troisiéme sous dossier qui s'appelle monProjetAlaFac Qui contient tous votre projet de la fac. (seulement le readme si vous avez juste mis un readme) Il vous reste seulement a dire a eclipse que vous souhaitez mettre des sources la dedans plutot que dans src, et c'est assez simple dans eclipse, dans monProjetSurMonOrdi vous faites F5 ou refresh pour voir apparaitre ce troisiéme sous dossier "monProjetAlaFac" right-clicquez sur le dossier selectionner "build path" puis dans le sous menu "add to source folder" et apres cela, vous voyez apparaitre un deuxieme dossier monProjetAlaFac juste a coté de src qui a le meme icone que src pour vous faire comprendre que c'est aussi considéré comme un dossier de source.

Lorsque vous voulez creer une nouvelle classe java, vous prenez soin de la creer dans monProjetAlafac.

Quand vous vouler commiter, vous le fait directement depuis une fenetre commande sans passer par eclipse, vous aller dans le bon dossier, vous faites git add monsupertruc.java ou bien git add monDossierQuiContientMonSupertruc.java git commit (faut mettre un message sur c'est quoi ce commit) git push (y a trois trucs a faire dans le bon ordre, pour dire 1- ce qu'on veut sauver, 2- sauver en local, 3- enfin sauver a la fac. Vous aurez peut etre des petits pb de noms de package a mettre au point pour vos java déja existant. Il faut que le nom de package correspondent au dossier ou le java se trouve.

10 Le projet "l'Île Interdite"

Ce projet de programmation vous invite à construire une version informatique et un peu simplifiée du jeu *L'île interdite.* L'Île renferme des trésors archéologiques inestimables, que vous êtes venus récupérer. Mais l'Île sombre, ses différentes parties disparaissent peu à peu, et l'ensemble sera bientôt englouti par les flots. Dans ce jeu de Matt Leacock, les joueurs doivent coopérer pour ralentir la progression de l'eau suffisamment longtemps pour récupérer quatre artefacts et s'échapper en hélicoptère avant l'inéluctable et imminente disparition des derniers vestiges de l'Île.

Aperçu des règles du jeu

Les joueurs doivent récupérer 4 artefacts, chacun lié à un élément (air, eau, terre, feu). L'île est formée par un ensemble de zones disposées en grille. Outre les zones ordinaires on a les cas particuliers suivants :

— un héliport, d'où les joueurs doivent s'envoler à l'issue de la partie,
— des zones associées à chacun des éléments, qui permettent de récupérer l'artefact correspondant.

Chaque zone est dans l'une des situations suivantes : « normale », « inondée » ou « submergée ». Une zone « inondée » peut être asséchée et revenir à la situation « normale ». Une zone « submergée » a définitivement disparu. Un joueur peut avoir en main des clés, chacune associée à un élément, et qui permettent de récupérer l'artefact correspondant. Le tour d'un joueur se déroule comme suit :

1. Effectuer jusqu'à trois actions à choisir parmi :
   — se déplacer vers une zone adjacente non submergée (haut, bas, gauche, droite),
   — assécher la zone sur laquelle il se trouve ou une zone adjacente si la zone visée est inondée mais pas submergée, ou
   — récupérer un artefact s'il se trouve sur une zone d'artefact et possède une clé correspondante.
2. Chercher une clé, ce qui peut soit ne rien donner, soit donner une clé associée à un élément, soit déclencher un événement spécial « montée des eaux », qui en particulier inonde la zone où se trouve le joueur (voir ci-dessous).
3. Inonder trois zones tirées au hasard, avec deux cas à considérer :
   — Si la zone correspondante est dans la situation « normale », elle devient « inondée ». Cette zone est alors en sursis, mais elle peut toujours être traversée ou utilisée normalement par les joueurs.
   — Si la zone correspondante est déjà dans l'état « inondée » ou « submergée », elle devient « submergée ». La zone ne peut plus être traversée ni utilisée par les joueurs, et cet état est définitif. Un joueur présent sur la zone lors de sa submersion peut s'échapper vers une zone adjacente, s'il en existe une non submergée. Dans le cas contraire le joueur se noie.

La partie est gagnée lorsque les 4 artefacts sont récupérés et que tous les joueurs sont regroupés à l'héliport. La partie est perdue dès qu'il n'est plus possible de la gagner.

Organisation

L'application sera organisée selon une architecture Modèle-Vue-Contrôleur (MVC), et vous pouvez vous inspirer du fichier `Conway.java` disponible sur la page du cours. Les différentes sections du sujet vous proposent de mettre progressivement en place les différents éléments du jeu, et suggèrent un ordre dans lequel les traiter. Précisons toutefois qu'il est toujours utile de réfléchir en amont à la manière dont les éléments suivants pourront s'insérer dans votre code.

10.1 On va se la couler douce

Dans un premier temps nous considérons uniquement l'île, sans aucun joueur et sans nécessité de tenir compte des zones spéciales.

1. Créer ce modèle réduit dans lequel seules l'île et les zones sont présentes.
   Nous vous encourageons à inclure des méthodes `toString()` permettant de représenter les différents objets, des affichages rendant compte des différents événements, et des tests unitaires quand cela est pertinent.
2. Créer une vue affichant les zones de l'île et leur état.
3. Ajouter un bouton « fin de tour ». Un clic sur ce bouton doit procéder à l'inondation de trois zones aléatoires.

∞. Faire en sorte que les trois zones inondées à la fin du tour soient choisies parmi les zones non encore submergées.

10.2 Noooon, pas la trempette !

Nous allons maintenant inclure des joueurs, qui au terme de cette partie pourront se déplacer et assécher des zones.

1. Étendre le modèle avec un joueur et les méthodes permettant de faire se déplacer le joueur de sa zone courante à une zone adjacente. Faire que la vue représente le joueur sur sa zone.
2. Étendre la vue et le contrôleur avec des moyens d'ordonner le mouvement du joueur.
   Au choix : clics de boutons, frappes au clavier, clics sur la zone cible, autre.
3. Étendre le contrôleur pour que le joueur puisse réaliser au maximum trois actions avant de cliquer sur le bouton « fin de tour ».
4. Étendre l'ensemble pour permettre plusieurs joueurs. Le contrôleur doit faire agir les joueurs à tour de rôle, en insérant une phase d'inondation (« fin de tour ») à la fin du tour de chaque joueur.
5. Étendre l'ensemble pour ajouter aux coups du joueur l'action d'assécher la zone sur laquelle il se trouve ou une zone adjacente, en choisissant une interface utilisateur cohérente avec celle permettant le déplacement.

∞. Faire que la vue indique le joueur dont c'est le tour et le nombre d'actions auxquelles il a encore droit.

10.3 Sa place est dans un musée

Pour finir de rendre le jeu opérationnel, nous allons ajouter les zones spéciales, la recherche de clé et l'action de récupérer un artefact, ainsi que la fin de partie.

1. Étendre le modèle pour inclure les zones spéciales, les clés que peut posséder un joueur, et l'action de récupérer un artefact. Étendre l'effet du bouton « fin de tour » pour que le joueur dont c'était le tour reçoive, aléatoirement, une clé ou rien.
2. Étendre la vue d'un nouveau panneau représentant les joueurs, et y indiquer pour chacun les clés et artefacts qu'il possède.
3. Étendre le contrôleur pour inclure l'action de récupérer un artefact et la fin de partie gagnante.

∞. Ajouter des critères caractérisant les parties perdues et arrêter la partie lorsqu'ils se réalisent. *Ces critères comprennent la disparition de l'héliport ou la noyade d'un joueur, mais vous pouvez en ajouter de plus subtils.*

10.4 La vraie aventure

Voici quatre manières possibles d'étendre votre application. Vous devez en réaliser au moins une, de votre choix. Notez que selon les choix de conception que vous avez faits jusqu'ici, certaines de ces propositions pourront demander des remaniements.

Échanges de clés. Considérons que pour récupérer un artefact, il ne faut plus une mais quatre clés du bon élément. Pour atteindre cet objectif, les joueurs devront échanger certaines des clés qu'ils ont en main.
Ajouter au répertoire parmi lequel le joueur doit choisir trois actions l'action consistant à donner une clé possédée par le joueur à un autre joueur présent dans la même zone.

Actions spéciales. La phase de recherche de clé peut également fournir des actions spéciales au joueur concerné. Le joueur peut utiliser une fois chaque action spéciale obtenue, sans que cela compte parmi les trois actions normales de chaque tour. Il y a deux actions spéciales.

1. Sac de sable : correspond à une action « assécher », qui peut viser n'importe quelle zone de l'île même éloignée du joueur.
2. Hélicoptère : permet au joueur de se déplacer vers une zone non submergée de son choix. Il peut emporter avec lui les autres joueurs présents sur la même zone de départ.

Ajouter ces issues à la phase de recherche de clé, et permettre le déclenchement de ces actions spéciales lorsque les joueurs y ont droit.

Simulation d'un vrai paquet de cartes. Dans le jeu ayant inspiré ce projet, les zones inondées et les clés obtenues ne sont pas décidées par un tirage aléatoire mais par la pioche de cartes dans deux paquets dédiés.
Créer une classe représentant un paquet de cartes et offrant les opérations suivantes : mélanger le paquet, tirer la première carte, poser une carte dans la défausse, mélanger la défausse et la replacer au sommet du paquet (cette dernière opération devant être réalisée automatiquement lorsque la dernière carte du paquet a été tirée).
Créer deux paquets de cartes concrets : un paquet contenant une carte par zone du jeu, et un paquet contenant un certain nombre de cartes pour chaque issue de la rechecherche de clé (clé de l'un des quatre éléments, rien, ou montée des eaux, auxquelles s'ajoutent encore sac de sable et hélicoptère). Faire enfin en sorte que les tirages aléatoires soient remplacés par le tirage de la première carte du paquet correspondant.
Note sur le traitement des cartes représentant les zones : une carte tirée est posée dans la défausse si la zone correspondante est encore en jeu, ou retirée de la partie si la zone correspondante est submergée. En cas de montée des eaux, la défausse est mélangée et replacée au sommet du paquet.

Personnages particuliers. Les joueurs peuvent endosser les rôles suivants au début de la partie, qui leur donnent des capacités particulières (chaque rôle ne pouvant être représenté qu'une fois) :

— Pilote : peut se déplacer vers une zone non submergée arbitraire (coûte une action).
— Ingénieur : peut assécher deux zones pour une seule action.
— Explorateur : peut se déplacer et assécher diagonalement.

— Navigateur : peut déplacer un autre joueur (coûte une action).
— Plongeur : peut traverser une zone submergée (coûte une action).
— Messager : peut donner une clé qu'il possède à un joueur distant (coûte une action).

Créer ces personnages particuliers et intégrer leurs actions à l'application (note : le messager n'a de sens que si l'extension « échanges de clés » a été réalisée).

11 TD3 Train : liaison dynamique.

On va construire une hiérarchie de classes représentant les éléments roulants d'un hypothétique compagnie ferroviaire. Les éléments décrits seront les suivants.

— La classe générale `EltTrain` regroupe tous les éléments de train (on peut imaginer qu'un train est une liste d'objets de cette classe). Chaque élément possède un nom (type `String`) et a un certain poids.
— Une `Locomotive` est un élément de train pouvant tracter d'autres éléments dans une certaine limite de poids.
— Un `Wagon` est un élément de train pouvant accueillir un certain nombre de passagers.
— Un `WagonBar` est un wagon ne pouvant pas accueillir de passagers (les passagers doivent avoir un siège réservé dans un autre élément adapté).
— Un `Autorail` est un élément de train pouvant tracter d'autres éléments dans un certaine limite de poids et pouvant accueillir un certain nombre de passagers.

On veut de plus avoir les comportements suivants.

— Il est impossible de créer un élément directement avec le constructeur de la classe `EltTrain` (on peut seulement passer par l'un des quatre autres types d'éléments).
— Chaque élément pouvant accueillir des passagers possède une méthode `boolean reserver()` qui renvoie `true` et enregistre qu'une place supplémentaire est accupée si la capacité maximale n'est pas encore atteinte, et renvoie `false` sinon.
— Tous les éléments possèdent une méthode `int calculerPoids()` qui estime le poids total de l'élément, passagers compris (on comptera 75kg par passager).
— Chaque nouvel élément créé reçoit un nom, qui est différent des noms des autres éléments déjà créés.

11.1 Organiser les classes en hiérarchies

Question 1 Deux propriétés caractérise les eléments de train, lesquelles ?

Question 2 Certain élements ont l'une, d'autres ont l'autre, y en a qu'on les deux. lesquelles ?

Question 3 Laquelle des deux propriétés est plus intéressante a hériter pour factoriser le plus de code.

Question 4, Comment faire pour que les élements qui doivent en hériter le puisse ?

Question 5 La la deuxieme proprité est caractérisée par la présence de quelle méthode ?

Question 6 Ah bon, et pour la deuxieme proprité comment l'implémenter de facon a aussi pouvoir regrouper du code qui voudrait utiliser cette méthode

Question 7 Dessiner le diagramme UML de la hierarchie obtenue, àl'exeption de la classe wagonBar.

Question 7bis Il est impossible de créer directement un element de train, ca implique quoi ?

Question 8 Quels sont parmi les attributs ceux dont la valeur ne bougera pas après leur initialisation ?

Question 9 Peut on garantir que les valeurs ne bougeront effectivement jamais ?

11.2 Constructeurs aussi en hierarchie.

Question 10 Ou c'est qu'on initialise l'attribut nom ?

Question 11 Comment faire pour facilement donner un nom différent a chaque element du train ?

Question 12 Ou c'est qu'on initialise capacité, reservation ?

Question 13 Ecrire le constructeur de eltReservable et wagon

Question 14 On voudrait quand même pouvoir dire que un wagon-bar est un wagon, donc hérite de wagon, cependant on ne peut par réserver dans un wagon bar comment faire ?

11.3 Réutiliser/Redefinir les méthodes

Question 15 Dans quelle classe se trouve la méthode réserver ? qui la réutilise ? Programmer la.

Question 16 En ce qui concerne la méthode calculerPoids(), peut on s'en tirer avec une seule définition ? Programmer.

11.4 Que se passe-t-il ?

Considérons la séquence d'instructions ci-dessous. Indiquez quelles instructions sont correctes ou incorrectes. Pour les instruction incorrectes, précisez si l'erreur est détectée à la compilation ou à l'exécution. Pour les instructions correctes, précisez le résultat obtenu. Pour l'exécution des instructions qui viennent après une instruction incorrecte, on ignorera l'effet de cette instruction incorrecte.

```
EltTrain e;
Wagon w;
WagonBar wb;
Locomotive l;
Autorail a;

e = new EltTrain(100);
l = new Locomotive(1000, 10000);
w = new Wagon(1000, 20);
wb = new WagonBar(1000);
a = new Autorail(1000, 5000, 10);
System.out.println(w.capacite);
System.out.println(wb.capacite);
System.out.println(e.nom);
System.out.println(wb.nom);
a.reserver();
w.reserver();
```