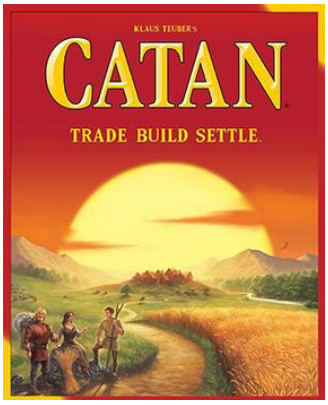
Catan
Projet de Programmation S4
Rapport de projet - Groupe VR2C

Sommaire

Introduction



Catan est un jeu de société stratégique, devenu l'un des jeux de plateau modernes les plus populaires depuis sa première publication en 1995.

Dans le jeu Catan, les joueurs assument le rôle de colons cherchant à coloniser une île nouvellement découverte appelée *Catan*. Le but du jeu est d'accumuler un certain nombre de **points de victoire**, généralement obtenus en construisant des colonies et des routes, en obtenant des **cartes de développement** et en réalisant des objectifs spécifiques.

Pour pouvoir construire ces bâtiments et acheter ces cartes de développement, il est nécessaire de **récupérer des ressources**, au moyen des colonies et des villes que l'on vient placer sur la carte.
Des **échanges** et de la **négociation** sont aussi possibles entre joueurs ce qui rajoute une vraie dimension **diplomatique**.

Catan est donc un jeu proposant une expérience très complète, et de nombreuses variantes existent venant rajouter du contenu de jeu supplémentaire.

Organisation du projet

Pour ce qui est de l'organisation du projet, nous avons employé une méthode d'organisation **Agile** de type **Scrum** pour le bon fonctionnement de l'avancement du projet.

Chaque mardi matin, nous tenions une **réunion** avec notre chargé de projet pour faire un point sur l'avancement des travaux. Cette rencontre était l'occasion de discuter des **progrès réalisés**, des **obstacles rencontrés**, et des **objectifs à atteindre** pour la semaine à venir. Immédiatement après cette réunion, notre équipe de cinq se réunissait pendant une heure pour une **séance de type scrum.**

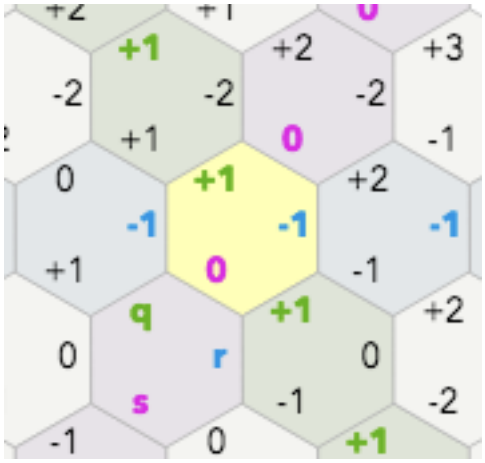
Ces réunions scrums avaient pour but d'établir les **issues** à réaliser pour la semaine suivante. Chaque membre de l'équipe pouvait ainsi **exprimer ses idées, partager ses difficultés** et **proposer des solutions.**

Nous utilisions **GitLab** pour la gestion de notre **code source** et le **suivi des tâches**. Grâce à Git, chaque développeur pouvait cloner le dépôt central sur sa machine locale, travailler sur une branche dédiée et soumettre ses modifications via des **merge requests**. Cela permettait une **intégration continue** et un contrôle de version rigoureux, **facilitant la collaboration** et **réduisant les risques de conflits de code.**

Les issues, créées et gérées dans GitLab, étaient assignées selon les compétences et les disponibilités de chacun. Cette méthode de travail nous a permis de maintenir une vision **claire** de l'avancement du projet, **d'anticiper les besoins en termes de développement,** et de réagir rapidement aux imprévus.

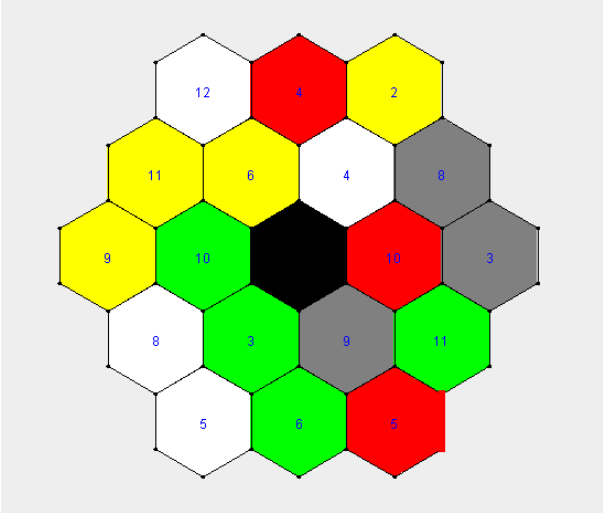
En plus de l'intérêt de GitLab, nous avons mis en place une **Pipeline** muni d'un **Linter** et d'un Build pour assurer le style de code **unifié** sachant que nous avons tous des habitudes différentes.

Implémentation de la carte



Le plateau de jeu de Catan est un hexagone composé **d'hexagones**. Chaque hexagone contient **une ressource** et **une valeur de dés**. On place des **routes** sur les **arêtes** des hexagones et des **villes** et **colonies** sur les **sommets** des hexagones. Lorsqu'une valeur tombe aux dés, si on a une colonie ou une ville sur le sommet d'un hexagone contenant cette valeur, on obtient des ressources.

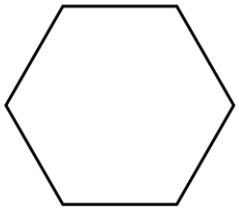
L'enjeu majeur de l'implémentation de la carte était de faire un **pavage hexagonal** avec des coordonnées pour retrouver efficacement les hexagones **en fonction de la position** de la souris sur l'écran. En effet, on veut pouvoir cliquer, par exemple, sur une arête pour placer une route, donc il faut pouvoir retrouver, en fonction de la position de la souris, quelle est l'arête la plus proche. Il fallait enfin pouvoir éviter de placer des villes ou des colonies à moins de deux sommets de distance les unes des autres.



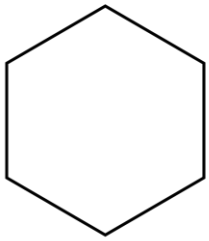
Le système de coordonnées utilisé pour les hexagones est un système de **coordonnées cubiques**. En effet, on a trois axes sur un hexagone (trois paires d'arêtes face à face) donc trois coordonnées. Les coordonnées sont appelées r, q et s, et pour chaque hexagone, leur **somme** est **toujours égale à zéro**, ces coordonnées étant des valeurs entières. Chaque coordonnée cubique représente le **centre d'un hexagone**. On a l'hexagone central de coordonnée (0,0), puis on trace les hexagones jusqu'à une distance de deux de celui-ci pour créer la carte de Catan. Les coordonnées cubiques se trouvent dans la classe *CubeCoordinates*.

Les coordonnées cubiques représentant le centre exact d'un hexagone, on a donc implémenté des coordonnées cubiques fractionnelles dans la classe *FractionalCubeCoordinates*. Elles servent à indiquer où l'on se situe exactement dans l'hexagone, afin de savoir quels sont les hexagones les plus proches de la position de la souris.

Pour le pavage hexagonal, on peut avoir soit un côté pointu tout en haut (***Pointy***), soit un côté plat (***Flat***). On utilise pour cela la classe *Orientation*, où l'on indique les angles des sommets (par rapport à un cercle trigonométrique). La classe *Layout* prend ensuite une *Orientation*, une taille et une origine. L'origine est un point sur l'écran, et la taille est un point qui représente la distance au point (0,0) sur l'écran. Cette taille sera celle des arêtes de chaque hexagone. La classe *Layout* s'occupe surtout des **conversions** entre les coordonnées écran, fractionnelles ou cubiques.



Hexagone en mode ***Flat***



Hexagone en mode ***Pointy***

Tout le reste s'opère depuis la classe *GameBoard*, où l'on **dessine les hexagones** et où l'on sélectionne les arêtes et sommets les plus proches. En effet, on voulait ensuite pouvoir placer les villes et les routes. Il fallait donc, pour les villes, **trouver les trois hexagones les plus proches**, et pour les routes, les deux les plus proches de la position de la souris. Cela était utile pour ensuite comparer les sommets entre eux et trouver quels sont les sommets ou arêtes communs à deux ou trois hexagones.

Pour les villes, chaque sommet de l'hexagone est partagé par trois hexagones adjacents. Une fois qu'on a identifié les trois hexagones les plus proches, on compare les sommets de ces trois hexagones. Lorsqu'on trouve le sommet commun aux trois hexagones, on peut placer la ville à cet endroit, sous réserve qu'il n'y ait pas déjà une ville ou une colonie sur ce sommet ou sur un sommet adjacent, respectant ainsi la règle de distance minimale.

Pour les routes, chaque arête est partagée par deux hexagones **adjacents**. Pour placer une route, on identifie les deux hexagones les plus proches de la position de la souris. Ensuite, on détermine l'arête commune à ces deux hexagones. On vérifie que cette arête n'est pas déjà occupée par une autre route et que les règles de connexion sont respectées : il doit y avoir une colonie, une ville, ou une route déjà existante à une extrémité de la nouvelle route.

Le système de coordonnées cubiques **facilite** ces vérifications en permettant de convertir facilement les positions sur l'écran en coordonnées hexagonales et vice-versa. Grâce à la classe Layout, qui gère les conversions entre les coordonnées écran et les coordonnées hexagonales, on peut **rapidement et précisément** déterminer quels hexagones sont les plus proches de la position de la souris et quelles sont les arêtes et les sommets correspondants.

La classe *GameBoard* utilise ces outils pour **dessiner** le plateau de jeu, gérer les interactions des joueurs avec la carte et appliquer les règles de placement des

routes, villes et colonies. En s'appuyant sur le pavage hexagonal et les coordonnées cubiques, le système **assure une gestion efficace et précise des éléments du jeu**, garantissant une expérience de jeu fluide et conforme aux règles de Catan.

Un dernier outil utilisé est les classes *Tile*, *TileEdge* et *TileVertex*. En effet, ces classes sont principalement utilisées pour **stocker et réunir les informations** sur les arêtes, sommets et hexagones. On les stocke dans des ***hashmaps*** pour pouvoir retrouver efficacement les objets recherchés, grâce à une clé qui est la coordonnée de l'objet. Les Tiles contiennent notamment **un type de ressource** et **une valeur de dés**. Les *TileVertex* contiennent **un bâtiment**, et **un port** s'il y en a un, tandis que les *TileEdge* contiennent **un bâtiment, qui est une route**. De plus, ils contiennent tous leurs coordonnées (pour les arêtes, un point début et un point fin de l'arête).

En résumé, l'utilisation du pavage hexagonal et des coordonnées cubes permet de :

1. **Identifier rapidement** et avec précision les hexagones, arêtes et sommets en fonction de la position de la souris.
2. Appliquer **les règles de placement** des villes et des routes, en évitant les erreurs de placement.
3. **Faciliter les conversions** entre les coordonnées écran et hexagonales, assurant une **interaction utilisateur optimale.**

Ce système **robuste et efficace** est essentiel pour la bonne marche du jeu, permettant aux joueurs de **se concentrer sur la stratégie et le plaisir du jeu** sans se soucier des aspects techniques du pavage et du placement.

Réseau (Chat et Chatlog) – Bots

Nous avons décidé d'implémenter le jeu **en réseau**. Ce choix a été fait car il permet à plusieurs joueurs de rejoindre une partie sur des ordinateurs **différents**. Pour cela nous avons utilisé des **sockets TCP/IP** (Transmission Control Protocol/Internet Protocol). Ce choix a été fait pour plusieurs raisons :

- **Fiabilité** : Lors d'une partie de Catan il est crucial que chaque action réalisée par les joueurs soit transmise aux autres afin que tous les joueurs aient exactement la même partie sur son ordinateur. Les sockets TCP garantissent alors que chaque message envoyé arrivera à destination sans perte.
- **Ordre** : Les messages doivent arriver dans l'ordre précis où ils ont été envoyés afin de maintenir le bon fonctionnement du jeu, Les sockets TCP garantissent aussi cela.
- **Connexion Orientée** : Lorsqu'un utilisateur se connecte au serveur, cette connexion est maintenue tout au long de la partie. Cela permet de gérer plus facilement les cas de déconnexion et de garantir que chaque utilisateur est toujours connecté à la partie.

Afin d'établir la connexion avec un serveur, nous avons opté pour **un serveur local.** Un joueur doit donc utiliser son ordinateur comme serveur et les autres doivent se **connecter directement** à son ordinateur. Si le joueur crée un serveur, une instance de Server **est créée** ce qui génère un nouveau **ServerSocket**. Celui-ci va attendre que les joueurs se connectent.
Chaque joueur doit donner **l'adresse IP** du serveur afin d'établir la connexion. Pour chaque connexion entrante le serveur crée une instance de **ClientHandler** qui va gérer sur un nouveau thread tous les messages **sortants et entrant** du joueur.

A chaque action d'un joueur, un message est envoyé au serveur. Celui-ci le récupère depuis le thread du joueur et le renvoie à tous les joueurs **en synchronisant les flux de sorties.**

Afin de gérer au mieux les messages les seuls objets passés dans le réseau sont des **instances de la classe *NetworkObject*** qui contient **l'id de l'utilisateur** ayant envoyé le message, un message indiquant l'action à réaliser, un TypeObject afin de faire un premier tri sur les messages et un **objet sérialisable** pour faire passer d'autres informations dans le réseau.

Le réseau a été une difficulté de taille dans la réalisation du projet car il a nécessité beaucoup de **documentation** et **d'apprentissage** en amont afin de comprendre son fonctionnement. Avec plus de temps nous aurions pu nous concentrer davantage sur les erreurs possibles qu'engendre le réseau, par exemple lorsqu'un joueur quitte la partie avant la fin.

Nous voulions qu'en plus du réseau, l'utilisateur **puisse jouer sans autres joueurs présents.** Nous avons donc **ajouté des bots** (héritant de la classe joueur), c'est-à-dire des joueurs jouant automatiquement.

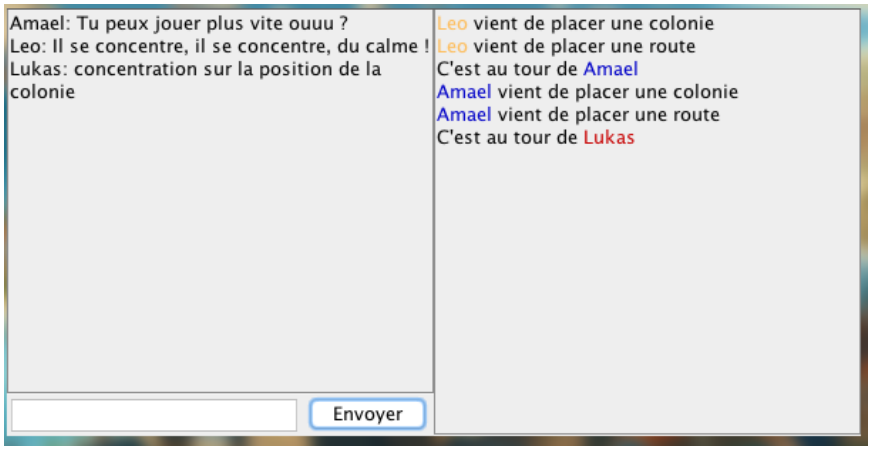
Lorsqu'un tour de bot commence, celui-ci **crée un thread** dans lequel il exécutera les actions possibles. La **première idée** a été d'implémenter les possibles coups du bot de façon **totalement aléatoire**. Puis nous avons ajouté des **algorithmes permettant de calculer les coups** qui permettraient au bot de gagner.

Celui-ci **pose donc ses colonies** en fonction des **plus grosses probabilités du dé** ainsi que des **meilleures ressources** (par exemple celles moins présentes sur le plateau). Les routes sont calculées afin de **continuer une route déjà existante** (pour envisager la route la plus longue). S'il lui manque des ressources et qu'il peut faire un échange (qu'il a des ressources **non nécessaires** à donner) alors il propose un trade aux joueurs qui peut lui donner **le plus de ressources nécessaires**. Les bots privilégient de **construire des villes et des colonies** afin de maximiser l'apport des points de victoire. Et seulement après, il cherche à **construire des routes et piocher des cartes.**

Afin de tester si les bots sont capables de finir le jeu (ce qui était pour nous le minimum à avoir pour pouvoir utiliser ces bots), nous avons mis en place **un mode bots solo** où les bots disputent ensemble une partie. Ce mode a été mis en place de manière à ce que **la partie soit rapide** notamment **en enlevant les animations** de dés et **les temps d'attente des bots** utilisés pour améliorer l'expérience utilisateur.

Ce mode permettrait aussi de comparer deux versions de bots afin de trouver le plus efficace.

Le ChatLog est une fenêtre permettant d'avoir un historique du jeu ainsi qu'**un chat avec les autres joueurs**.
Le chat des joueurs est construit **à partir du réseau**, lorsque l'utilisateur rentre un message dans le *JTextField* ce message est **transmis au serveur** qui le **retransmet** à tous les autres joueurs. Il est ensuite affiché sur le chatlog.
En plus de cette fonctionnalité, le **chatlog est utilisé à des fin d'historique du jeu**, chaque action est retranscrite dans le *JTextPane* afin d'en **informer le joueur**. Les événements du dé de 20 sont eux aussi affichés ce qui permet aux joueurs de les découvrir.



Dé à 20 faces

Un **dé à 20 faces** lancé par les joueurs une fois par tour de table a été implémenté. **Chaque face** correspond à un **événement unique**, ajoutant de la complexité au jeu.

Description des Événements

Chaque événement est conçu pour apporter une nouvelle dimension au jeu, **sans déstabiliser complètement l'expérience de jeu**. Voici la liste détaillée des événements associés aux 20 faces du dé :

1) Épidémie chez les moutons : Tous les moutons meurent d'une maladie étrange (excepté, *grâce à un miracle*, le mouton du tutoriel).

Impact : **Moyen à fort**. Affecte principalement les joueurs dépendant de la laine.

2) Affichage des cartes de développement : Les cartes de développement actuellement possédées par les joueurs s'affichent dans le ChatLog.

Impact : **Faible à moyen.** Influence la stratégie et la diplomatie.

3) Noël : Chaque joueur reçoit deux ressources aléatoires.

Impact : **Moyen à fort.** Avantage distribué uniformément.

4) Activation du voleur : La case sur laquelle le voleur se trouve s'active exceptionnellement une seule fois.

Impact : **Moyen.** Récolte inattendue de ressources.

5) Échange des mains : Le joueur ayant le plus de points de victoire échange sa main avec celui en ayant le moins.

Impact : **Très fort.** Change radicalement la dynamique de jeu.

6) Perte de ressource : Tous les joueurs perdent une ressource.

Impact : **Moyen.** Pénalité modérée.

7) Inondations : Les ports sont désactivés pendant deux tours.

Impact : **Moyen à fort.** Affecte le commerce et les stratégies reposant sur le commerce maritime.

8) Échange des probabilités : Les numéros des cases avec des probabilités fortes échangent avec celles de probabilité faible.

Impact : **Très fort.** Change la stratégie des récoltes.

9) Cartes chevaliers : Chaque joueur pioche son nombre de chevaliers en cartes ressources.

Impact : **Fort.** Avantage significatif pour certains joueurs.

10) Inversion de l'ordre de jeu : L'ordre de jeu s'inverse.

Impact : **Faible.**

11) Brouillard : Un brouillard s'abat sur la carte, faisant disparaître les cases pendant deux tours.

Impact : **Très fort.** Complique la planification.

Le jeu avant…

…et après l'activation du brouillard

12) Arnaqueur au marché : Les échanges avec les autres joueurs peuvent être altérés.

Impact : **Moyen.** Introduit de l'incertitude dans les échanges.

13) Seuil du voleur : Les joueurs en dessous du seuil du voleur (7) doivent abandonner la moitié de leurs cartes.

Impact : **Fort.** Pénalité significative.

14) Point de victoire : Le joueur avec le moins de points de victoire gagne un point de victoire.

Impact : **Moyen.** Aide à équilibrer le jeu.

15) Incendie : Un incendie fait rage dans les forêts et champs de la carte. Tous les joueurs perdent leur bois et leur blé.

Impact : **Très fort.** Affecte le développement.

16) Perte de ressource par bâtiment : Chaque joueur perd une ressource aléatoire par bâtiment posé sur la carte.

Impact : **Fort.** Pénalise les joueurs développés.

17) Anniversaire : C'est l'anniversaire du joueur qui vient de lancer le dé, chacun lui donne une ressource aléatoire.

Impact : **Moyen.** Avantage personnel.

18) Pot commun : Tous les joueurs mettent une ressource dans un pot commun.

Impact : **Moyen.** Redistribution des ressources.

19) Récupération du pot commun : Le joueur ayant lancé le dé récupère le contenu du pot commun.

Impact : **Fort.** Avantage significatif.

20) Double gain : La valeur des dés tombée ce tour retombe, permettant de gagner à nouveau les ressources.

Impact : **Moyen à fort.** Double gain de ressources.

Cette amélioration de Catan a pour vocation une **d'augmenter significativement les rebondissements** lors d'une partie et l'interaction entre les joueurs, les événements obligeant les joueurs à **adapter constamment** leur stratégie.

Pistes d'extensions / Chemins abandonnés (phase de coopération, économie, menu multiplayer, implémentation meilleure du réseau)

Pistes d'extensions :

1. **Modification de l'Implémentation du Réseau**

Le serveur **n'est pas le gestionnaire de la partie**, ce qui nécessite le lancement de **plusieurs instances en local** qui communiquent entre elles par le biais du serveur. Nous pourrions améliorer cette configuration en faisant en sorte que **le serveur devienne le gestionnaire central** de la partie et qu'une seule instance de partie existe. Cela permettrait de **faciliter la gestion des parties**, **de réduire les risques de désynchronisation** et **d'améliorer l'évolutivité et la maintenabilité du système**.

2. **Menu Multijoueur pour choisir et créer un serveur**

Pour lancer une partie en multijoueur il faut qu'un joueur **ouvre un serveur et que les autres le rejoignent grâce à l'adresse IP**. Ce menu pourrait inclure des **options pour configurer les paramètres de la partie** (présence du dé 20, nombres de joueurs, etc) et aurait pour but de **faciliter** la connexion entre les joueurs et de permettre la **création de parties publiques et privées.**

3. **Compatibilité Multi-Plateforme Immédiate**

Développer la **compatibilité multiplateforme** pour permettre aux joueurs sur PC et mobile de jouer ensemble. Cette extension profiterait grandement de l'interface utilisateur actuelle entièrement faite à base de clics.

Chemins Abandonnés :

1. **Economie autour des Ports**

Mettre en place une **économie dynamique pour les ports**, où une ressource trop échangée devient **rare** et donc **plus coûteuse** à échanger. Par exemple, si **trop de joueurs échangent du blé**, le blé deviendrait une ressource **abondante** par rapport aux autres et donc **le coût en ressource pour obtenir d'autres ressources avec du blé augmenterait.**

Cette idée a été abandonnée car elle a été estimée **inutile** à cause des **interactions des joueurs par les échanges**, qui créait entre eux **une économie artificielle.**

2. **Marchand itinérant**

Cette idée propose d'introduire un **marchand itinérant** qui se **déplace sur la carte** et **propose des échanges** de ressources aux joueurs. Ce marchand pourrait offrir des **échanges uniques ou rares**, ajoutant une dynamique supplémentaire au **commerce du jeu et à la stratégie**.

Idée abandonnée car trop complexe par rapport à son impact sur la partie.

3. Mode Coopératif

Le but était de créer une phase où les joueurs avaient pour but de **s'allier pour atteindre des objectifs communs**, comme lutter contre des **menaces extérieures** ou gérer des **catastrophes naturelles**. Les joueurs ayant **le plus contribué** à la phase de coopération auraient été **récompensés**.

Cette idée a été abandonnée à cause du peu d'impact qu'elle aurait eu sur la partie.

4. Génération de la carte

Les valeurs des tuiles sont réparties **semi-aléatoirement** (les 8 et 6 sont toujours au même endroit, le reste est généré aléatoirement). Cela a été fait pour avoir une **bonne répartition des tuiles de hautes probabilités**. L'idée était de pouvoir avoir une **carte totalement générée aléatoirement** afin d'avoir des parties différentes.

L'idée a été abandonnée en raison de sa complexité, qui aurait eu un **impact disproportionné** sur le déroulement de la partie.

5. Progression et quêtes

L'idée était de permettre de créer une **progression sous forme de niveaux** de profil et de **quêtes** afin **d'inciter le joueur à jouer pour augmenter son niveau**. Une autre idée était de **créer des quêtes inclues** dans la partie et récompensant le joueur avec des ressources. Cela avait pour but de **dynamiser les parties** et créer de **nouvelles stratégies**. Cependant cela a été abandonné par son manque d'intérêt par rapport aux demandes du cahier des charges.

6. Ajouts d'une nouvelle monnaie

L'idée était **d'ajouter une nouvelle monnaie** appelée "Or" qui aurait pu être **utilisée dans les échanges** à la banque comme une ressource joker. Par exemple, un joueur possédant 3 bois et 1 or peut **échanger ces ressources contre une autre ressource de son choix.** L'or aurait été **obtenu lorsque le dé ne fournit pas de ressources** au joueur.

Cela aurait permis au joueur de recevoir des ressources **quoi qu'il arrive**, évitant ainsi les tours à "blanc", et rendant l'expérience de jeu plus **dynamique**. Cette nouveauté a été abandonnée par manque de temps et elle va de pair avec la **métropole**.

7. Métropole

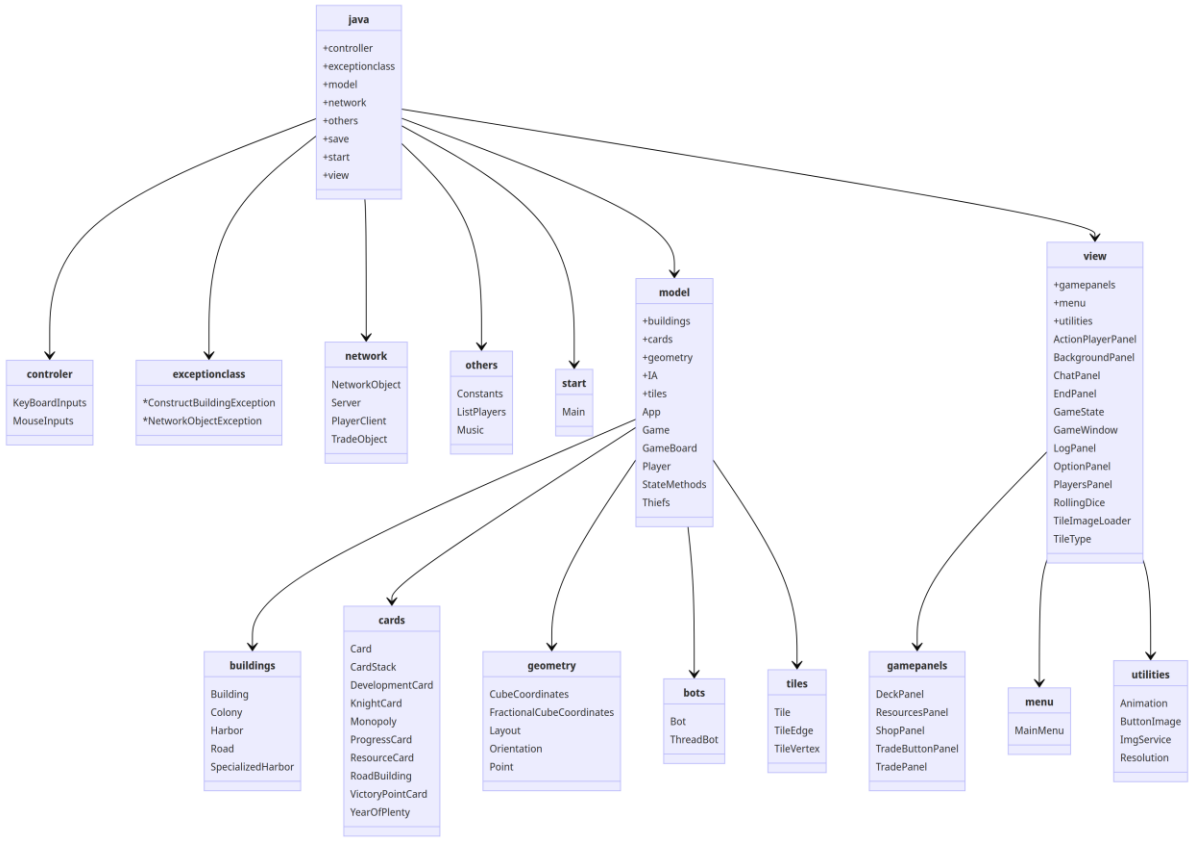
Une métropole aurait été un **nouveau bâtiment** plaçable uniquement lorsque 1 ville et 2 colonies étaient reliées par des routes. Elle aurait remplacé la ville présente et permis de **générer 3 ressources**. Elle aurait permis **d'augmenter le nombre de cartes maximum que le voleur accepte**, mais aussi de débloquer la génération d'or. Elle apporte **3 points de victoire**. Elle a été abandonnée par manque de temps et d'intérêts dans le mode à 4. Elle est plus intéressante dans un mode **1 contre 1.**

8. Elevage le plus prospère

Nouvelle carte spéciale “**élevage le plus prospère**” qui fait gagner **2 points de victoires** à la manière de la carte “armée la plus grande” et “route la plus longue”. La carte est donnée au joueur qui a le **plus grand nombre de moutons** dans son élevage. **Un joueur peut mettre des moutons dans son élevage** lors de son tour et il ne pourra **plus jamais accéder à ces moutons**. Cependant ces moutons sont **invulnérables aux événements extérieurs** *(comme le mouton du tutoriel).* Abandonnée par manque de temps.

Les pistes d'extension proposées dans ce rapport visent à **enrichir l'expérience de jeu** et à **offrir de nouvelles fonctionnalités aux utilisateurs**. En abandonnant certains **chemins trop complexes**, nous avons pu nous concentrer sur des **extensions réalisables** qui apporteront une réelle valeur ajoutée. Ces idées peuvent servir de base pour de futurs **développements**, assurant ainsi une évolution continue et l'amélioration de notre jeu de **Catan**.

Annexe



Graphe des classes