c` et le fichier `interface.c` des tâches `B.6` et `B.7`. Documentez ensuite ces fonctions et implantez les dans `carte.c`.

Vous devez maintenir le dépot git à jour au fur et à mesure de l'avancement de votre travail. Vos commits doivent avoir l'indication *Tâche B.6* (en plus d'un message décrivant le commit).

2.3.7 Tâche B.7 – `carte.c`

Implantez toutes les fonctions restantes du fichier `carte.h` dans le fichier `carte.c`.

Vous devez maintenir le dépot git à jour au fur et à mesure de l'avancement de votre travail. Vos commits doivent avoir l'indication *Tâche B.7* (en plus d'un message décrivant le commit).

2.3.8 Tâche B.8 – `interface.c`

Implantez toutes les fonctions du fichier `interface.h` dans le fichier `interface.c`.

Conseil : contentez vous dans un premier temps d'un affichage simple. Quand il fonctionnera, vous pourrez le faire évoluer vers une esthétique de meilleure qualité.

Vous devez maintenir le dépot git à jour au fur et à mesure de l'avancement de votre travail. Vos commits doivent avoir l'indication *Tâche B.8* (en plus d'un message décrivant le commit).

2.4 Lot C

Le Lot C est complètement libre. Rédigez un petit rapport ou présentez oralement votre projet au chargé de projet, selon ce que vous et lui préférez.

Voici quelques idées :

— Vous pouvez tenter de mesurer l'impact environnemental de votre code. Par exemple en observant sa consommation mémoire avec `top` ou `valgrind`, le temps CPU utilisé avec `time`, ou les lectures écritures sur le disque avec l'outil `iotop`. Vous pouvez aller à fond et regarder la consommation électrique de la machine pendant l'exécution du jeu, mais il vous faut un wattmètre.

— Vous pouvez utiliser une bibliothèque d'interface graphique pour passer d'un jeu en console à un jeu graphique.

— Vous pouvez changer des règles. Par exemple créez des machines qui fabriquent des objets à partir des ressources, créez d'autres améliorations que le joueur peut acheter en cours de route, rendre le jeu multijoueur ... Les changements doivent être drastiques. Une modification mineure ne sera pas une contribution très intéressante.

Organisez votre Lot en tâches, répartissez les tâches comme dans les autres lots.

Vous devez maintenir le dépot git à jour au fur et à mesure de l'avancement de votre travail.

2.5 Séquencement du travail

Voici, à titre indicatif, les dates importantes du projet. Elles sont modulables selon le bon vouloir de votre chargé de projet. Chacun de ces rendus est séparé du suivant par deux semaines pour vous laisser le temps de corriger le Lot A et le Lot B si votre encadrant estime qu'il n'est pas correct ou qu'il est insuffisant pour pouvoir continuer avec le lot suivant. En particulier, puisque le Lot B nécessite la validation du A et que le C nécessite celle du B, les dates de rendus peuvent être amenées à changer.

Vous pouvez bien entendu rendre votre Lot A avant le 22 et le B avant le 19 si vous estimez avoir fini.