Documentation ATP

Mathieu Bourgais

September 2019

The English version of this documentation can be found below the French version.

1 Modèle médiéval d'échange des pierres de construction

Cette section a pour but de résumer en quelques page le comportement général du modèle d'échange de pierres de constructions au Moyen Age en Haute Normandie. Cette section sert aussi de documentation pour le code associé (stoneModel.gaml), disponible à l'adresse suivante : https ://github.com/IDEES-Rouen/ATP/models

Le modèle se compose d'agents consommateurs (chateaux/églises) et producteurs (carrière) actifs dans un environnement partagé qui effectue lui-aussi des actions à chaque pas de temps, mettant à jour les informations de la simulation. Le comportement de ces trois entités actives est découpé en différentes hypothèses de complexités, sur le modèle de la modélisation incrémentale [1], comme le montre la figure 1. La modélisation incrémentale permet de mesurer les effets d'une montée en complexité du comportement d'un type d'acteurs donné. Ainsi, un sous-modèle particulier est une compositions des hypothèses de chaque composants. Par exemple, le sous-modèle H1.2.3 est composé du niveau 1 de complexité de l'environnement, du niveau 2 de complexité des consommateurs et du niveau 3 de complexité des producteurs. Chaque fois que l'on augmente de complexité sur un axe, cela signifie que l'on considère tous les niveaux précédents comme actifs.

La complexité de l'environnement se découpe en 5 niveaux :

— H0 : Environnement sans distance particulière, localisation aléatoire des agents.
— H1 : Une distance est appliquée sur l'environnement, contraignant les échanges entre producteurs et consommateurs.
— H2 : L'environnement se caractérise par une différenciation des sols ainsi que de l'altitude, modifiant la notion de distance ainsi que la localisation des agents.
— H3 : L'environnement se dote de frontières politiques, contraignant les échanges.

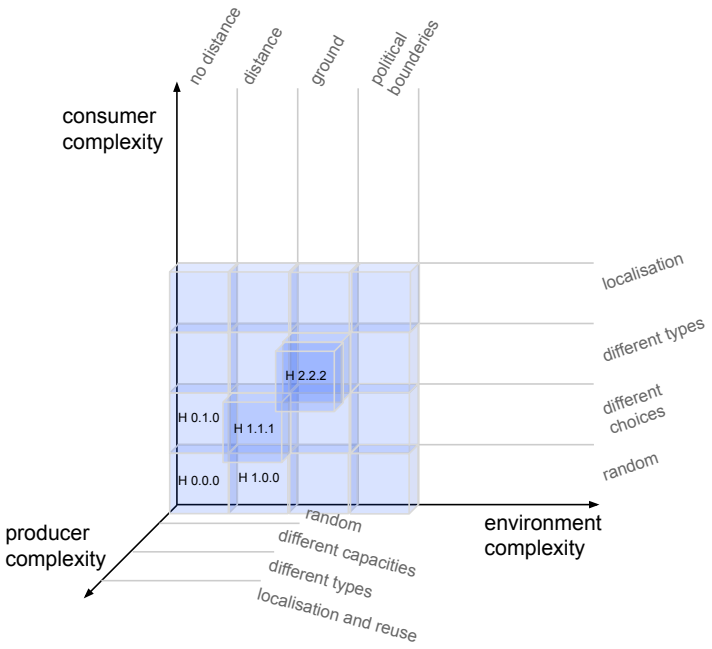

FIGURE 1 – Modélisation incrémentale de l'échange de pierres de construction au Moyen-Age

— H4 : Environnement réel (taille, forme, sol, rivières, frontières politiques).

La complexité des consommateurs se découpe en 4 niveaux :

— H0 : Les consommateurs n'ont besoin que d'un seul type de pierre de construction.
— H1 : Les consommateurs peuvent reprendre leur construction (après avoir déjà obtenus une quantité initiale de pierres dont il avait besoin pour terminer la construction une première fois)
— H2 : Une différenciation est faite entre les consommateurs sur leur prestige (soit ils sont prestigieux, soit ils ne le sont pas). Parmi les consommateurs prestigieux, on distingue ceux prioritaires de ceux qui ne le sont pas. Les consommateurs ont aussi des besoins dans 2 types de pierre différents : de la pierre de taille et du blocage.
— H3 : Les consommateurs sont placés initialement selon des règles expertes (cura et al.).

La complexité des producteurs se découpe en 4 niveaux :

— H0 : 1 seul type de pierre est produit, de façon infinie.
— H1 : La production est finie : chaque producteur ne peut produire qu'une certaine quantité de pierre par pas de temps, sur une ressource finie qui

s'amenuise à chaque tour en fonction de la production du tour précédent.
- H2 : Différenciation du type produit : Le type 1 (la pierre de taille) est produit sans limite de ressource (mais avec une limite par pas de temps), le type 2 (la pierre de blocage) est produit en limité. Les producteurs de type 1 sont connus au début de la simulation, les producteurs de type 2 sont les mêmes que les producteurs de type 1, auxquels on ajoute des créations de producteur initiées par les consommateurs (voir achat de type 2 pour plus de détails).
- H3 : Les consommateurs deviennent des producteurs par le mécanisme de la réutilisation.

Le comportement global de la simulation est résumé par la figure 2. Les comportement d'initialisation et de mise à jour sont réalisés par l'environnement (en vert), le comportement d'achat est réalisé par les consommateurs (en bleu) et le comportement de production est réalisé par les producteurs (en rouge). Ce comportement global évolue en fonction du niveau d'hypothèse de chaque composant du modèle, symbolisé par les blocs plus petits et de couleur plus foncé (vert lorsque c'est un niveau de complexité de l'environnement, bleu lorsque c'est un niveau de complexité des consommateurs et rouge pour les niveaux de complexités.

Ci-dessous, les détails de chaque bloc de comportement. Dans la suite du document, le niveau de complexité de l'environnement est noté He, celui des consommateurs est noté Hc et celui des producteurs est noté Hp.

1.1 Initialisation

L'initialisation du modèle est réalisée par l'environnement au lancement de la simulation et se compose de deux processus : la localisation des agents créés et l'initialisation des valeurs de ces agents. Ces deux processus sont influencés par la complexité de l'environnement, la complexité des consommateurs et la complexité des producteurs. L'algorithme 1.1 montre le fonctionnement global de ce bloc d'initialisation : on lance l'initialisation des valeurs puis, en fonction de la complexité des consommateurs, on crée le nombre requis de consommateurs prestigieux, parmi lesquels on crée le nombre requis de consommateur prioritaire, puis on crée les consommateurs non prestigieux avant de terminer par la création des producteurs.

La fonction *initialiserValeur*, décrite par l'algorithme 1.2 permet de donner une valeur initial à certaines valeurs du modèle qui dépendent de la complexité des consommateurs, des producteurs et de l'environnement.

La fonction *creationConsommateur*, décrite par l'algorithme 1.3 permet de créer des consommateurs en fonction de la complexité des consommateurs et de l'environnement, ainsi qu'un indiquant si le consommateur créé est prestigieux ou non. La fonction *calculDistance* permet de calculer initialement les valeurs de probabilité et de pourcentage utilisée lors de l'achat. Ainsi, une grande partie de ces valeurs est calculée au lancement, permettant d'alléger le calcul de chaque pas de temps par la suite.

La création de producteur décrite par l'algorithme 1.4 est réalisée en fonction

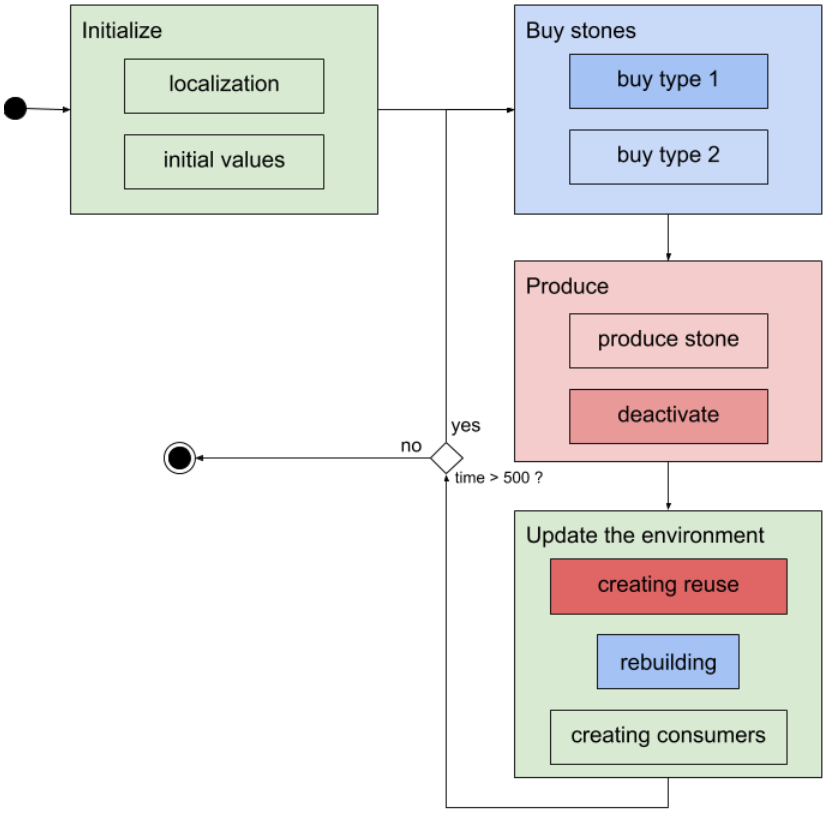

FIGURE 2 – Diagramme représentant les grandes composantes du comportement global du modèle d'échange de pierres de construction au Moyen-Age

de la complexité de l'environnement et des producteurs, ainsi que du type de pierre devant être produit.

1.2 Achat de pierre

Le comportement principal des consommateurs, tant qu'ils n'ont pas récupéré autant de pierres que leur besoin, est d'acheter de la pierre. Ce processus est présenté de façon globale par l'algorithme 1.5 dans lequel on observe que l'achat par défaut (lorsque les consommateurs n'ont besoin que d'un type de pierre) correspond à l'achat de pierres de type 2 (pierres de blocages).

L'algorithme 1.6 explique le fonctionnement de l'achat de la pierre de type 1, qui est aussi expliqué par le diagramme 3. Dans cet algorithme, la liste des producteurs à visité (variable nommée producteurAVisite) est différente en fonction du degré de complexité des producteurs. Successivement, elle va contenir les pro-

```
début Initialisation Globale {
    initialiserValeurs(He,Hc,Hp);
    si(Hc>1){
        repeter nbConsoPrestige/an{
            creationConsommateurs(He,Hc,vrai);
        }
        changer priorite chez (nbPrioritaire/an parmi Consommateur);
    }
    repeter nbConsoNonPrestige/an{
            creationConsommateurs(He,Hc,faux);
        }
    creationProducteurs(He,Hp,type1);
}
fin
```

Code 1.1: Déroulement de l'initialisation globale du modèle d'échange de pierre de construction

```
début initialiserValeur(He,Hc,Hp) {
    si(Hc<1){
        probaReConstruction = 0.0;
    }
    si(Hp<3){
        probaReutilsation = 0.0;
    }
    si(He>1){
        différenciationSol();
    }
    si(He>2){
        fontierePolitique();
    }
    si(He>3){
        chargerCarteReelle();
    }
}
fin
```

Code 1.2: Détail de la fonction initialiserValeur

ducteurs de type 1 ou les producteurs de type 1 additionné des consommateurs offrant de la réutilisation.

Le pourcentage utilisé dans l'algorithme (variable nommée pourcentageDistance) dépend de la distance entre le consommateur et le producteur en jeu à cet instant, selon la fonction décrite par le graphique 4. Dans le cadre de la pierre de type 1, la valeur d1 est égale à la distance avec le producteur de type 1

```
début creationConsommateurs(He,Hc,Prestige) {
    si(Hc<3){
        localisationAleatoire();
    } sinon {
        si(He>1){
            localisationSelonSol();
        }
        si(He>2){
            localisationSelonFrontieresPolitiques();
        }
        LocalisationRelativeAuxConsommateurs();
    }
    si(Prestige){
        estPrestigieux = vrai;
    } sinon {
        estPrestigieux = faux;
    }
    calculsDistance();
}
fin
```

Code 1.3: Détail de la fonction creationConsommateurs

le plus proche, la distance d2 étant une distance maximum liée au constructeur, qui est à déterminer par l'étude.

Cette procédure d'achat du type 1 de pierre peut se résumer par l'équation suivante : $r = \sum_{i \in V} min(b - s, f(i) \times p_i)$ avec V l'ensemble des vendeurs (carrières + autres chantier en cas de réemploi) trié du plus près au plus loin, b le besoin du consommateur, s la quantité de pierre stockée (précédemment achetée), p_i la quantité maximum pouvant encore être produite par le vendeur i et $f(i)$ la valeur de la fonction d'influence pour le vendeur i. Cela signifie que seule une portion de ce qui est rendu disponible par le vendeur est récupéré, et que cette proportion est plus petites pour les vendeurs plus éloignés.

L'algorithme 1.8 explique le fonctionnement de l'achat de pierre de type 2, tout comme le diagramme 5. L'algorithme 1.7 explique comment de nouveaux producteur de type 2 sont ajoutés si aucun producteur encore actif ne se trouve à une distance inférieur à une distance minimale, qui correspond dans ce cas à la valeur d2 sur le graphique présent en figure 4.

Comme pour l'achat de pierres de type 1, la liste des producteurs à visité dépend du niveau de complexité des producteurs. Le fonctionnement de la distance diffère de l'achat de pierres de type 1 : la fonction distance décrie par la figure 4 agit dans ce cas comme une probabilité de commander de la pierre à un producteur, plus ce producteur est éloigné et moins il y a de chance qu'il soit choisi. Sur ce graphique, la valeur d2 correspond à une distance minimale fixée qui est une variable interne de chaque consommateur, la variable d1 correspond

```
début creationProducteurs(He,Hp,type) {
    si(Hp==0){
        ressource = infini;
        stockMax = ressource;
    }
    si(Hp==1){
        ressource = infini;
        stockMax = parametreStockMax;
    }
    si(Hp>1){
        si(type=type1){
            ressource = infini;
            stockMax = parametreStockMax1;
        } sinon {
            ressource = parametreRessource;
            stockMax = parametreStockMax2;
        }
    }
    localisationAleatoire();
    si(He>1){
        localisationSelonSol();
    }
    si(He>2){
        localisationSelonFrontieresPolitiques();
    }
    si(He>3 et type==type1){
        localisationSelonCarte();
    }
    si(type==type1){
        creationProducteurs(He,Hp,type2);
    }
}
fin
```

Code 1.4: Détail de la fonction creationProducteurs

à une valeur maximale fixée à l'avance, variable de chaque consommateur. Une fois un producteur sélectionné, une quantité aléatoire du maximum récupérable possible est pris par le consommateur. Cette mécanique propose de reproduire la notion de négociation, très probablement à l'oeuvre de façon importante sur l'achat de pierre de blocage, moins prestigieuse que la pierre de taille. La présence d'une distance minimale (d2) permet d'avoir des producteurs que l'on sélectionne à coup sûr (s'ils sont à une distance inférieur à d2).

L'équation suivante représente ce processus : $r = \sum_{i \in V} alea(min(b - s, p_i))$ avec avec V l'ensemble des vendeurs (carrières + autres chantier en cas de ré-

```
début achatPierre{
    tant_que(non estConstruit){
        si(Hc>1){
            achatType1();
        }
    }
    si(He>0){
        activerProducteur();
    }
    achatType2();
}
fin
```

Code 1.5: présentation globale de l'achat de pierre

```
début achatType1{
    tant_que(non estVide(producteurAVisite) et non estConstruit){
        si(He > 0){
            trierParDistance(producteurAVisite);
            producteurTemp = producteurAVisite[0];
            collectType1 = collectType1 +
                min(besoinType1 - collectType1,
                (producteurTemp.productionType1
                x pourcentageDistance));
        } sinon {
            aleatoire(producteurAVisite)
            producteurTemp = producteurAVisite[0];
            collectType1 = collectType1 +
                min(besoinType1 - collectType1,
                producteurTemp.productionType1);
        }
        producteurAVisite = producteurAVisite - producteurTemp;
    }
}
fin
```

Code 1.6: présentation de l'achat de pierre de type 1 (pierre de taille)

emploi) trié du plus près au plus loin, en ayant à chaque pas de temps une probabilité $f(i)$ de sélectionner le vendeur i, avec b le besoin du consommateur, s la quantité de pierre déjà récupérée, *alea* une fonction aléatoire et p la production d'une producteur.

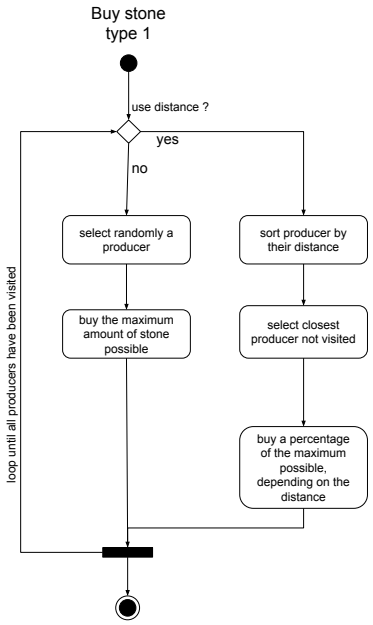

FIGURE 3 – Diagramme d'activité de l'achat de pierre de type 1 (pierre de taille)

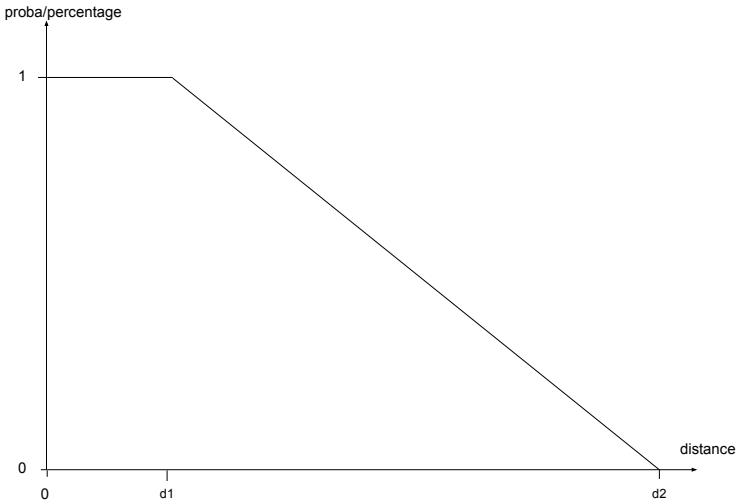

FIGURE 4 – fonction indiquant le calcul de la puissance de l'aire d'influence

1.2.1 Production

Les producteurs n'ont qu'un seul comportement actif qui consiste à produire de la pierre en onction de ce qui leur a été commandé. Leur second comporte-

```
début activerProducteur{
    trierParDistance(producteurAVisite);
    producteurTemp = producteurAVisite[0];
    si(distance(self, producteurTemp)>distanceMin){
        creationProducteurs(He,Hp,type2);
    }
}
fin
```

Code 1.7: présentation de la création de nouvelles carrières de type 2

```
début achatType2{
    tant_que(non estVide(producteurAVisite) et non estConstruit){
        si(He > 0){
            trierParDistance(producteurAVisite);
            producteurTemp = producteurAVisite[0];
            si(aleatoireDistance(producteurTemp)){
                collectTemp = min(besoinType2 - collectType2,
                    (producteurTemp.productionType2);
                collectType2 = collectType2 +
                    aléatoire x collectTemp;
            }
        } sinon {
            aleatoire(producteurAVisite)
            producteurTemp = producteurAVisite[0];
            collectType2 = collectType2 +
                min(besoinType2 - collectType2,
                producteurTemp.productionType2);
        }
        producteurAVisite = producteurAVisite - producteurTemp;
    }
}
fin
```

Code 1.8: présentation de l'achat de pierre de type 2 (pierre de blocage

ment, pour les producteurs de type 2 (à partir de Hp>0), consiste à diminuer la quantité de ressource initiale sur laquelle ils peuvent fournirent les consommateurs.

La quantité de pierre produite par chaque producteur à chaque pas de temps, symbolisé dans l'algorithme 1.9 par la variable *pierreCommandee*, est définie lors de la phase d'achat des consommateurs. En fonction du niveau de complexité des producteur, cette variable va être bornée ou non par une quantité de production maximale par pas de temps.

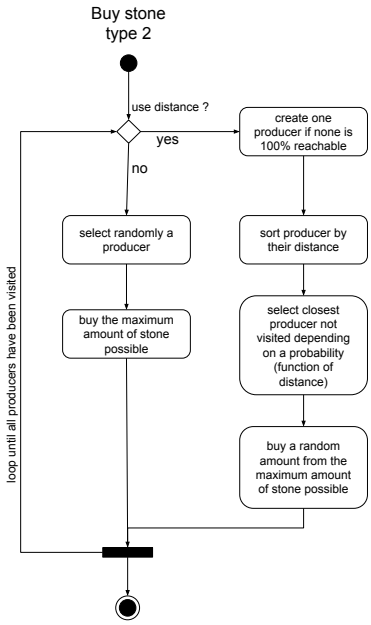

FIGURE 5 – Diagramme d'activité de l'achat de pierre de type 2 (pierre de blocage)

```
début production{
    pierreProduite = pierreCommandee;
    pierreCommandee = 0;
    si(Hp>1 et type==type2){
        ressource = ressource - pierreProduite;
    }
}
fin
```

Code 1.9: présentation de la production de pierre

1.2.2 Mise à jour

La mise à jour de la simulation est une opération réalisée par l'environnement à chaque tour ayant pour but, en fonction de la complexité des consommateurs et des producteurs, de reconstruire les consommateurs déjà construits, de mettre en place le réemploi et de créer de nouveaux consommateurs. Ce processus global est expliqué par l'algorithme 1.10 ainsi que le diagramme en figure 7.

La fonction de reconstruction ajoute un besoin pré-défini aux consommateurs ayant déjà satisfait leur besoin, en fonction d'une probabilité pré-définie. La fonction de réemploi sélectionne une quantité pré-définie de consommateur

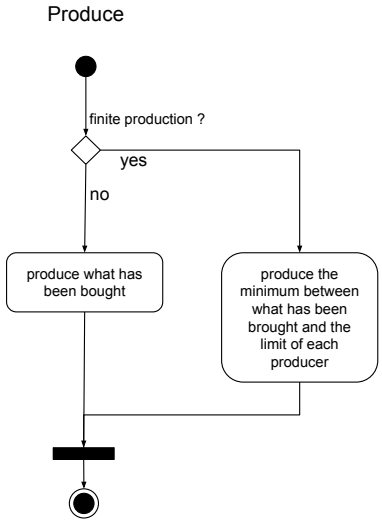

FIGURE 6 – Diagramme d'activité de la production de pierre

et place 10% de ce que le consommateur à déjà récolté comme pierre sur la liste des matériaux pouvant être réemployé par d'autres consommateurs. Enfin, la procédure de création de nouveau consommateurs ajoute de nouveaux consommateurs prestigieux ou non, prioritaires ou non, en fonction des données pré-définies.

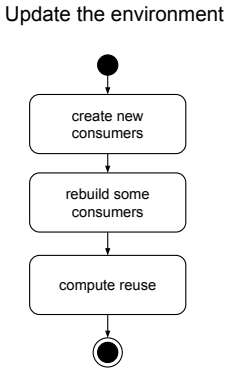

FIGURE 7 – Diagramme d'activité de la mise à jour de l'environnement

```
début miseAJour{
    si(Hc>0){
        reconstruction();
    }
    si(Hp=3){
        reemploi();
    }
    repeter nbConsoPrestige/an{
            creationConsommateurs(He,Hc,vrai);
    }
    changer priorite chez (nbPrioritaire/an parmi Consommateur);
    repeter nbConsoNonPrestige/an{
            creationConsommateurs(He,Hc,faux);
    }
}
fin
```

Code 1.10: présentation de la mise à jour de l'environnement

2 Medieval model of building stone exchange

English Version of the documentation.

The purpose of this section is to summarize in a few pages the general behaviour of the stone exchange model in the Middle Ages in Upper Normandy. This section also serves as documentation for the associated code (stoneModel.gaml), available at the following address : https ://github.com/IDEES-Rouen/ATP/models

The model consists of consumer agents (castles/churches) and producers (quarry) active in a shared environment that also performs actions at each time step, updating the simulation information. The behaviour of these three active entities is broken down into different assumptions of complexity, based on the incremental modelling model [1], as shown in figure 1. Incremental modelling makes it possible to measure the effects of an increase in complexity of the behaviour of a given type of actor. Thus, a particular sub-model is a composition of the assumptions of each component. For example, the H1.2.3 sub-model is composed of environmental complexity level 1, consumer complexity level 2 and producer complexity level 3. Each time we increase complexity on an axis, it means that we consider all the previous levels as active.

The enviroment's complexity is broken down into 5 hypotheses :

— H0 : Environnement with no particular distance, random agent's location.
— H1 : A distance is applied to the environnement, constraining actions between producers and consumers.
— H2 : The environnement is caracterized by a ground differenciation and an elevation, changing the distance and the location of agents.
— H3 : Political boundaries are applied to the environment, constraining

the actions.
— H4 : Real environnement (size, shape, ground, rivers, political boundaries).

The consumer's complexity is broken down into 4 hypotheses :
— H0 : Consumers only need one type of stone.
— H1 : Consumers can be rebuilt.
— H2 : Different types of consumers are introduces : prestigeous one and not prestigeous one, priority one and not prioority one. Consumers also need 2 types of stone.
— H3 : Consumers are located according realistic rules (cura et al.).

The producer's complexity is broken down into 4 hypotheses :
— H0 : 1 type of stone is infinitly produced
— H1 : Production is finite ; each producer produce a given quantity of stone at each time step, on a diminishig ressource.
— H2 : Different type of production ; Type 1 is produced without limit on the ressource, type 2 is limited in production. Différenciation du type produit : Le type 1 (la pierre de taille) est produit sans limite de ressource (mais avec une limite par pas de temps), le type 2 (la pierre de blocage) est produit en limité. Type 1 producers are known at the begining, and paired with a type 2 producer. Type 2 producers are also added during the simulation (see buy type 2 below).
— H3 : Consumers may become producers with the reuse process.

The overall behaviour of the simulation is summarized in the figure 2. The initialization and update behaviors are performed by the environment (in green), the purchase behavior is performed by consumers (in blue) and the production behavior is performed by producers (in red). This overall behaviour evolves according to the assumption level of each component of the model, symbolized by the smaller and darker coloured blocks (green when it is a level of complexity of the environment, blue when it is a level of complexity of consumers and red for the levels of complexity.

Below, the details of each behaviour block. In the rest of the document, the level of complexity of the environment is noted He, that of consumers is noted Hc and that of producers is noted Hp.

2.0.1 Initialization

The initialization of the model is performed by the environment at the start of the simulation and consists of two processes : the location of the agents created and the initialization of the values of these agents. Both processes are influenced by the complexity of the environment, the complexity of consumers and the complexity of producers. The algorithm 2.1 shows the global functioning of this initialization block : the initialization of values is launched and then, depending on the complexity of the consumers, the required number of prestigious consumers is created, among which the required number of priority consumers is created, then the non-prestigious consumers are created before finally creating the producers.

```
start global Initialization {
    initializeValue(He,Hc,Hp);
    if(Hc>1){
        repeat nbConsumPrestige/year{
            creationConsumers(He,Hc,true);
        }
        change priority in (nbPrioritaries/year among Consumers);
    }
    repeat nbConsumNotPrestige/year{
            creationConsumers(He,Hc,false);
        }
    creationProducer(He,Hp,type1);
}
end
```

Code 2.1: Global initialization of the trade construction stone model

The function *initializeValue*, described by the algorithm 2.2, allows to give an initial value to certain values of the model that depend on the complexity of consumers, producers and the environment.

```
start initializeValue(He,Hc,Hp) {
    if(Hc<1){
        probaRebuild = 0.0;
    }
    if(Hp<3){
        probaReuse = 0.0;
    }
    if(He>1){
        groundDifferenciation();
    }
    if(He>2){
        politicalBoundaries();
    }
    if(He>3){
        loadRealMap();
    }
}
end
```

Code 2.2: Details for the initializeValue function

The function *creationConsumer*, described by the algorithm 2.3 allows to create consumers according to the complexity of consumers and the environment, as well as an indicating if the consumer created is prestigious or not. The function *calculateDistance* allows you to initially calculate the probability and

percentage values used when purchasing. Thus, a large part of these values are calculated at launch, making it possible to reduce the calculation of each time step afterwards.

```
start creationConsumer(He,Hc,Prestige) {
    if(Hc<3){
        randomLocation();
    } else {
        if(He>1){
            groundLocation();
        }
        if(He>2){
            politicalLocation();
        }
        relativeLocation();
    }
    if(Prestige){
        isPrestigeous = vrai;
    } else {
        isPrestigeous = faux;
    }
    calculateDistance();
}
end
```

Code 2.3: Details for the creationConsumer function

The creation of producers described by the algorithm 2.4 is carried out according to the complexity of the environment and the producers, as well as the type of stone to be produced.

2.0.2 Buying stone

The main behaviour of consumers, as long as they have not recovered as many stones as they need, is to buy stone. This process is presented in a global way by the algorithm 2.5 in which we observe that the default purchase (when consumers only need one type of stone) corresponds to the purchase of type 2 stones (blocking stones).

The algorithm 2.6 explains how to buy type 1 stone, which is also explained by the diagram 3. In this algorithm, the list of producers visited (variable called producerToVisit) is different depending on the degree of complexity of the producers. Subsequently, it will contain type 1 producers or type 1 producers plus consumers offering reuse.

The percentage used in the algorithm (variable called percentageDistance) depends on the distance between the consumer and the producer at stake at that time, depending on the function described in the graph 4. In the case of

```
start creationProducer(He,Hp,type) {
    si(Hp==0){
        ressource = infinity;
        maxStock = ressource;
    }
    if(Hp==1){
        ressource = infinity;
        maxStock = parameterStockMax;
    }
    if(Hp>1){
        if(type==type1){
            ressource = infinity;
            maxStock = parameterStockMax1;
        } sinon {
            ressource = parameterRessource;
            maxStock = parameterStockMax2;
        }
    }
    randomLocation();
    if(He>1){
        groundLocation();
    }
    if(He>2){
        politicalLocation();
    }
    if(He>3 et type==type1){
        mapLocation();
    }
    if(type==type1){
        creationProducer(He,Hp,type2);
    }
}
end
```

Code 2.4: Details for the creationProducer function

type 1 stone, the value d1 is equal to the distance to the nearest type 1 producer, the distance d2 being a maximum distance related to the manufacturer, which is to be determined by the study.

This procedure for purchasing type 1 stone can be summarized by the following equation : $r = \sum_{i \in V} min(b - s, f(i) \times p_i)$ with V all sellers (quarries + other sites in case of reuse) sorted from closest to farest, b the consumer's need, s the quantity of stone stored (previously purchased), p_i the maximum quantity that can still be produced by the seller i and $f(i)$ the value of the influence function for the seller i. This means that only a portion of what is

```
start buyStone{
    while(not isBuilt){
        if(Hc>1){
            buyType1();
        }
    }
    if(He>0){
        activateProducer();
    }
    buyType2();
}
end
```

Code 2.5: Global presentation for the buying process

made available by the seller is recovered, and that this proportion is smaller for more distant sellers.

```
start buyType1{
    while(not isEmpty(producerToVisit) and not isBuilt){
        if(He > 0){
            sortByDistance(producerToVisit);
            producerTemp = producerToVisit[0];
            collectType1 = collectType1 +
                min(besoinType1 - collectType1,
                (producerTemp.productionType1
                x percentageDistance));
        } else {
            randomSort(producerToVisit);
            producerTemp = producerToVisit[0];
            collectType1 = collectType1 +
                min(needType1 - collectType1,
                producerTemp.productionType1);
        }
        producerToVisit = producerToVisit - producerTemp;
    }
}
end
```

Code 2.6: Presentation of the type 1 buying stone

The algorithm 2.8 explains how type 2 stone purchasing works, as does the diagram 5. The algorithm 2.7 explains how new type 2 producers are added if no producer still active is less than a minimum distance away, which in this case corresponds to the value d2 on the graph in figure 4.

As with the purchase of type 1 stones, the list of producers to be visited depends on the level of complexity of the producers. The functioning of the distance differs from the purchase of type 1 stones : the distance function described in the figure 4 acts in this case as a probability of ordering stone from a producer, the further away the producer is and the less likely he is to be chosen. In this graph, the value d2 corresponds to a fixed minimum distance which is an internal variable of each consumer, the variable d1 corresponds to a maximum value fixed in advance, variable of each consumer. Once a producer is selected, a random quantity of the maximum possible recoverable quantity is taken by the consumer. This mechanism proposes to reproduce the notion of negotiation, most probably at work in an important way on the purchase of blocking stone, less prestigious than dimension stone. The presence of a minimum distance (d2) makes it possible to have producers who are definitely selected (if they are at a distance of less than d2).

The following equation represents this process : $r = \sum_{i \in V} alea(min(b - s, p_i))$ with V all sellers (quarries + other sites in case of reuse) sorted from the nearest to the farthest, having at each time step a probability $f(i)$ of selecting the seller i, with b the consumer's need, s the quantity of stone already recovered, $alea$ a random function and p a producer's production.

```
start activateProducer{
    sortByDistance(producerToVisit);
    producerTemp = producerToVisit[0];
    if(distance(self, producteurTemp)>distanceMin){
        creationProducer(He,Hp,type2);
    }
}
end
```

Code 2.7: Creation of new type 2 quarries

2.0.3 Production

Producers have only one active behaviour, which consists in producing stone in anointing what has been ordered from them. Their second behaviour, for type 2 producers (from Hp>0), is to reduce the amount of initial resource on which they can supply consumers.

The quantity of stone produced by each producer at each time step, symbolized in the algorithm 2.9 by the variable *orderedStone*, is defined during the consumer purchase phase. Depending on the level of complexity of the producers, this variable will be limited or not by a maximum production quantity per time step.

```
start buyType2{
    while(not isEmpty(producerToVisit) and not isBuilt){
        if(He > 0){
            trierParDistance(producerToVisit);
            producerTemp = producerToVisit[0];
            if(randomDistance(producerTemp)){
                collectTemp = min(needType2 - collectType2,
                    (producerTemp.productionType2);
                collectType2 = collectType2 +
                    random[0;1] x collectTemp;
            }
        } else {
            random(producerToVisit)
            producerTemp = producerToVisit[0];
            collectType2 = collectType2 +
                min(needType2 - collectType2,
                producerTemp.productionType2);
        }
        producerToVisit = producerToVisit - producerTemp;
    }
}
end
```

Code 2.8: Buying type 2 stone

```
start production{
    stoneProduced = orderedStone;
    orderStoned = 0;
    if(Hp>1 and type==type2){
        ressource = ressource - stoneProduced;
    }
}
end
```

Code 2.9: Stone production

2.0.4 Update of the environment

The simulation update is an operation carried out by the environment at each turn with the aim, depending on the complexity of consumers and producers, of rebuilding consumers already built, setting up reuse and creating new consumers. This global process is explained by the algorithm 2.10 and the diagram in figure 7.

The reconstruction function adds a pre-defined need to consumers who have already met their need, based on a pre-defined probability. The reuse function selects a pre-defined amount of consumer and places 10% of what the consumer

has already harvested as a stone on the list of materials that can be reused by other consumers. Finally, the procedure for creating new consumers adds new prestigious or not, priority or not, according to predefined data.

```
start update{
    if(Hc>0){
        rebuild();
    }
    if(Hp=3){
        reuse();
    }
    repeat nbConsumPrestige/year{
            creationConsumers(He,Hc,true);
    }
    changer priority in (nbPrioritary/an among Consumer);
    repeat nbConsumNotPrestige/year{
            creationConsumers(He,Hc,false);
    }
}
end
```

Code 2.10: Update carried out by the environment

Références

[1] Clémentine Cottineau, Paul Chapron, and Romain Reuillon. Growing models from the bottom up. an evaluation-based incremental modelling method (ebimm) applied to the simulation of systems of cities. *Journal of Artificial Societies and Social Simulation*, 18(4) :9, 2015.