**Gérer le voleur en cas de 7 aux dés :**

Chaque case ressource (ResourceCase) contient un attribut  « boolean » qui indique si il possède le voleur. Lorsque les dés font sept, cela appelle différentes fonctions. Tout d’abord, une fonction discard() qui permet aux utilisateurs de se défausser de la moitié de leurs cartes ressources, si ils en ont plus de sept.

discard() en CLI -> Une boucle « for » sur la moitié des cartes ressources demande au joueur de se défausser des cartes nécessaires. S’il ne possède pas la ressource, un message d’erreur est affiché.

discard() en GUI -> On crée une nouvelle JFrame avec une liste des joueurs devant se défausser. Pour chaque joueur dans la liste, on ouvre des JPanel permettant de se défausser des cartes jusqu’à que la liste soit vide. On empêche la fermeture de la JFrame pour que le joueur ne puisse pas continuer sans se défausser de ses cartes.

Ensuite, une méthode thiefAction() qui se décompose en deux parties est appelée. Dans un premier temps, elle demande de nouvelles coordonnées pour déplacer le voleur. Cette méthode vérifie que ces coordonnées correspondent à une ResourceCase, puis que cette nouvelle case ne comprend pas déjà le voleur. Si ces deux conditions ne sont pas respectées, on redemande les coordonnées jusqu’à qu’elles soient valides.

En CLI -> Vérifie que les nouvelles coordonnées correspondent à une ResourceCase, puis que cette nouvelle case ne contient pas déjà le voleur. Si ces deux conditions ne sont pas respectées, on redemande les coordonnées jusqu’à qu’elles soient valides.

En GUI -> On utilise des ResourceCaseView qui ont comme attribut une ResourceCase. Seulement les ResourceCaseView sont cliquables, ce qui empêche de cliquer sur une case d’un autre type (RoadView / ColonyView / autres). De plus, la ResourceCaseView contenant le voleur n’est pas cliquable.

Ensuite le voleur est déplacer à l’aide d’une méthode switchThief(int[] coord) dans Board. Cette méthode échange le voleur avec la case de coordonnées coord en supprimant le voleur de l’ancienne case et le mettant sur la nouvelle.

Dans un second temps, thiefAction() récupère une liste contenant les joueurs qui possèdent une colonie autour de la nouvelle case. Puis le joueur actuel peut choisir un joueur parmi la liste (si celle-ci n’est pas vide) pour lui dérober au hasard une de ses ressources.

En CLI -> On demande un joueur parmi la liste tant que le joueur entré par les joueur actuel n’en fait pas partie. Ensuite on appelle player.stealResource(Player p)

En GUI -> On affiche une JFrame avec plusieurs JPanel qui contiennent les différents joueurs de la liste. Le joueur actuel doit cliquer sur l’un d’entre eux pour ensuite activer la méthode player.stealResource(Player p)

player.stealResource(Player p) -> player vole une ressource au hasard à p. Si p n’a pas de ressources, ne fait rien.

Finalement, le jeu reprend son cours.

**Acheter, stocker et utiliser des cartes de développement**

Les cartes de développement sont séparées en trois classes, implémentant DevelopmentCards (pour que chacune puisse être reconnue en tant que carte de développement): Knight pour les cartes chevaliers, Progress pour les cartes progrès et VictoryPoints pour les cartes points de victoire.

Ces différentes classes ne contiennent pas de méthodes, en effet, on effectuera les actions nécessaires en fonction du type effectif de la carte utilisée.

Il existe plusieurs types de cartes progrès et de cartes points de victoire, nous avons donc décidé de faire de Progress et VictoryPoints des Enum.

Progress -> Contient Progress.MONOPOLY pour la carte Monopole, Progress.ROAD\_CONSTRUCT pour la carte Construction de routes et Progress.INVENTION pour la carte Invention.

VictoryPoints -> Contient diverses valeurs pour plusieurs constructions (ex : des églises, des universités, etc.)

Stocker les cartes : Chaque objet Player a en attribut une liste de DevelopmentCards à laquelle on ajoute les cartes qu’il pioche.

Acheter les cartes : A la création de la partie, on crée un objet de type DevelopmentCardsDeck qui contient toutes les cartes de développement qui pourront être piochées durant celle-ci. On y ajoute deux cartes de chaque cartes points de victoire, deux cartes de chaque cartes progrès et quatorze cartes chevalier. Le deck est ensuite mélangé, puis lorsqu’un joueur achète une carte de développement il prend la première dans la liste. On vérifie évidemment qu’il a les ressources nécessaires et que le deck n’est pas vide.

Utiliser les cartes :

Les cartes de type VictoryPoints ne sont pas utilisables. Elles ne sont utiles que lors du calcul du score des joueurs.

Les cartes Knight appellent la fonction thiefAction() qui permet de déplacer le voleur et de voler une carte ressource au hasard à un des joueurs possédant une colonie autour de la nouvelle case contenant le voleur. (voir Gérer le voleur en cas de 7 aux dés)

Ensuite la méthode game.refreshMostPowerfulArmy() est appelée. Elle augmente le nombre de chevalier du joueur actuel de un et, si il a au moins trois chevaliers et plus de chevaliers que celui qui détient l’armée la plus puissante (ou si aucun ne détient ce titre), le désigne en tant que tel.

Les cartes Progress.MONOPOLY permettent d’avoir le monopole sur une ressource choisie.

CLI -> On demande une ressource au joueur qui la joue, puis on appelle une fonction game.monopoly(Resource r) qui prend toutes les ressources r de tous les joueurs et les donne au joueur actuel.

GUI -> On ouvre une JFrame contenant un JPanel ou l’utilisateur peut choisir une ressource. Ensuite la même méthode game.monopoly(Resource r) est appelée.

Les cartes Progress.ROAD\_CONSTRUCT donnent de constructions de routes gratuites au joueur.

CLI -> Une boucle « for » permet de demander des coordonnées et de poser deux routes.

GUI -> Lorsque la carte est utilisée une fonction constructRoad() de GameView est appelée. Un « int » constructRoad est alors initialisé à un, puis à chaque appel de constructRoad() il est incrémenté de un. S’il est inférieur à deux, il nous reste des routes à poser. Tous les boutons sont donc désactivés pour empêcher le joueur d’effectuer d’autres actions que poser des routes. S’il est supérieur à deux, alors les boutons sont réactivés et construcRoad est réinitialisé à zéro, ce qui permet de sortir de la boucle.

Dans RoadView, si gameView.getConstructRoad() renvoie un chiffre supérieur ou égal à 1, on rappelle la fonction constructRoad() de GameView (si c’est égal à zéro, on est pas en train d’utiliser une carte de développement).

Que ce soit en CLI ou en GUI, les fonctions putRoad() prennent le boolean early en true pour que le joueur n’ait pas à payer.

Les cartes Progress.INVENTION permettent au joueur de piocher deux cartes ressource de son choix.

CLI -> Une boucle « for » qui lui demande deux ressources et les lui fait piocher.

GUI -> Une JFrame avec deux JPanel permettant chacun de choisir une ressource.

Ensuite, dans les deux cas on utilise la fonction player.gainResource(Resource r), qui fait piocher la ressource au joueur, pour les deux ressources choisies.

**Intelligence Artificielle**

Au début de la partie, la méthode earlyGame(game) est appelée pour que l’IA pose deux colonies et deux routes.

En cours de partie, lorsque c’est au tour d’une IA, la méthode midGame(game) est appelée.

Une fonction wantsTo() lui permet de faire un choix binaire. Grâce à cette fonction on lui demande d’abord s’il veut piocher une carte de développement. Puis s’il veut en utiliser une. Les dés sont ensuite lancés. S’il a les ressources nécessaires il essaie de poser une colonie. Ensuite, il a le choix entre poser une route, poser une ville, piocher une carte de développement, utiliser une carte de développement ou faire un échange. Pour effectuer ce choix, on utilise un Random. Si l’IA ne peut pas effectuer l’action choisit, il passe son tour.

Pour choisir quelle case choisir, une fonction pour chaque type de case fait appel à la fonction findFreeCase(MovableCase c, Player p) de Board. Cette fonction retourne une liste des cases disponibles de même type que c. (pour Colony et Road les cases doivent être libre, pour Town elles doivent avoir le même joueur que celui passé en paramètres et pour ResourceCase elles ne doivent pas avoir le voleur). Ensuite la liste est mélangée puis l’IA prend la première case de la liste.

Pour les choix de ressources (par exemple les ports ou Progress.MONOPOLY) une fonction askResourceMin() permet de retourner la ressource que l’IA a en le moins d’exemplaires. Cette fonction appelle askResourceMin(Resource r) avec comme paramètre null. askResourceMin(Resource r) a le même comportement que la première, seulement si une ressource est entrée en paramètre, il ne peut pas la retourner. (c’est cette fonction qui est utilisée dans les ports pour éviter d’échanger une ressource contre la même) Il existe aussi une fonction askResourceMax() qui retourne la ressource que l’IA a en le plus d’exemplaires. Cette fonction est utilisée lorsque l’IA doit se défausser de cartes ressources. (par exemple dans la fonction discard() du voleur)

**Page de Settings**

Sur la page d’accueil, lorsqu’on appuie sur « Jouer », cela nous redirige vers la page de configuration de la partie. On peut choisir la taille du plateau (4x4 ou 6x6), le nombre de joueurs (en comptant les IA), le nombre d’IA et les noms des joueurs non-IA.

Dans le terminal, il suffit de se laisser guider par les instructions pour effectuer les mêmes paramètres que précedemment.