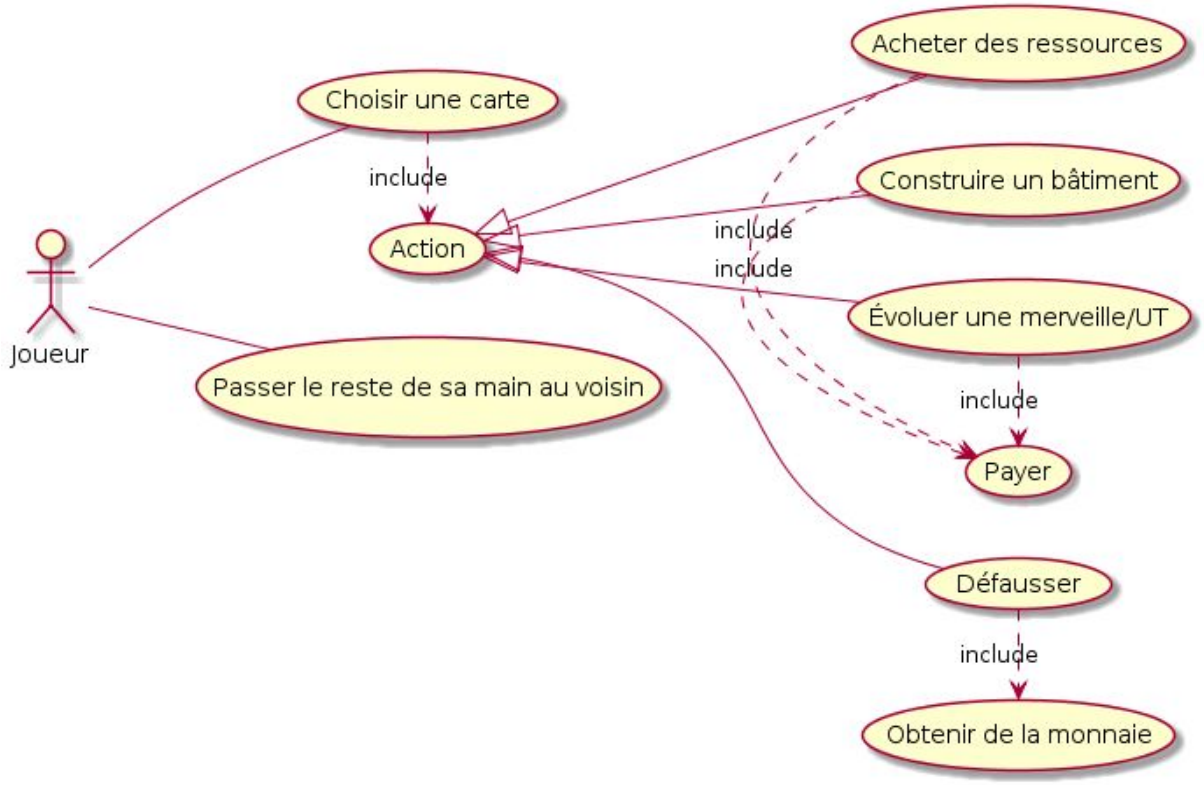
7 Wonders UTBM
Rapport JAVA

ARAHHOU Hicham
LI Jiale
VASSEUR Aurélien

TABLE DES MATIÈRES

1 - Définir le comportement des acteurs - Diagramme d'utilisation

Afin d'initier au mieux la conception du programme, il est judicieux de commencer par s'interroger sur son rôle et la manière dont les utilisateurs interagissent avec. Ainsi, nous avons commencé par dresser un diagramme de cas d'utilisation de notre programme.

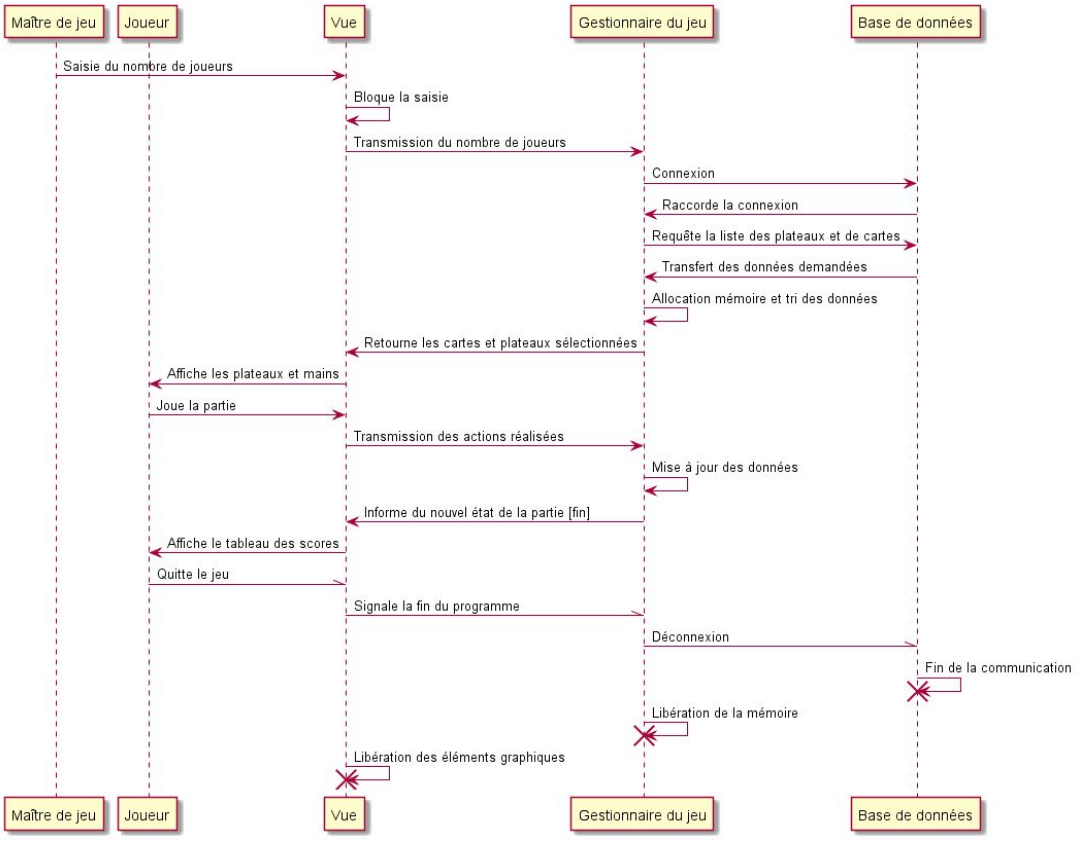


Dans notre cas, la situation est assez simple : il n'existe qu'un seul type d'acteur interagissant avec notre programme : un joueur. Ainsi, en s'intéressant aux diverses actions que celui-ci peut mener, nous pourrons ensuite bâtir l'application adaptée.

Dans le cadre du jeu 7 Wonders, le joueur ne s'exprime qu'en pleine partie (on néglige le fait de lancer/quitter l'application). Puisque le choix de la merveille est aléatoire, le joueur effectue des actions qu'au travers des cartes en jeu. Ainsi, celui-ci peut piocher une carte et effectuer une action liée. Une fois son tour (qui se résume à sélectionner une carte et réaliser une action) terminé, il passe le restant de sa main au joueur adjacent.

2 - Squelette de l'application - Diagramme de séquence

Grâce au précédent diagramme des cas d'utilisations de notre programme, nous connaissons dorénavant le comportement des acteurs interagissant avec. Par conséquent, nous sommes en mesure de déterminer la manière dont l'ensemble du programme composé de l'interface graphique (ou Vue), le gestionnaire du jeu (association Controller-Models) et la base de données coopèrent.



Afin de distinguer l'acteur (unique) sélectionnant le nombre de joueurs en début de partie, nous l'appelerons "Maître de jeu", bien qu'il soit assimilable à un joueur dans les faits.

Une fois le nombre de joueur connu, la mise en place du jeu est possible :

- instanciation des éléments en mémoire
- récupération des données (cartes, merveilles) présent en base
- affichage des éléments graphique pour chacun des joueurs

On notera ici que chaque joueur possède une fenêtre dédiée. Puisque le rendu du projet se fait par l'intermédiaire d'un unique programme, le programme "client" est mimé via ces fenêtres séparées pour les joueurs. Ainsi, un processus est associée à chacune d'entre elle, permettant à tous les joueurs de jouer simultanément (et de les faire patienter si certains joueurs n'ont pas encore fini de jouer).

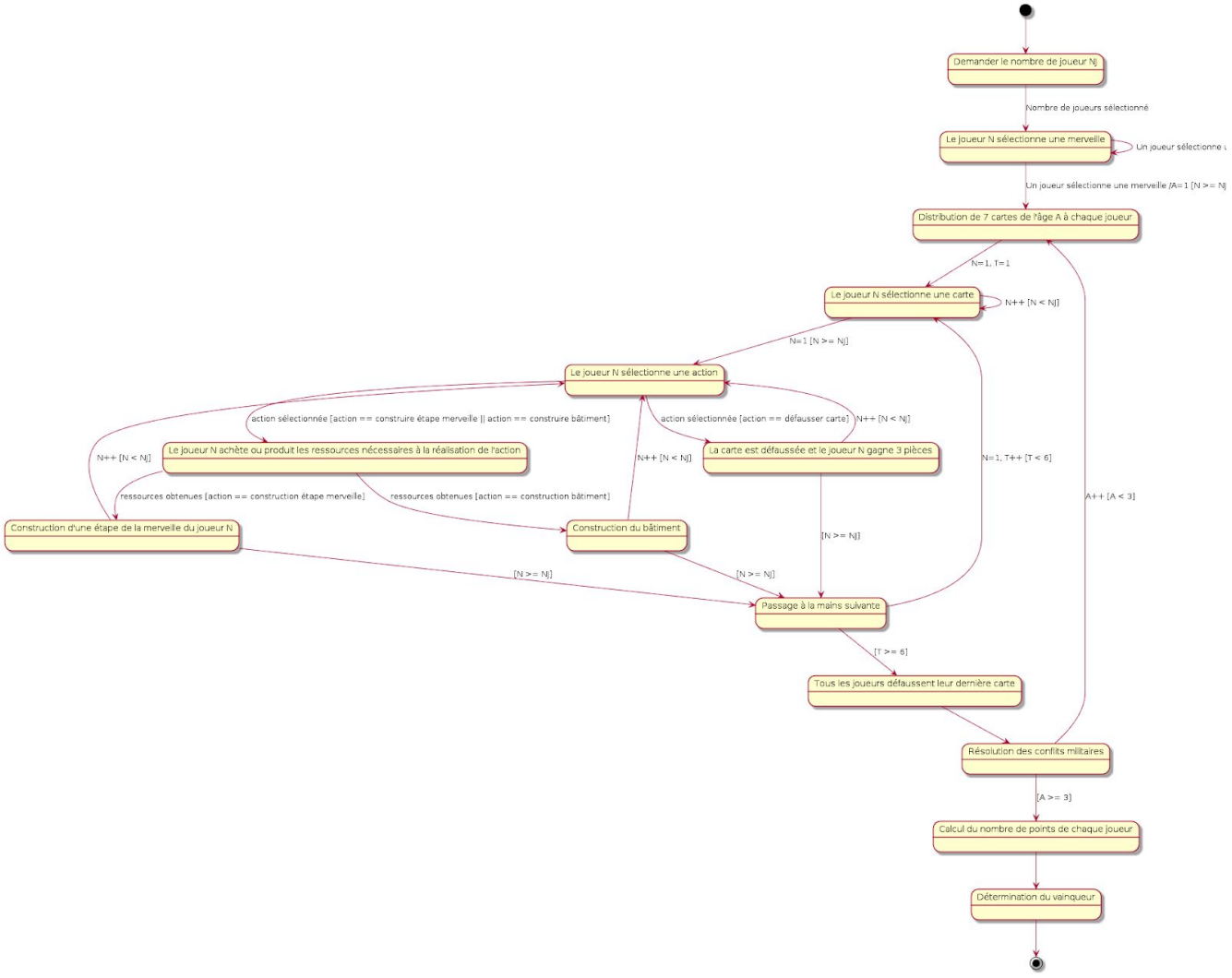
Le déroulé d'une partie en elle-même se focalise grandement sur les règles du jeu et n'est donc pas traité dans le diagramme de séquence qui ne se contente que de donner la direction générale de l'ensemble du programme.

Evidemment, au cours de la partie, le gestionnaire de jeu met à jours les différentes données (scores, cartes jouées, évolution dans la partie, etc.) afin d'assurer le bon déroulement d'une partie.

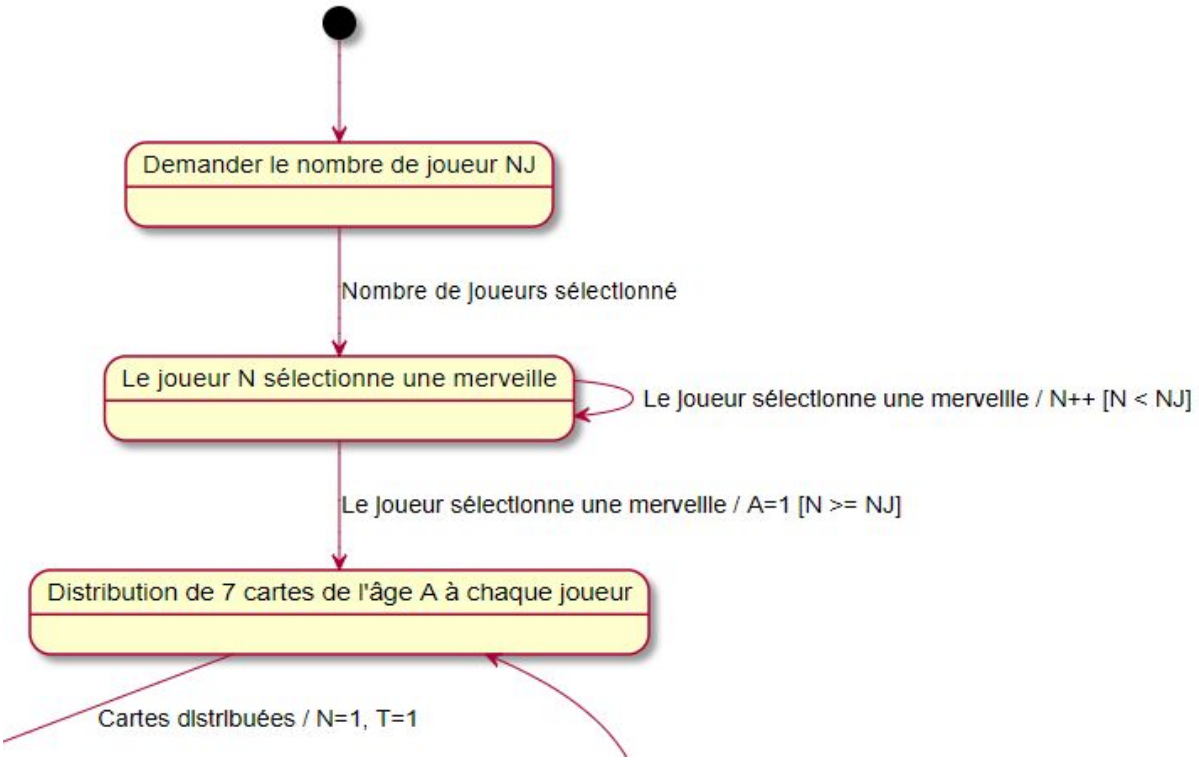
Lorsqu'une partie s'achève, les résultats des différents joueurs (scores et classement) sont affichés et il est naturellement proposé de recommencer une nouvelle partie ou de tout simplement quitter le jeu. Lorsque le joueur quitte le jeu (en interagissant avec la Vue), l'information est transmise à toutes les entités du programme afin de correctement libérer l'espace mémoire par déréférencement (pour que le garbage collector puisse réaliser sa mission).

3 - Déroulement d’une partie - Diagramme état/transition

Nous allons, dans cette troisième partie, nous intéresser au déroulement d’une partie. Nous nous appuierons pour ce faire sur le diagramme état / transition présent ci-dessous. Un lien permettant la visualisation du diagramme est présent en annexe.



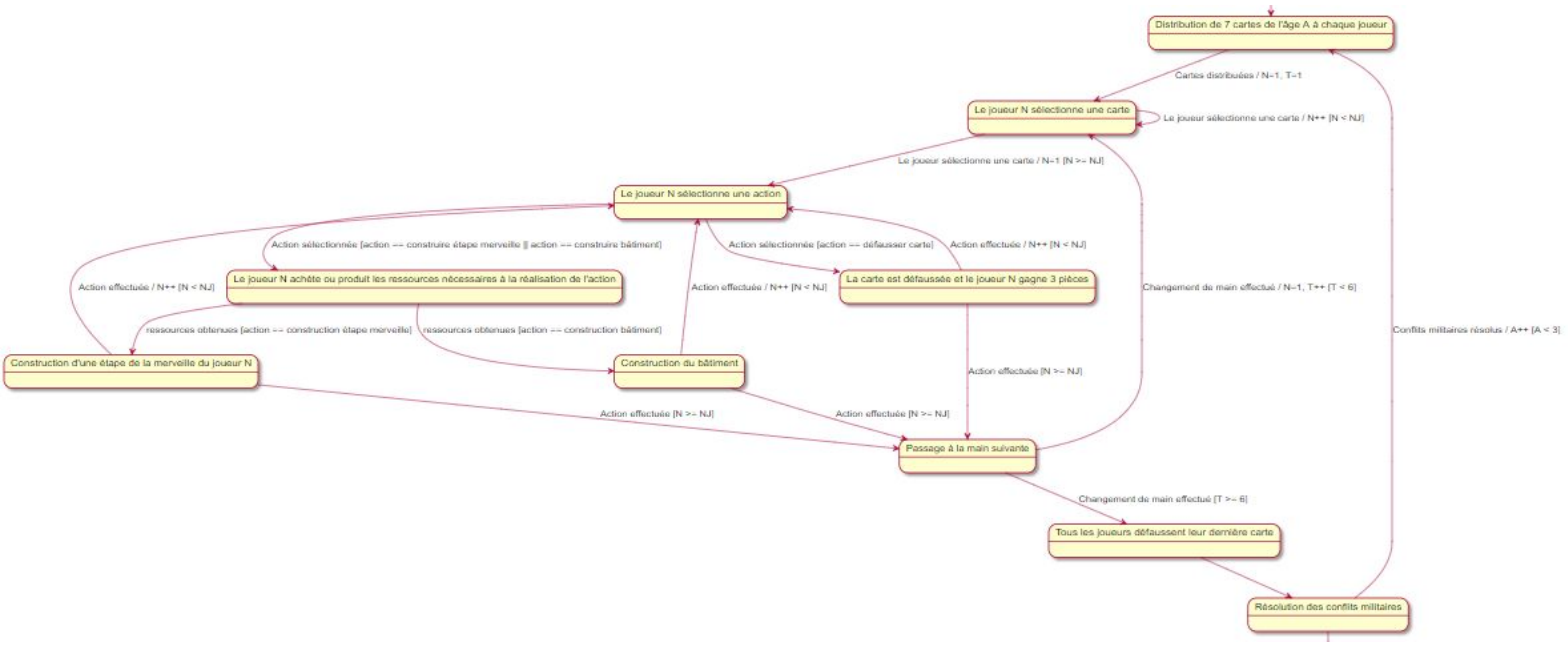
3.1 - Préparation de la partie



La première étape consiste à préparer les différents éléments qui sont nécessaires au bon déroulement de la partie. Nous allons en effet commencer par demander à un utilisateur de renseigner le nombre total de joueurs. Suite à cela, nous demanderons à chacun d'entre eux de sélectionner une merveille.

Une fois ces sélections effectuées, nous pouvons démarrer la partie. Une partie est composée de 3 âges, représentés ici par la variable A, qui sont eux-même composés de 6 tours, représentés par la variable T.

3.2 - Le cycle des âges

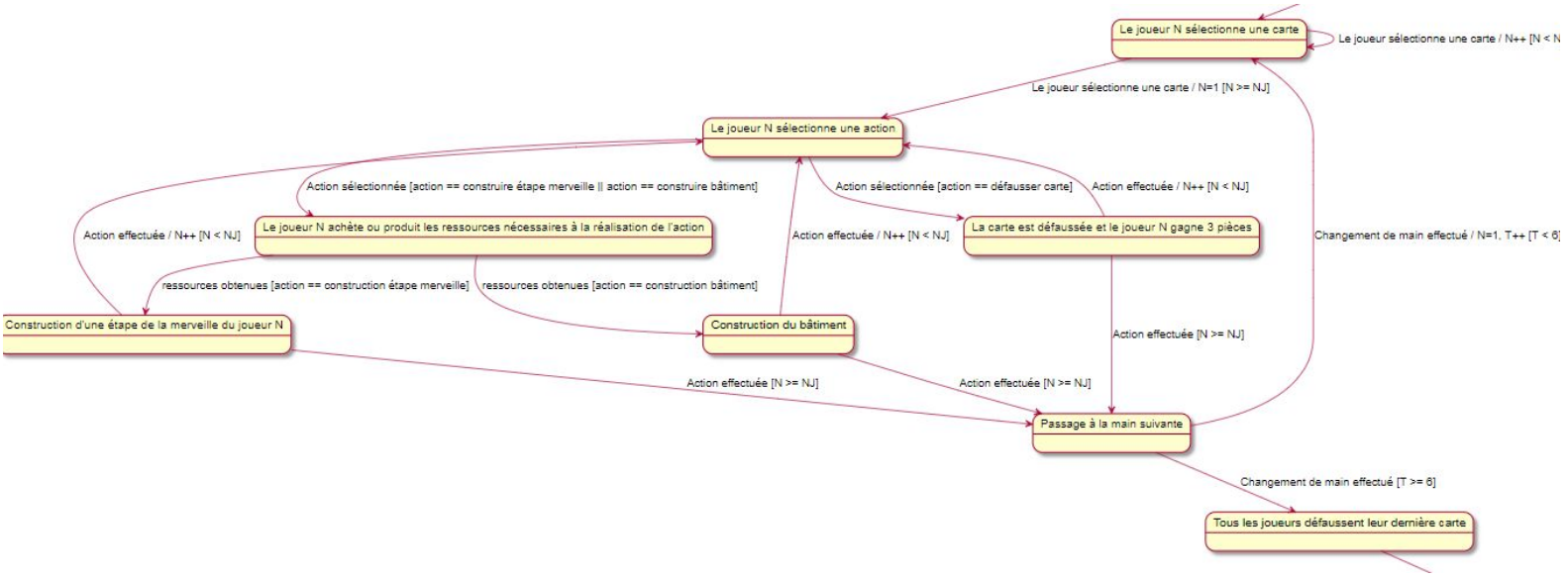


Afin de représenter le cycle des âges, nous mettons en place une boucle qui effectuera 3 itérations. La première étape de cette itération concerne la distribution des cartes. En effet, au commencement de chaque âge, tous les joueurs reçoivent un nouveau deck de 7 cartes.

Suite à cela, nous devons effectuer les 6 tours de jeu qui composent l'âge. Pour ce faire, nous implémentons une nouvelle structure itérative. Ainsi, chaque itération représentera un tour de jeu.

Après avoir effectuées ces 6 itérations, nous devons terminer l'âge en cours. Ainsi, tous les joueurs défaussent leur dernière carte et effectuent la phase de résolution des conflits militaires. Il nous faut ensuite vérifier si les 3 âges ont été effectués. Si tel est le cas, nous terminons la partie. Sinon, nous effectuons une nouvelle itération. Cela correspondant au passage à l'âge suivant.

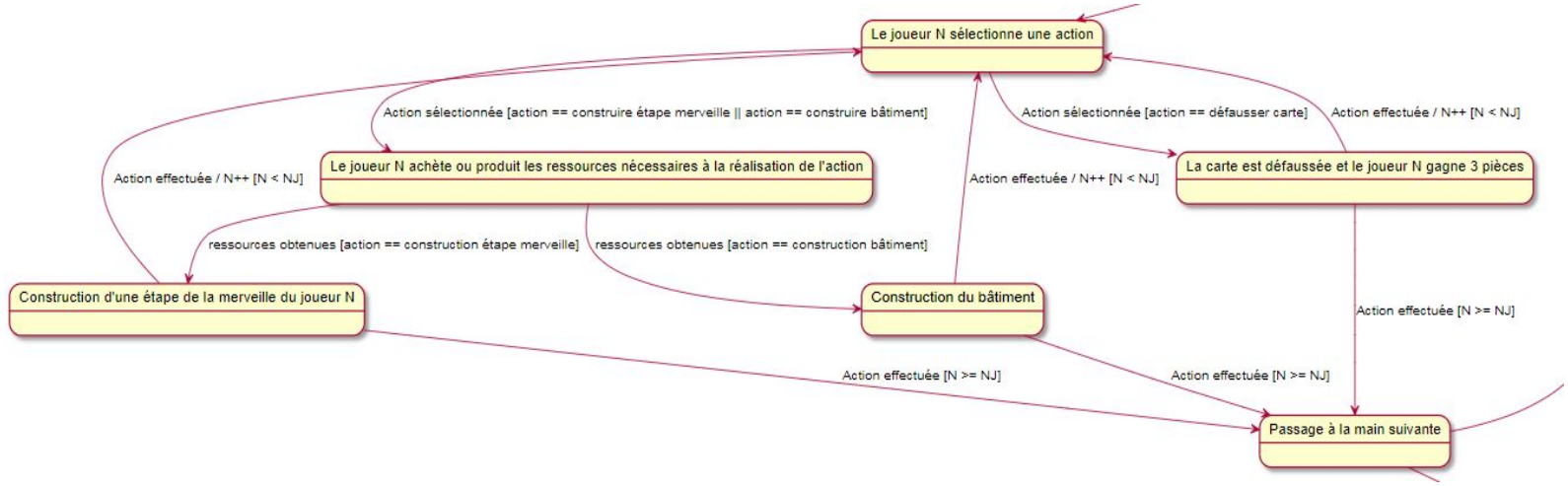
3.3 - Déroulement d'un tour de jeu



Au début de chaque tour, les joueurs devront commencer par sélectionner une carte parmis celle composant leur deck. Après avoir effectué ce choix, ils devront choisir et effectuer une action. Il est important de souligner que les joueurs ne peuvent effectuer qu'une seule action par tour.

A la fin de chaque tour de jeu, nous devons réaliser la phase de passage à la main suivante. Cette étape consiste à interchanger les decks des différents joueurs. En effet, chacun d'entre eux se verra remettre le deck de son voisin. Remarquons tout de même que le sens de rotation qui doit être adopté pour effectuer cette redistribution varie en fonction des âges. Ainsi, pour les âges 1 et 3 nous devons respecter le sens des aiguilles d'une montre alors que pour l'âge 2 nous devrons respecter le sens inverse.

3.4 - Les actions



Nous allons désormais nous intéresser aux différentes actions que peuvent réaliser les joueurs au cours d'un tour.

Les actions possibles sont au nombre de 3 :

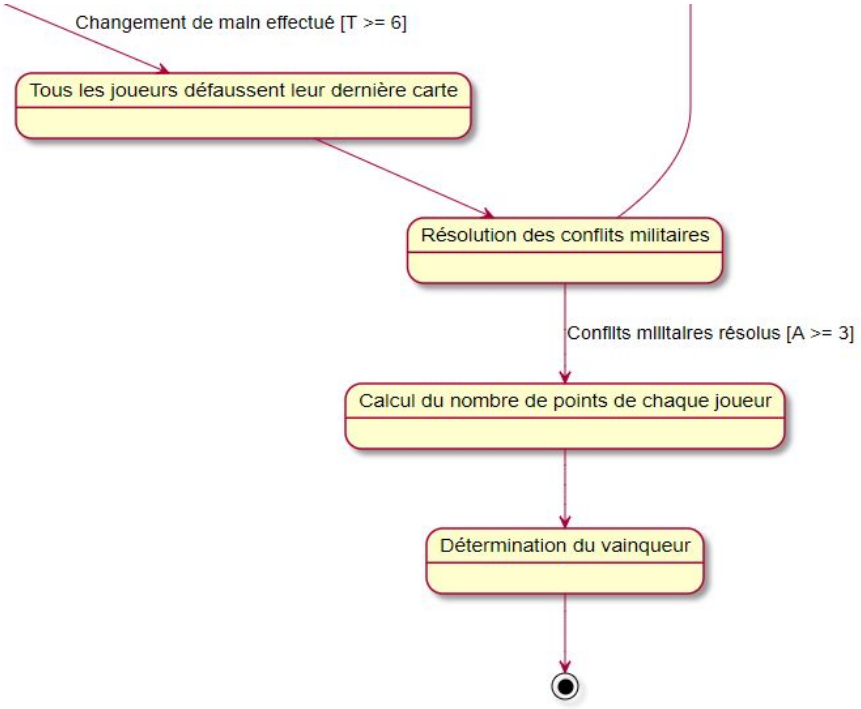
1. Construire le bâtiment correspondant à la carte
2. Construire une étape de la merveille
3. Défausser la carte pour obtenir 3 pièces

Afin de pouvoir effectuer les actions 1 et 2, le joueur doit acheter ou produire un certain nombre de ressources. Ainsi, il est nécessaire de précéder ces actions par une sous-action permettant au joueur de se procurer les ressources nécessaires.

L'action numéro 3 ne nécessite quand-à elle aucune étape intermédiaire. En effet, elle permet au joueur de défausser la carte précédemment sélectionnée et de gagner 3 pièces.

Après la réalisation d'une action, nous vérifions si l'ensemble des joueurs on jouer durant le tour. Si ce n'est pas le cas, un nouveau tour de boucle est effectué afin qu'ils puissent également effectuer leur action. Dans le cas contraire, nous terminons le tour et redistribuons les decks comme vu précédemment.

3.5 - Fin de la partie



Après avoir effectué les 3 âges, il nous faut terminer la partie. Pour ce faire, nous calculons le nombre de points de chaque joueur et déterminons le vainqueur. En cas d'égalité de points, le joueur possédant le plus grand nombre de pièces remporte la victoire. Une égalité sur le nombre de pièces n'est toutefois pas dé-partagée.

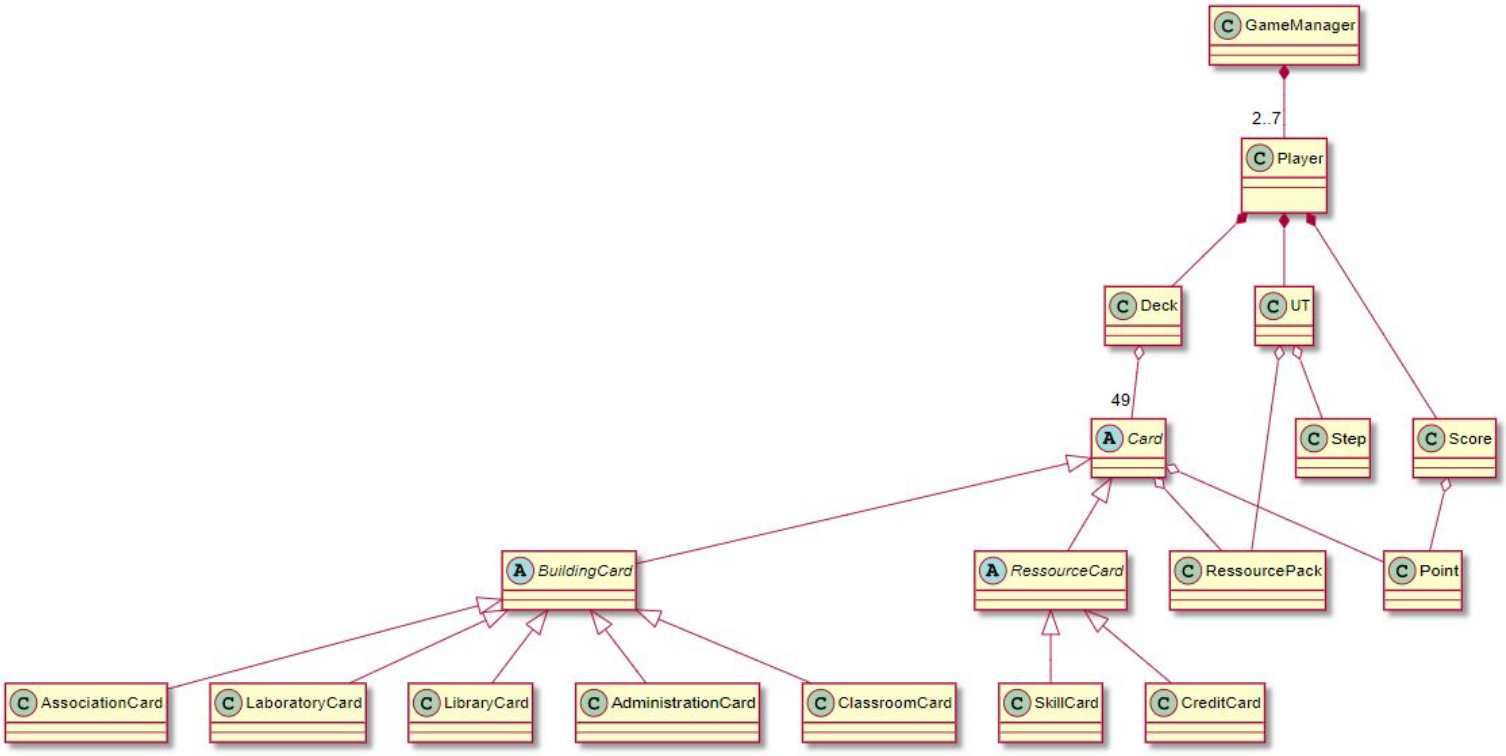
4 - Structure du programme - Diagramme de classes

Ayant dorénavant un schéma détaillé de l'ensemble de notre application, nous pouvons déterminer avec une certaine aisance les différentes classes dont nous aurons besoin. Nous allons procéder en deux, de façon progressive, afin de décrire plus facilement le diagramme de classe.

4.1 - Classes socles

Tout d'abord, focalisons-nous sur les classes socles à implémenter. La logique de notre application nécessite trois besoins se traduisant par une ou plusieurs classes :

1. Gérer la partie dans son ensemble et être en mesure de transférer les données adéquates à la vue,
2. Offrir les outils de gameplay nécessaire au joueur,
3. Définir les éléments du jeu ainsi que leurs règles.



Ce premier élément fait grandement écho au rôle du Contrôleur. Il s'agit effectivement de ce rôle de haut-niveau qui nous intéresse. On nommera ainsi GameManager la classe jouant ce rôle.

Pour l'aspect gameplay, comme l'appui le diagramme de cas d'utilisation, l'ensemble des moyens de gameplay sont à implémenter dans la classe représentant nos joueurs. Il s'agit ainsi sans surprise de la classe Player. Celle-ci compose donc le GameManager puisque c'est ce dernier qui a pour rôle de superviser la partie.

Enfin, les autres classes déployées ne sont que la traduction pure des règles du jeu 7 Wonders. Les noms des classes ont été adapté au monde de l'UTBM. Afin de faire le parallèle avec le jeu original, voici une table d'association des noms :

Nom Original	**Nom Parodié (UTBM)**
Wonder	UT
Raw Material	Credit
Manufactured Product	Skill
Scientific	Laboratory
Civilian	Library
Commercial	Administration
Military	Classroom
Guild	Association

4.2 - Structuration complète

Ayant en notre possession une structure de classe cohérente vis à vis de la structure du programme et des éléments du jeu, il convient de définir plus en détails ces éléments.

Cela passe évidemment par la déclaration des attributs et des méthodes de chaque classe.

Note: Un lien permettant la consultation du diagramme de classe est présent en annexe. Afin de ne pas le rendre illisible, les constructeurs, getters et setters n'ont pas été mentionnés.

Comme nous l'évoquions précédemment, le Contrôleur (GameManager) supervise le déroulé d'une partie et se doit donc de contrôler les actions et interactions du joueur, d'où la présence d'une liste de joueur au sein de ses attributs.

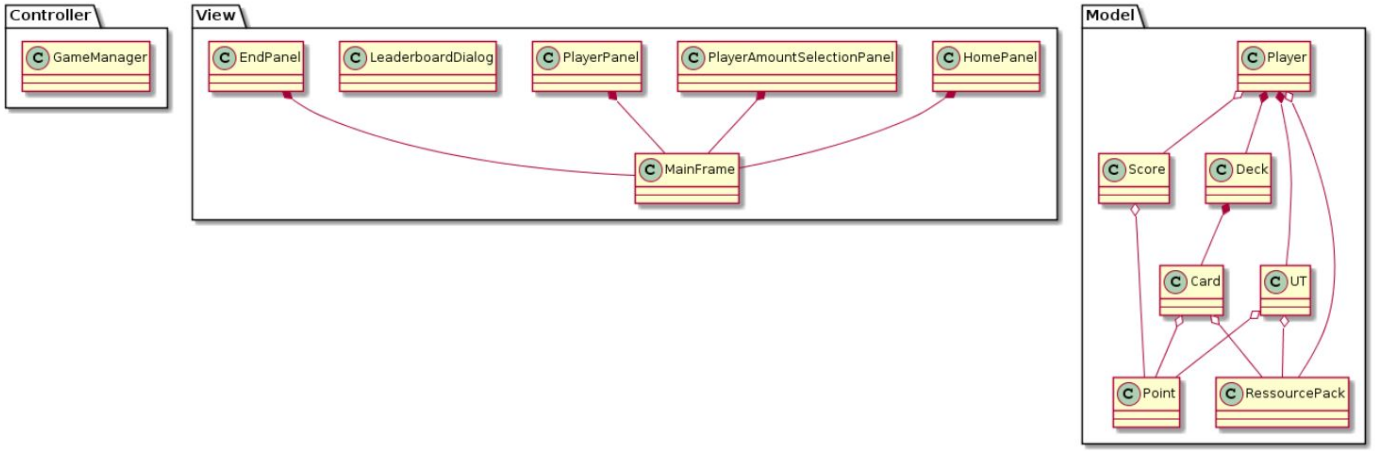
Le joueur représenté par la classe Player, quant à lui, sera composé majoritairement des éléments de gameplay (méthodes) et des outils lui permettant de jouer (attribut), sans compter la gestion du score, évidemment.

Dès lors, les classes suivantes dans la hiérarchie sont les éléments à proprement parler du jeu : UT et Card, grossièrement. Ceux-ci seront enrichi de l'ensemble des données nécessaire à leur définition et leur utilisation par le joueur (effets, ressources produites, cadre d'utilisation, etc.). Afin de différencier et d'ordonner correctement les cartes en fonction des rôles qu'elles peuvent incarner, nous avons choisi de rendre abstraite la classe Card afin de ne rendre instanciable que les cartes ayant un rôle spécifique et bien défini.

Cette dynamique permet plusieurs choses. Tout d'abord, la relation d'héritage permet de profiter de la puissance du polymorphisme. La spécialisation quant à elle assure la cohérence de la diversité des cartes qui seront implémentées en jeu. On évite alors la possibilité de voir apparaître des cartes aux rôles ambigus ou sortant du cadre du jeu.

Enfin, il est important de noter la présence de types énumérés qui ont pour principal but de restreindre les données entrés pour les différents attributs des classes concernés. En plus d'expliciter le rôle des attributs, les types énumérés encadre le champ des possibles pour assurer le respect des règles du jeu et la cohérence de l'univers.

5 - MVC - Organisation du code



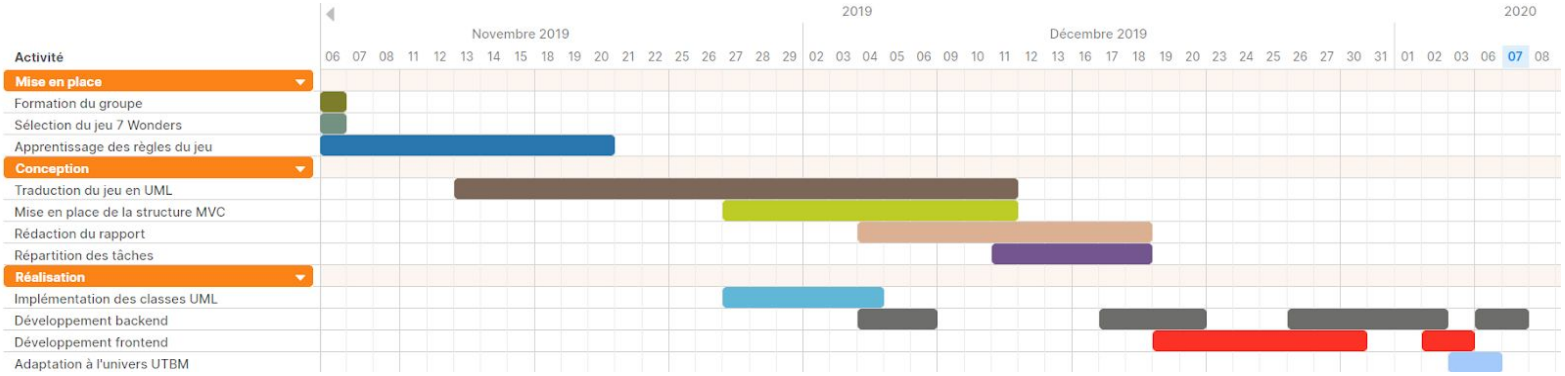
Afin d'accroître la modularité et la maintenabilité de notre projet, nous avons décidé de suivre le modèle d'architecture logicielle MVC. Comme nous pouvons le voir sur le diagramme ci-dessus, l'ensemble des classes constituant le projet sont réparties en 3 catégories : Controller, View et Model.

La catégorie "Controller" contient la logique permettant la gestion des actions de l'utilisateur ainsi que la logique métier du programme. Ainsi, la classe GameManager gérera le déroulement du jeu.

Les classes appartenant au groupe "View" constituent quant-à elles la gestion de l'interface graphique. Nous y retrouvons l'ensemble des classes correspondant aux différentes interfaces présentent au sein de l'application. Nous pouvons par exemple citer le cas de la classe PlayerPanel qui correspond à l'interface de jeu proposée à chaque joueur.

Pour finir, la catégorie "Model" est constituée de l'ensemble des classes permettant la gestion et la récupérations de l'ensemble des données utilisées par le programme. Nous pouvons citer en guise d'exemple les classes Player et Card qui appartiennent à cette catégorie.

6 - Gantt - Organisation du travail



Afin de faciliter la répartition des tâches et d'assurer le bon déroulement du projet, nous nous sommes appuyés sur le diagramme Gantt présenté ci-dessus.

Comme nous pouvons le constater, nous avons dans un premier temps effectué la mise en place du projet en étudiant l'ensemble des règles et spécificités du jeu sélectionné, à savoir 7 Wonders.

Nous sommes ensuite entrée dans la phase de conception. Nous avons, au cours de cette étape, travaillé sur la retranscription du jeu en UML. Cela à conduit à la réalisation de l'ensemble des diagrammes présents dans de ce rapport.

Après avoir conçu, via les diagrammes UML, la majeure partie des classes nécessaires au fonctionnement du programme, nous avons débuté la phase d'implémentation du jeu JAVA. Pour ce faire, nous avons commencé par créer l'ensemble des classes décrites dans les diagrammes UML. Suite à cela, nous avons développé le backend et le frontend afin d'intégrer la logique du jeu et les différentes fonctionnalités proposées à l'utilisateur.

Annexes

A.1 - Diagramme État/Transition au format SVG

http://www.plantuml.com/plantuml/svg/fLPFJoit4B_IfnXnGIf55KAYKjKWqaGz81eXIjK3oi7PdGItNZjux3mXyM6UEB6VOx_OinqJxyExI20kqUxOy_ipCyxwWWpJnjQYkFjf3iV7vo0NiykYS663S335ciaADGW4gUg5HgWG_bCMhORPvG4msYRudv1UtYj5-AyAMR_1x4qI0RMDmD9m9MNRx8Yk4cft1uAL23Ng3N8XCF1r6OhkInz_mXr6XLzWTdG4TpFutNcQvvWL9wFvmXgSC5tH_h3ZqOc7FHzvt5JK2EUJV6L-XP9fW-IVnM7xl48OG_ILMM4FzfN8Mtyhf7Zp7iIk1Nj4_JTwLJxgr4uyVCfk6oJlUNJoCzoEJlRhGP3eHMG4HSOu82FXzw9dVVWXxZX0x8dSRSL-bbbu2PvprY8FUNqnQOfaj7gCnkFoTvEFSCUsuT48ghPPCak4UgjtFeIk5ePnUOEah2wH_bGwBkv1KYhfH5ck4Th6i7LtL9-Ue7VZedqsl4PfEYLZiQP8PYVSKw41qIddYgqamYciUVlYfFeoviKfiiXma-C3BfSkweOucvTUB7-Av0gdMFxVY_7-JB3oldrnGMLXhLLbkVT8e95Y1y1LkNHlZBkOFxI2WMuR9ZYnG1NET9gamGuM-K8sLV-iLvfLwSyjoV2mA2rIJyF3UzRoEUp3-8VbeWe_NAu-njWtEvFkUdLevMuKVPbOPo0hksk2BrjV0OgUsAUd9qVxsFna9csgGpIIrRNeYNtQI8RsSIFNZ8YjC4vrpRW4idp3PFmG9C0YFGv-Wy8ENmQeC80byjt7oDkuTJREvls850ytLbUCza7sSCxLMJ9rgoo5Cnpl8-c_r-KB5t4WMlBsHMVVl3vwA0QNl6oDfCG6_-A26wOVtMWi1IyDxUJSj8tRyNeL82Z33aDGXnIkup_d8D3Gmi35PPBZWzvg87zXyHqOkmwSpjz7Zgwl5PU69aoKLh1WudqA3-xwSPhwiSs5odRkb-k0wf-oAz0WSJ6q4EH-uLAYdg91NNCPukawrZQTi5UoOLm-x1ZxShDOv76NwE82PMLhyGq0

A.2 - Diagramme de Classes au format SVG

http://www.plantuml.com/plantuml/svg/b5NTZl8s4BtdAIBrXcgxBu0gLDZkLYjH5IrXxvru0XQEdVg7DgtwxfrnJEmGLlg-6vGvFZEUdpFc5-kOSRwL2mkkKAm5sx4Q3j0nmvmsnMgrMDIIMLIyXeU_CyLEO8f_5pyKKbYtbwm7OujLiJ66zJk4VXwmDJAGYYTlshJC2QqG0SKf3AALrX8OOP9rQ0lb5i_FYwSY9BJy4Q6h5Xo13kELtgWSynrd3dR0E9XACyFnCC_XK6i3QoJYXRbVVORwyjx4j3-P51G76MD2IE8OmOZAEtZ1o7RvA_ga61HoSu8yQWEkFeShxSCyHihPDuq4VYplQCUIE489bqUEaC0i_u6OSdweu6-tOrs2_hkDYiXnIYTizvOwXAHZYO2bbD08gQ7DiJeqgKYqQgeukFC4ifh6Tm09jGlu9W2Hdk7Khpkqr5lVVu2rsfiQR289FcJvDH4bGRnnJ1FLT4PpJtwPIzxl4Twp-h8UfCNrSYXbB27hfWrvK79vZq_Y2jI5KY-xeQchsDsCLFa--D7FITXIUgKcnqISl4uF-TpFVZ-lyh4ySjIP_IMgvHMEvKGRh49TOmEAGIIKH928u6EprR2S1sU4Eg7j-WxYo4hyfDbgwn2Pj1VXvhRaTmjVQw5UcL517diqN7mvOcNVD3rqQR7KrhT3wp_WJozCa0oCMeqBiccrL-wcv2tK6brFRmPWqpWmn4AxF0iRwiyl9M0TFUTNR05hX7VrX-5WXbdUec31T47ITiUCKvZCMNIKMR9GXsbB9qNI96Axqm52LoOznFaCpj6RL3c-m-6XcXSJNzJXsSabmZX_fDX6I0TcsuyJN8vVnA8hCzQauyk9bR4DjFeAwRQqSBuYoX8h-waGoGTh78iybeyLYWSlmlLe0ohTH-6zthuHZwlndJAp3hefvO18x2FGKc4Rio7GQSYGT6wdGa_BykM5OQ2pnHn9SKE9TS_PUY4v2ZcdnBKsm8KZV08VBaBA6RfZbGx_y_qyq4vKXZqws73QHag7UlqW0_erMhSFNB6oMipyucbeMxu80S27Pxu8oIdyZmrEsZ7vAMegfoDFcus2fPlp0O5oHZJ-13ceHSvfX69ok2M15wN_ai0dVXtMt8394TzsmKn9W_9jcZ3c_LJyJrKLBBSEmi1lax5iwmKaHc2_ffUSCjEUUclx-MLnIzE

BTaEyAw4155O44mMK5ZWWQ7NEpigxbqroMyfk_xWsKYgjtB2cOJakwcbfAPa6StmrEMdO8CiSXA8yKnNJ8VrpSxkyVhmXlhH7hU7IlWMqKGYJrq_4-_iTWok9IjeJ63s13TVpXdQxYUDYNi-wil2g385fRVy8-Gn3qYgq6YWVhAG5T2DTDn9KutRbCkpFUNdQxnBdZjSM-R1um7-M4k3zhHX3x6oGyoozbHU9iM6KoSclKE0NJUPKpU58CtPPglUsGHof53V3jXkp83yDxOd6CzGpgVNTBkAbv-DWYXHrauVYnac7vj4KwN0KPDbN_Zqd5xN_aUU-bDriOGVRKsvmIEYb3vQRgDuo--shxYh6GHY6WRkVaR5ICo0aX5GX6DYHQFa0xFq0ZySvZjWMzLJ5lZZnfJhRCHTakuMoypD8vo05WE7CQmHd23n8xiQxPSxbSP-ljqQk7rg6SIoxMW-5VxpW0tjhZTqAPGW5TjKz-sFv5dOHYDRqLaWGQRC7OPO0Wf_od7XDfSNYq6TwlEsbagU0gDePY2gP8gAuYpZME548FequwXx8mJfKFpCJjip_81fHP-DZrdhw7HZxgC77i4EwFfefTenH1QTQ_S2hzq1WjXrCkhpMqkvnULfWHNMbuIxNbo1LsAoqdNMWr07y0m00