Comportement basé sur différents behavior. L’explorer dispose d’un attribut Behavior qui définit ses prochains coups et d’un attribut GameState qui contient l’état de l’inventaire, de la map et des points d’action. A chaque tour de jeu l’explorer envoie le GameState au Behavior et lui demande de choisir les actions. //Pourquoi ne pas le garder en attribut s’il ne change pas ?

Les actions du jeu sont représentées par des objets de type Action auxquels on peut passer le nom de l’action et ses éventuels paramètres, et appeler la commande toJSONString pour le transformer en String compatible pour le moteur de jeu.

Un Behavior contient une queue d’action qui lui permet de définir son comportement pour plusieurs tours. Quand on demande à un behavior d’effectuer la prochaine action, il vérifie la queue. Si elle n’est pas vide, il se contente d’effectuer la prochaine action. Sinon il « réfléchit » une action propre à son comportement. //pas mal pour éviter la centaine de if

Le bot commence avec l’InitialBehavior dont le but est de définir les boundaries de la map. On est au courant au début du jeu du bord de la map où le drone spawn. Ce behavior va donc d’abord envoyer 2 echo, Nord et Sud si on spawn à l’Ouest ou à l’Est, et vice-versa. //Toujours gauche droite (facile à obtenir à partir de l’enum Heading (-1,+1)%4). Puis il se déplace vers le bord qui lui est le plus proche, et effectue un dernier echo dans une troisième direction afin de définir exactement les dimensions de la map. Il instancie alors une map d’une dimension correcte, puis change l’état de l’Explorer.//changer le behavior

DroneExplore : le but de cet état est d’explorer un minimum la carte. Comportement encore à définir précisément. En tout cas il faut minimum une crique. Quand on trouve une crique, explorer les blocs alentours. Change d’état quand on estime qu’assez de case ont été trouvée (par rapport ou pas aux contrats ? Peut etre trop précis). //Crique : Minimal. Appelle ensuite la phase terrestre.

La phase terrestre alterne entre deux états, exploration et récolte (Un troisième pour le craft OU l’inclure dans récolte). Il faudrait avoir un objet Stats//appelé résults qui record tous les résultats des actions du bot, ce qui permet de prévoir le nombre de PA par point de ressource etc (mettre une limite fiable d’échantillons, par exemple pour 1 échantillon de bois récolté on peut pas dire combien de PA ça prendra par bois en grande quantité).

Concernant les stats, comment traiter les string json que renvoie le moteur de jeu ? Plusieurs idées : la classe abstraite Behavior implémente une méthode acknowledgeResults et chaque état serait capable de traiter les données pour l’aider à sa décision. Mais peut etre meilleure idée : la classe Stats s’en charge, design pattern observer des behavior sur les Stats pour les tenir à jour ?

Alternance entre explore et récolte : trouver des cases contenant les ressources. Puis déterminer s’il vaut mieux chercher encore ou si ça vaut le coup de try sur ces cases (selon la difficulté ou la distance par exemple). Se baser sur les cases connues pour en trouver d’autres, selon l’altitude par exemple. Une classe chargée de trouver où il est plus probable de trouver certains biomes ?

Pour une première implémentation histoire de finir un contrat (puisqu’ils deviennent de plus en plus dur), une simple alternation explore/recolte sans trop réfléchir ? //OK