



Université de Cergy-Pontoise

Rapport de Projet

pour l'Unité d'Enseignement "Génie Logiciel et Programmation"
Licence d'Informatique 2e Année

sur le sujet

Histoire

rédigé par

Matteo STAIANO, Mathieu HANOUN



Mai 2017

Table des matières

1 Présentation du projet	3
1.1 Contexte	3
1.2 Objectif du projet avec spécifications	3
1.3 Organisation	3
1.4 Environnements de travail et outils utilisés	3
2 Backend	4
2.1 Données	4
2.1.1 Peuple	4
2.1.2 Évènements	5
2.2 Classes fonctionnelles	6
2.2.1 Guerre et commerce	6
2.2.2 Immigration et croissance	7
2.2.3 RunningLoop	7
2.2.4 Gestionnaire d'évènements	8
2.3 Systèmes de logs et de débogage	9
2.3.1 Log4J	9
2.3.2 JUnit	10
3 Interface Utilisateur	11
3.1 Fenêtre	11
3.2 Écoulement du temps	12
3.3 Statistiques centrales	12
3.4 Graphiques	12
3.5 Sélectionner un peuple	13
3.6 Sélectionner un évènement	14
3.7 Logs	15
3.8 Système de thèmes	15
3.9 Tray Icon	16
4 Conclusion	17

Table des figures

1 Évolution du bellicisme en fonction de la population	5
2 Diagramme UML des différents évènements	6
3 Schéma de fonctionnement de la RunningLoop	7
4 UML de la RunningLoop	8
5 UML des différents gestionnaires	9
6 Aperçu des logs obtenus grâce à Log4J en HTML	10
7 Fenêtre	11
8 ChosingFrame (1re étape)	11
9 ChosingFrame (2e étape)	11
10 Contrôle du temps	12
11 Détails	12
12 Graphique (vue détaillée)	13
13 Graphique (vue globale)	13
14 Sélection du peuple	14
15 Sélection des évènements	14
16 L'affichage des logs	15
17 Différences entre deux thèmes	16
18 L'icône du programme	16

Remerciements

Merci à notre professeur référent T. Liu pour son accompagnement, ses conseils et ses encouragements tout au long de ce projet.

1 Présentation du projet

1.1 Contexte

Le module de Génie Logiciel et Programmation de L2 nous demandant la réalisation d'un projet en Java et souhaitant représenter un système et son évolution suivant différents stimulis, aléatoires ou non, il paraissait judicieux de s'orienter vers un sujet de ce type.

Ayant initialement choisi Psychologie et le projet étant déjà attribué à un autre groupe, le choix d'Histoire parut logique.

1.2 Objectif du projet avec spécifications

L'objectif de ce projet était de réaliser un simulateur d'évènements historiques simulant plusieurs peuples et leurs interactions, positives ou négatives.

Nous souhaitions mettre en place un système d'évènements, aléatoires ou non, pouvant altérer l'état et les statistiques des peuples, et ainsi rendre chaque simulation unique.

Il semblait intéressant d'ajouter également un système de graphiques permettant la mise en avant d'un aspect pédagogique, et facilitant la création d'uchronies.

1.3 Organisation

- Matteo Staiano : Interface Graphique
- Mathieu Hannoun : Conception backend
- Commun : Réflexion algorithmique, debugging, compte-rendus et éléments de livraison finale.

1.4 Environnements de travail et outils utilisés

- Programmation Java : Eclipse, IntelliJ IDEA
- VCS : GitHub
- Production du rapport L^AT_EX : TeXnicCenter

2 Backend

En ce qui concerne la partie backend, c'est à dire le moteur du programme, l'objectif a été de réaliser des structures de données adaptées aux besoins du projet en gardant une séparation entre les classes de données et les classes qui réalisent des opérations, qui seront généralement appelées "Gestionnaire" ou "Manager".

Il a aussi été crucial de pouvoir réaliser toutes les fonctionnalités demandées indépendamment de la partie graphique.

Le fonctionnement du projet se basera sur un système de tours, durant lesquels auront lieu les différents événements et interactions entre peuples.

2.1 Données

2.1.1 Peuple

La classe de données *Peuple* est celle qui sera le plus souvent utilisée. Elle contient les différents attributs de chaque peuple.

Chaque peuple commence avec des attributs principaux bien spécifiques dont découlent des attributs secondaires. Le calcul de ceux-ci est détaillé ci-dessous.

Attributs principaux

- Ressources
- Population
- Agressivité
- Éducation
- Territoire

Attributs secondaires

- Technologie = (Ressources + Éducation) / 2
- Densité = Territoire - Population
- Richesse = (Ressources + Territoire) / 2
- Nombre de soldats = (Population + Agressivité) / 2
- Bellicisme = ($\ln(\text{Population} + 1) / 4$) * ((Richesse + Agressivité) / 2)
- Attractivité = (Richesse + Technologie) / 2
- Puissance militaire = (Technologie + Nombre de soldats) / 2
- Puissance politique = (Puissance militaire + Richesse) / 2
- Immigration = (Densité + Richesse) / 10

Formules développées Les formules ci-dessous sont les mêmes que celles du paragraphe précédent, mais elles ne contiennent ici que des attributs principaux.

Puissance militaire : $(\text{Ressources} + \text{Education} + \text{Population} + \text{Agressivité}) / 4$.
Puissance militaire $\in [0 ; 100]$ (valeur max choisie avec toutes les ressources à 100).

Densité : La densité de population ne correspond pas à un calcul normal de densité, ce qui explique qu'il est possible d'obtenir des valeurs négatives. Nous avons opté pour ce choix afin de simplifier les calculs liés à l'immigration.

Immigration : $(3 * \text{Territoire} - (2 * \text{Peuple}) + \text{Ressources}) / 20$.
Immigration $\in [-10 ; 35]$ (valeur max choisie avec Territoire de 100 et des Ressources de 400, ce qui est cohérent avec les valeurs obtenues lors des simulations). Lorsque toutes les valeurs sont moyennes, l'immigration est à 5.

Belligerance : $(\ln(\text{Population} + 1) * (\text{Ressources} + \text{Territoire} + (2 * \text{Aggressivité}))) / 16$.
 Belligerance $\in [0 ; 202]$ (valeur max choisie avec un Territoire de 100, des ressources de 400 et une agressivité de 100, ce qui est cohérent avec les valeurs obtenues lors des simulations. Lorsque toutes les valeurs sont moyennes, le belligerance est de 50.

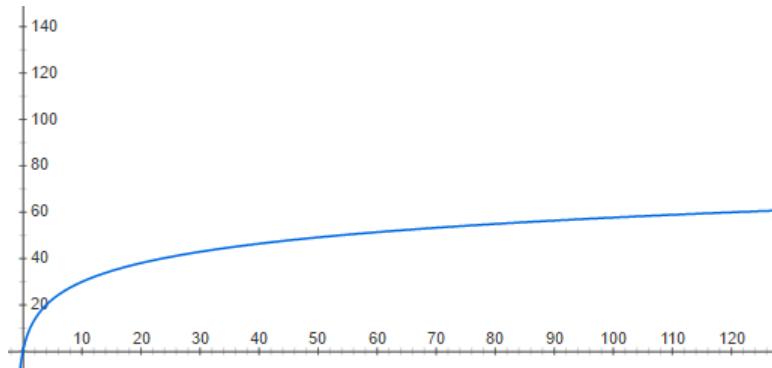


FIGURE 1 – Évolution du belligerance en fonction de la population

Nous avons choisi d'utiliser une fonction logarithmique pour éviter que les pays ne se fassent la guerre systématiquement lorsque leur population devient trop basse, évitant ainsi de faire durer les guerres éternellement.

2.1.2 Événements

Actions locales Au début de chaque "tour¹", chaque peuple a une probabilité donnée (permettant un ajustement) de recevoir un événement aléatoire influant sur ses attributs principaux. L'amplitude de l'événement est elle aussi aléatoire (faible, moyen, fort) et en influencera les conséquences. Par exemple l'événement "Faible" Crise Économique fera perdre des ressources au peuple impacté.

Actions globales Chaque "tour" il y a une probabilité qu'un événement affecte tous les peuples. Celui-ci agira sur l'attribut principal donné de tous les peuples, par exemple l'événement Séisme diminuera la population de chaque peuple d'une valeur comprise entre 1 et 3.

Réactions locales Chaque "tour", après la résolution des actions, chaque peuple aura une probabilité (dépendant des attributs du peuple) de déclencher un événement réaction, qui influera sur ses différents attributs. Par exemple, si la richesse d'un peuple passe au dessus d'un certain seuil (ici 50), l'événement Amélioration de l'éducation¹ pourra se déclencher, augmentant ainsi l'éducation du peuple.

1. Voir chapitre sur la Running Loop

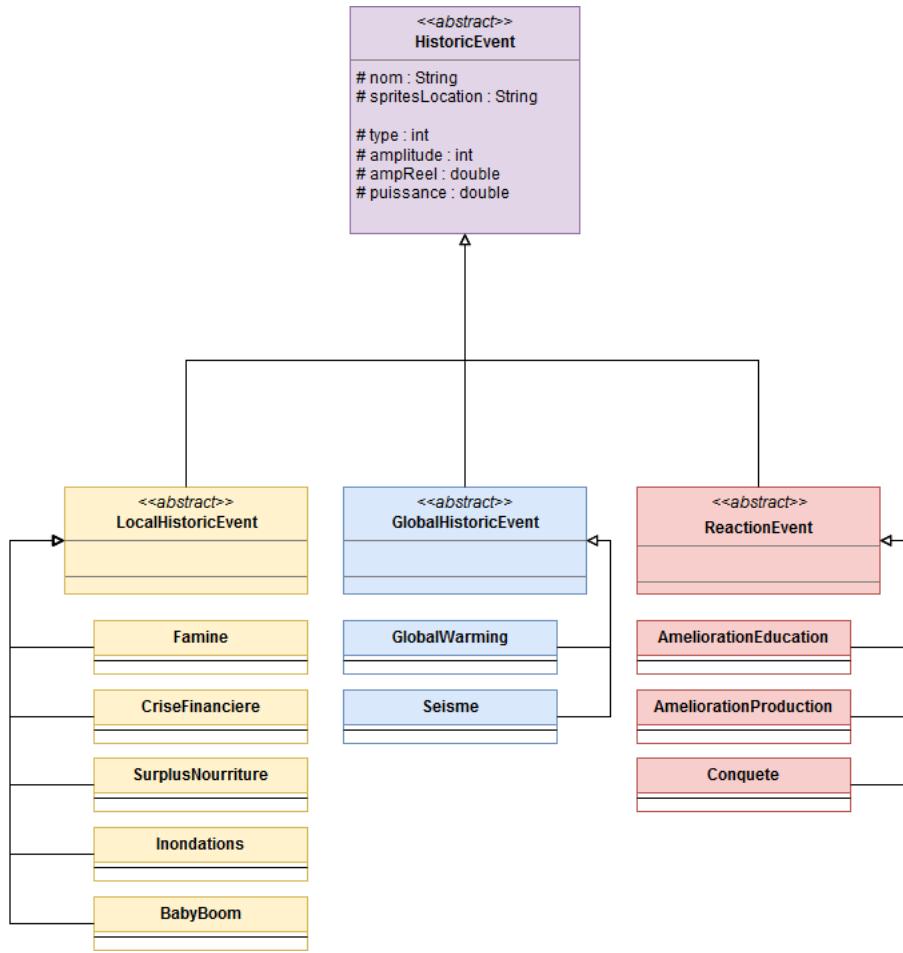


FIGURE 2 – Diagramme UML des différents évènements

2.2 Classes fonctionnelles

2.2.1 Guerre et commerce

Chaque tour, des guerres et des liens commerciaux commencent, ou non, entre les peuples, en fonction de leurs caractéristiques secondaires. Ces relations n'influent que sur les attributs principaux.

La guerre est coûteuse en population et en richesses pour les deux peuples. Le commerce apporte un bénéfice mutuel, mais plus un peuple dispose de puissance politique et plus il sera capable de tirer bénéfice de ses liens commerciaux.

On étudiera chaque binôme de pays potentiel pour identifier lesquels ont les valeurs d'attributs nécessaires pour entrer en guerre.

Un couple de peuple est "éligible" à entrer en guerre si la somme de leurs bellicismes est supérieure ou égale à un seuil fixé (ici 150).

Pour le commerce, il s'agit de la somme de leur attractivité qui sera prise en compte, on étudiera comme ci dessous pour éviter tout doublon.

	Peuple 1	Peuple 2	Peuple 3
Peuple 1	X	Guerre / Commerce	Guerre / Commerce
Peuple 2	X	X	Guerre / Commerce
Peuple 3	X	X	X

Les conséquences de la guerre sont ensuite résolues, la population de chaque pays en guerre est diminuée de la puissance militaire de leur adversaire multipliée par un coefficient de réduction, ici 0.01.

La guerre est traitée avant le commerce, car dans notre système deux pays en guerre ne peuvent pas commercer entre eux.

2.2.2 Immigration et croissance

Chaque "tour", les pays perdront ou gagneront de la population, proportionnellement à leur taux d'immigration (un taux positif signifie qu'un pays attire les migrants, tandis qu'une immigration négative poussera les habitants du pays en question à le quitter).

Une amélioration possible serait une approche plus réaliste en réalisant un "pool" de population constitué de toutes les populations quittant leurs pays de départ. Cette masse de population serait répartie en fonction du taux d'immigration positive de chaque pays et, s'il y a des "excédents" par rapport à la demande, ceux-ci seraient attribués par rapport au pays de départ.

La croissance est une augmentation fixe de la population de chaque pays à chaque tour, ici la croissance sera de 1 par tour.

L'immigration est gérée dynamiquement en fonction des besoins de chaque pays. Un pays ayant une densité de population négative² verra sa population en trop "quitter" le pays et être affectée à une "pool" de population qui sera répartie sur les pays en demande de population, l'ordre de répartition étant aléatoire et changeant à chaque tour pour éviter que les premiers pays de la liste ne s'accaparent tous les flux migratoires.

2.2.3 RunningLoop

La Running loop, ou boucle de fonctionnement est l'épine dorsale de la partie backend. Elle va gérer l'ordre dans lequel seront effectués les différents calculs liés aux guerres, commerces et événements, selon le schéma ci-dessous.

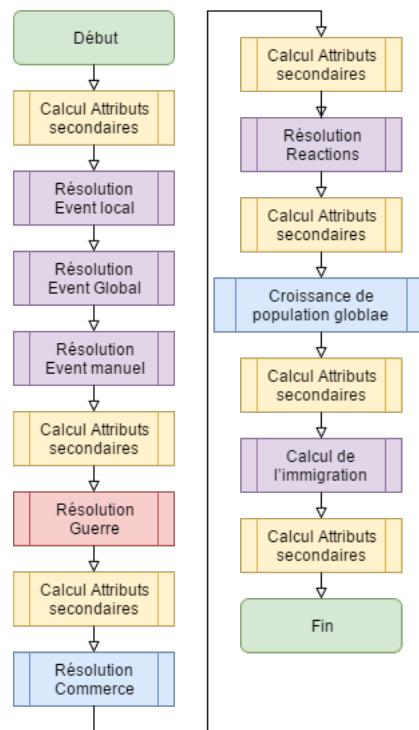


FIGURE 3 – Schéma de fonctionnement de la RunningLoop

Étant l'une des classes principales de la partie calcul et n'étant instanciée qu'une seule fois, c'est elle qui contiendra les différents éléments statiques :

-
2. Se référer au calcul de la densité.

- Logs de débogage
- Logs à afficher à l'utilisateur
- Nombre de cycles écoulés
- Liste d'évènements entrés manuellement

Cette classe implémente aussi les différentes méthodes pour ajouter du texte aux différents logs.

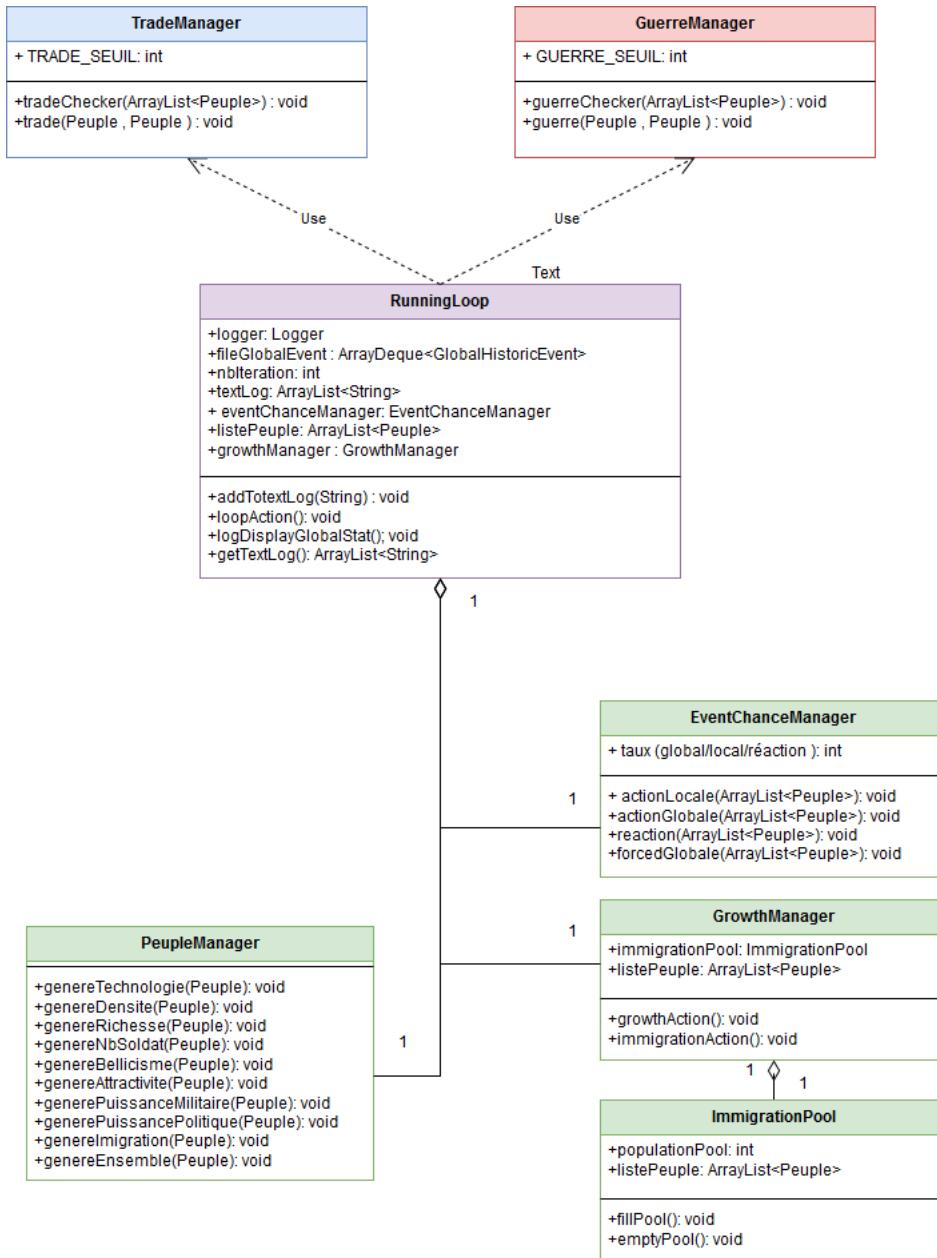


FIGURE 4 – UML de la RunningLoop

2.2.4 Gestionnaire d'évènements

Il existe un gestionnaire pour chaque type d'évènement (local, global, réaction). Ceux-ci instancient chacun une liste contenant tous les différents éléments.

C'est dans ces classes que sont implémentées les différentes méthodes d'action des évènements, tandis que les probabilités d'apparition des évènements sont gérées par la classe EventChanceManager. L'objectif était ici d'encapsuler chaque élément de manière à rendre le programme le plus clair possible.

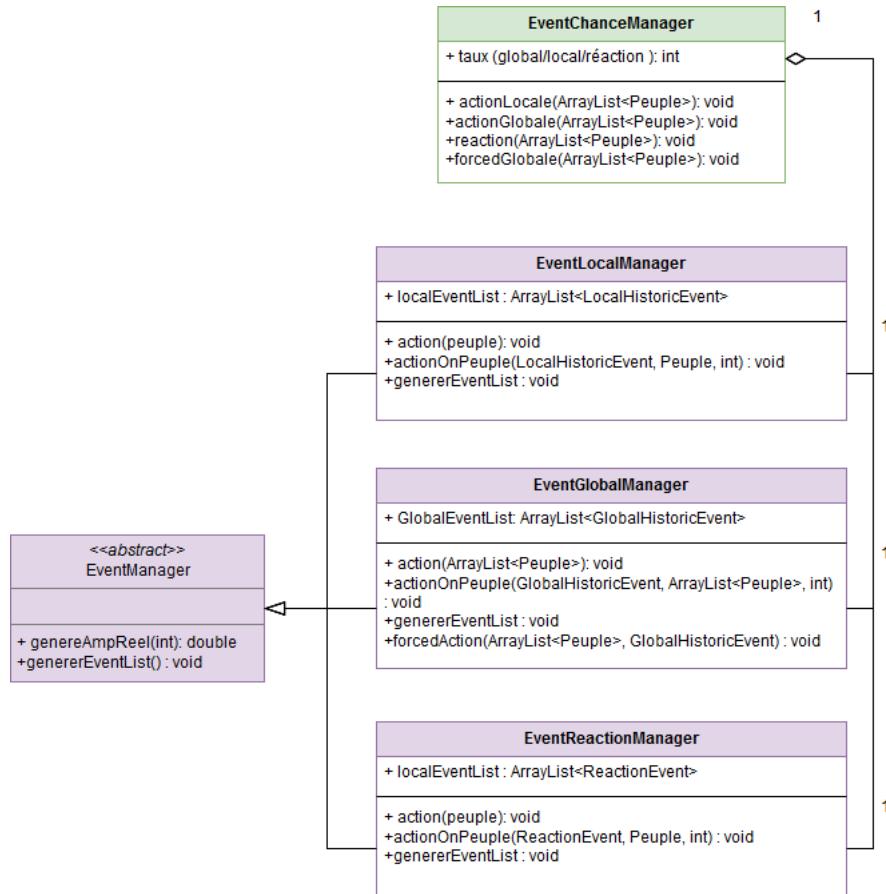


FIGURE 5 – UML des différents gestionnaires

2.3 Systèmes de logs et de débogage

2.3.1 Log4J

Durant les premières phases du développement, nous avons été confrontés à plusieurs problèmes de débogage et d'équilibrage dans les mécaniques mises en place.

Plutôt d'utiliser, comme nous le faisions précédemment, de simples envois de chaînes de caractères sur le flux d'erreur standard, nous avons choisi de mettre en place et d'utiliser la librairie Log4J pour obtenir un système de logs plus complet.

Cela nous a permis de mettre en forme les différentes données renvoyées par le programme, de les hiérarchiser et de pouvoir comprendre quand et comment certaines fonctionnalités pouvaient poser problème. Par exemple, ce système nous a permis de résoudre un bug faisant augmenter de manière exponentielle les ressources des différents pays à partir d'un certain nombre de cycles.

Log session start time Thu May 18 23:30:28 CEST 2017				
Time	Thread	Level	Category	Message
Log session start time Thu May 18 23:30:30 CEST 2017				
Time	Thread	Level	Category	Message
Log session start time Thu May 18 23:30:30 CEST 2017				
Time	Thread	Level	Category	Message
0	AWT-EventQueue-0	INFO	ugp.gp.histoire.managers.TradeManager	Commerce entre Aztèques et Belges iteration : 0
1	AWT-EventQueue-0	INFO	ugp.gp.histoire.managers.TradeManager	Commerce entre Aztèques et Britanniques iteration : 0
1	AWT-EventQueue-0	INFO	ugp.gp.histoire.managers.TradeManager	Commerce entre Aztèques et Byzantins iteration : 0
1	AWT-EventQueue-0	INFO	ugp.gp.histoire.managers.TradeManager	Commerce entre Aztèques et Chinois iteration : 0
2	AWT-EventQueue-0	INFO	ugp.gp.histoire.managers.TradeManager	Commerce entre Aztèques et Espagnols iteration : 0
2	AWT-EventQueue-0	INFO	ugp.gp.histoire.managers.TradeManager	Commerce entre Aztèques et Grecs iteration : 0
2	AWT-EventQueue-0	INFO	ugp.gp.histoire.managers.TradeManager	Commerce entre Belges et Britanniques iteration : 0
2	AWT-EventQueue-0	INFO	ugp.gp.histoire.managers.TradeManager	Commerce entre Belges et Byzantins iteration : 0
2	AWT-EventQueue-0	INFO	ugp.gp.histoire.managers.TradeManager	Commerce entre Belges et Chinois iteration : 0
2	AWT-EventQueue-0	INFO	ugp.gp.histoire.managers.TradeManager	Commerce entre Belges et Espagnols iteration : 0
2	AWT-EventQueue-0	INFO	ugp.gp.histoire.managers.TradeManager	Commerce entre Belges et Grecs iteration : 0
2	AWT-EventQueue-0	INFO	ugp.gp.histoire.managers.TradeManager	Commerce entre Byzantins et Chinois iteration : 0
2	AWT-EventQueue-0	INFO	ugp.gp.histoire.managers.TradeManager	Commerce entre Byzantins et Espagnols iteration : 0
2	AWT-EventQueue-0	INFO	ugp.gp.histoire.managers.TradeManager	Commerce entre Byzantins et Grecs iteration : 0
2	AWT-EventQueue-0	INFO	ugp.gp.histoire.managers.TradeManager	Commerce entre Chinois et Grecs iteration : 0
2	AWT-EventQueue-0	INFO	ugp.gp.histoire.managers.TradeManager	Commerce entre Chinois et Espagnols iteration : 0
2	AWT-EventQueue-0	INFO	ugp.gp.histoire.managers.TradeManager	Commerce entre Chinois et Grecs iteration : 0
2	AWT-EventQueue-0	TRACE	ugp.gp.histoire.managers.RunningLoop	Peuple [nom=Aztèques, population=21, ressources=87, 278125, education=30.0, territoire=70.0, agresseur=30.0, nbSoldat=30, migration=10, technologie=48.6390825, densité=39.0, richesse=68.6390825, attractivité=58.6390825240999994, puissanceMilitaire=39.31953125, puissancePolitique=42.399218749999994] iteration : 0
3	AWT-EventQueue-0	TRACE	ugp.gp.histoire.managers.RunningLoop	Peuple [nom=Belges, population=51, ressources=99.73124999999999, education=50.0, territoire=70.0, agresseur=50.0, nbSoldat=50, migration=8, technologie=49.855825, attractivité=64.855825, puissanceMilitaire=54.9328125, puissancePolitique=42.39921874999999] iteration : 0
3	AWT-EventQueue-0	TRACE	ugp.gp.histoire.managers.RunningLoop	Peuple [nom=Britanniques, population=31, ressources=89.9884375, education=80.0, territoire=40.0, agresseur=50.0, nbSoldat=50, migration=5, technologie=74.94921875, attractivité=56.77404731772249, puissanceMilitaire=82.474609375, puissancePolitique=71.014605515] iteration : 0

FIGURE 6 – Aperçu des logs obtenus grâce à Log4J en HTML

2.3.2 JUnit

Dans la même optique que précédemment et pour aller un peu plus loin en automatisant les tests sur certaines des fonctions de notre moteur, nous avons choisi d'utiliser le framework JUnit. Celui-ci nous a notamment été utile pour tester les différentes évolutions des peuples au fil des tours, sur de très grandes itérations.

3 Interface Utilisateur

3.1 Fenêtre

La fenêtre principale du programme a une taille prédéfinie de 1280x720 pixels et n'est pas redimensionnable. Cette dimension a été choisie car elle correspond à une résolution de 16 :9 et est à la fois suffisamment petite pour ne pas déborder de la plupart des écrans, et suffisamment grande pour nous permettre d'afficher des éléments suffisamment détaillés.

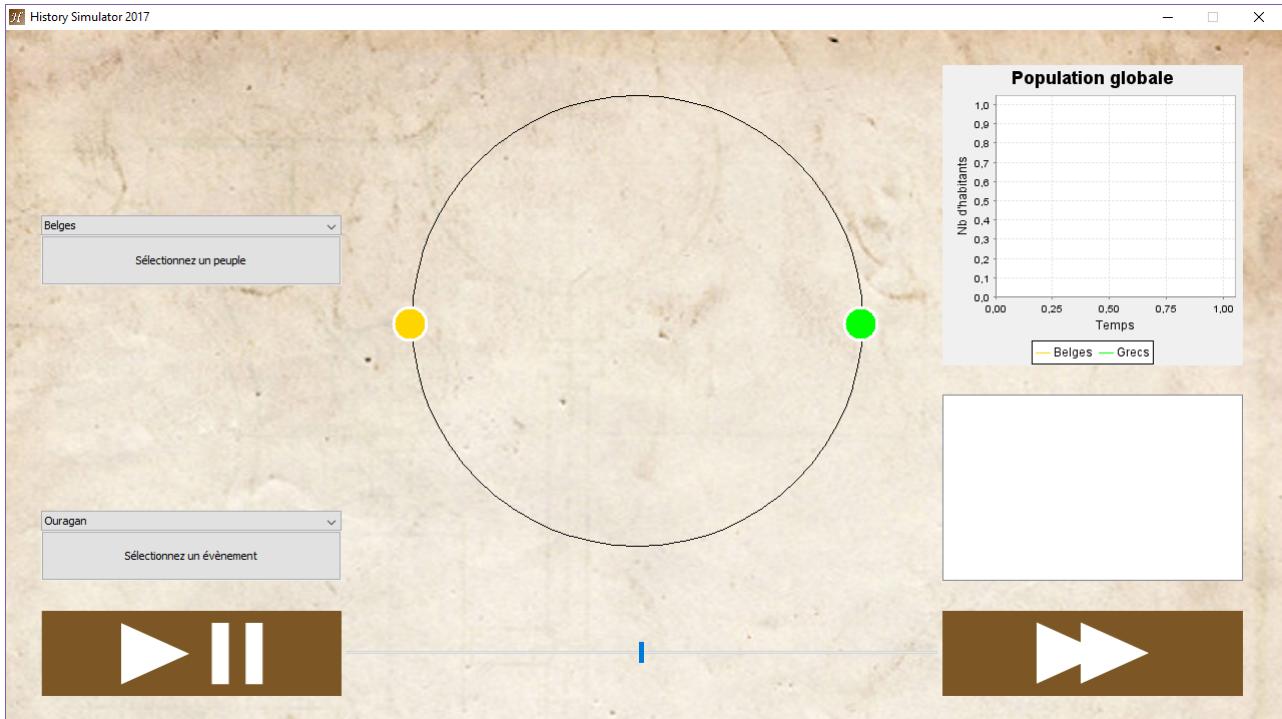


FIGURE 7 – Fenêtre

Une fenêtre secondaire s'ouvre avant la fenêtre principale : la ChosingFrame. Celle-ci permet de sélectionner dans un premier temps le nombre de peuples (entre 2 et 16 pour des soucis d'optimisation), puis de choisir quels seront les peuples en question.

Un menu déroulant présent en bas de cette fenêtre permet à l'utilisateur de choisir le thème qu'il souhaite utiliser (voir plus bas pour les détails).

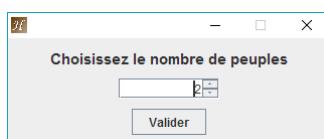


FIGURE 8 – ChosingFrame (1re étape)

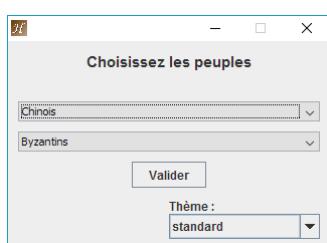


FIGURE 9 – ChosingFrame (2e étape)

3.2 Écoulement du temps

Dans la partie basse de l'interface, trois éléments permettent de gérer le passage du temps. À gauche, un bouton permet de mettre en pause ou de reprendre le passage automatique des cycles. Au milieu, un curseur sert à régler la vitesse d'écoulement du temps lors des cycles automatiques. Enfin, à droite, un bouton permet de passer au cycle suivant manuellement, même si l'écoulement du temps est en pause.



FIGURE 10 – Contrôle du temps

3.3 Statistiques centrales

Au centre de la fenêtre, des informations sont affichées. Par défaut, il s'agira d'un diagramme représentant les pays ainsi que leurs relations. Chaque pays possède une couleur et un halo changeant de couleur en fonction de ses relations : rouge s'il est en guerre, bleu s'il fait du commerce, blanc sinon. De plus, des traits lient les peuples en fonction de leurs relations (mêmes codes couleurs que pour les halos).

Si un peuple est sélectionné³, les statistiques affichées seront celles du peuple en question, et plus de détails seront fournis sur celui-ci.

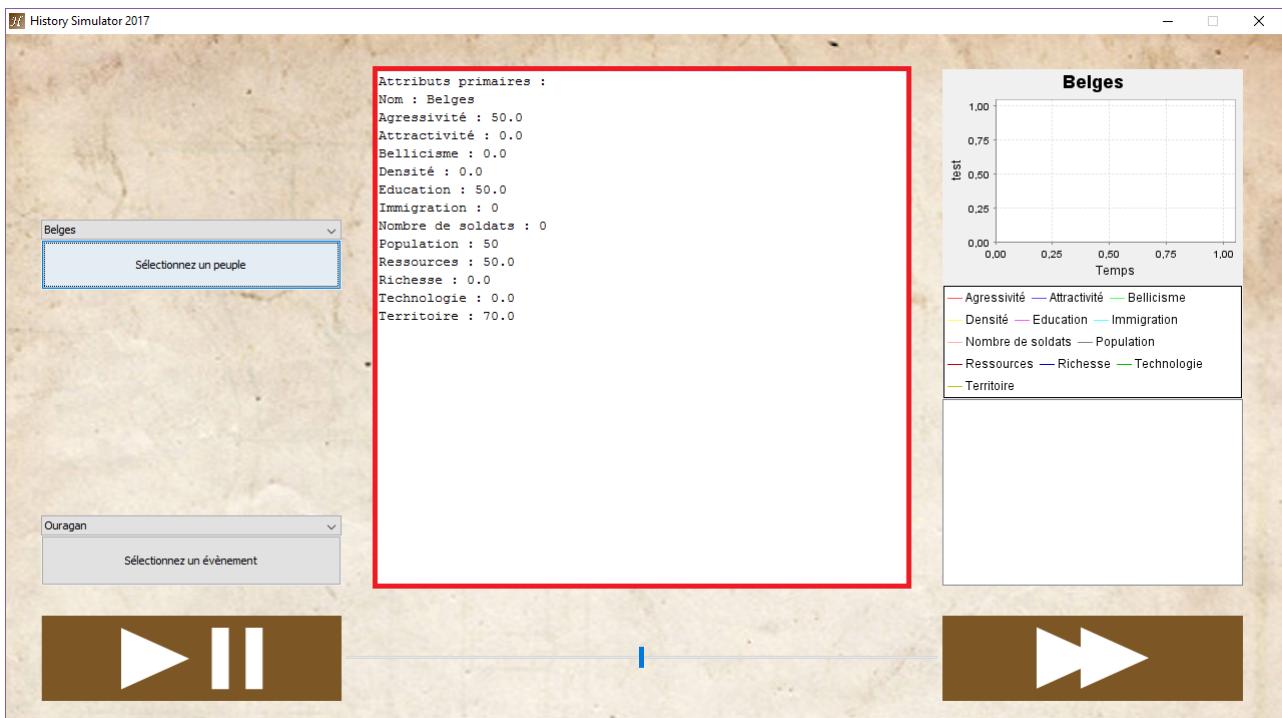


FIGURE 11 – Détails

3.4 Graphiques

En haut à droite de la fenêtre, un graphique en courbes est présent et affiche par défaut la population de chaque peuple.

Lorsqu'un peuple est sélectionné, le graphique montre l'ensemble de ses données plutôt que la population totale, permettant ainsi un aperçu visuel de l'état des différents peuples.

3. Voir plus bas pour la sélection d'un peuple

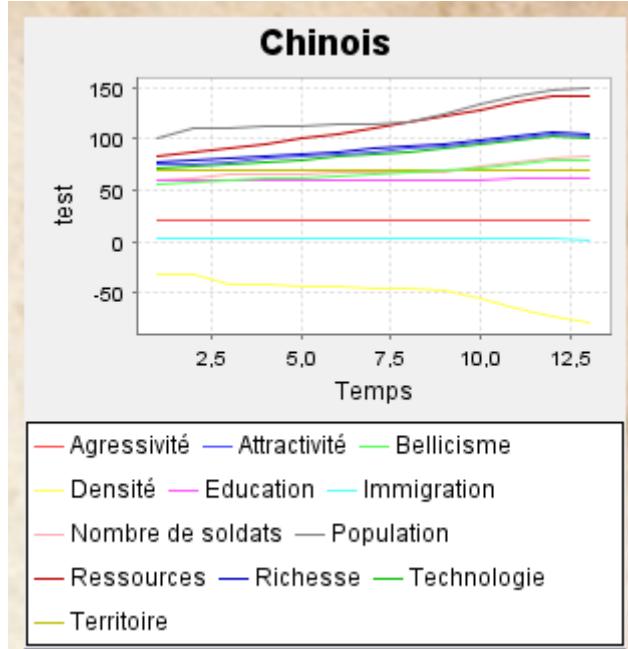


FIGURE 12 – Graphique (vue détaillée)

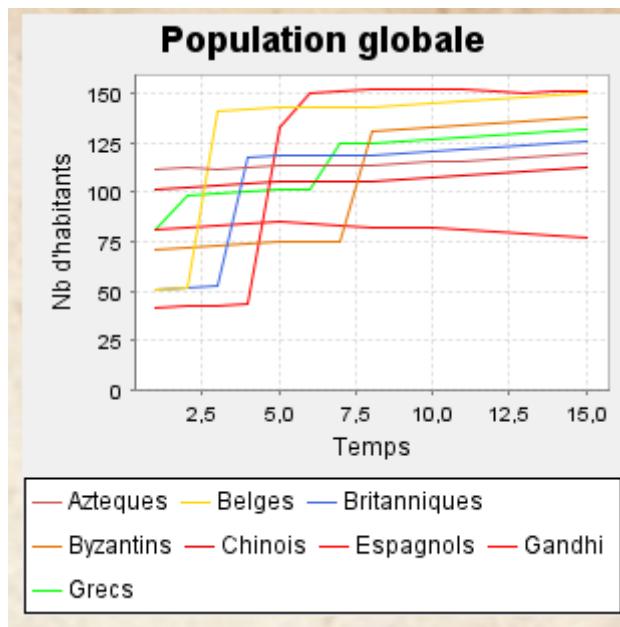


FIGURE 13 – Graphique (vue globale)

3.5 Sélectionner un peuple

Un menu déroulant comportant tous les peuples et un bouton de validation sont à gauche de la fenêtre. Ceux-ci permettent de sélectionner un peuple. En validant, on a accès à ses statistiques détaillées au centre de l'interface.

Il y a également une option *Aucun* permettant de revenir à l'affichage des statistiques globales des relations entre les peuples.

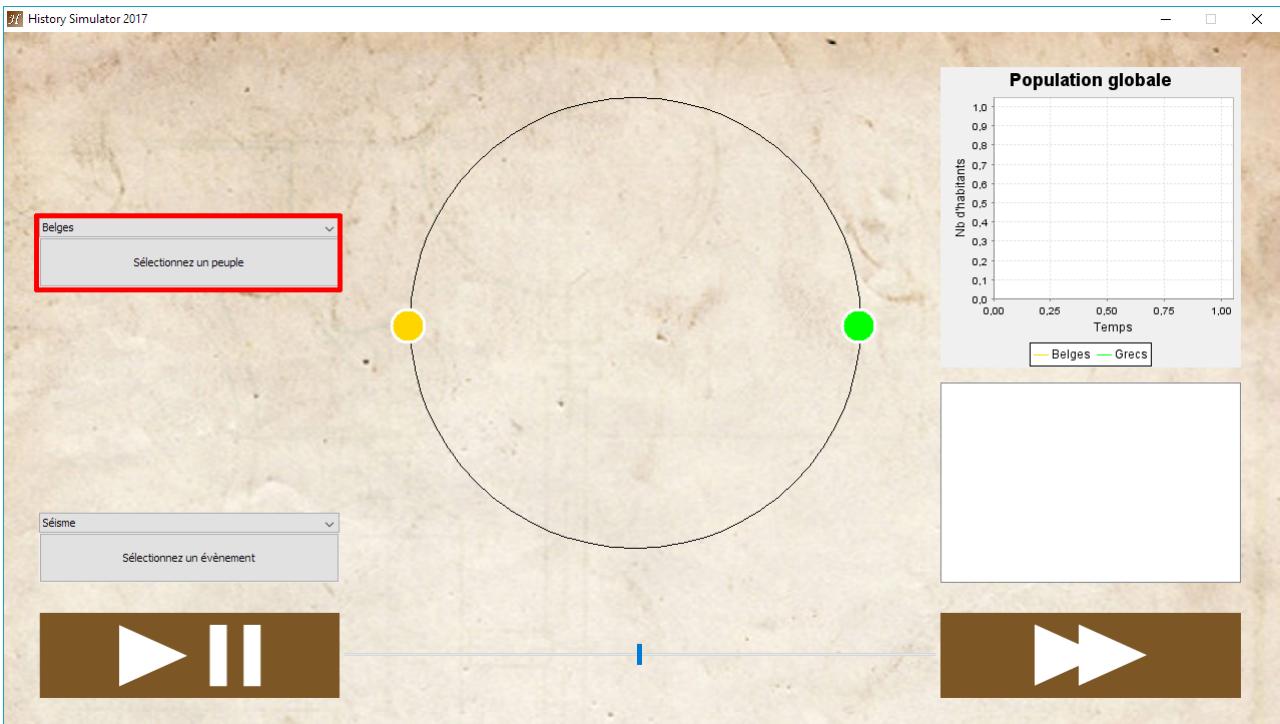


FIGURE 14 – Sélection du peuple

3.6 Sélectionner un évènement

Les évènements peuvent être déclenchés manuellement par l'utilisateur, via un menu déroulant. Les évènements envoyés manuellement sont globaux, c'est-à-dire qu'ils concernent l'ensemble des peuples. Lorsqu'un évènement est envoyé manuellement, il sera déclenché au début du prochain cycle. Ainsi, si l'utilisateur envoie par exemple trois séismes, les trois se déclencheront simultanément.

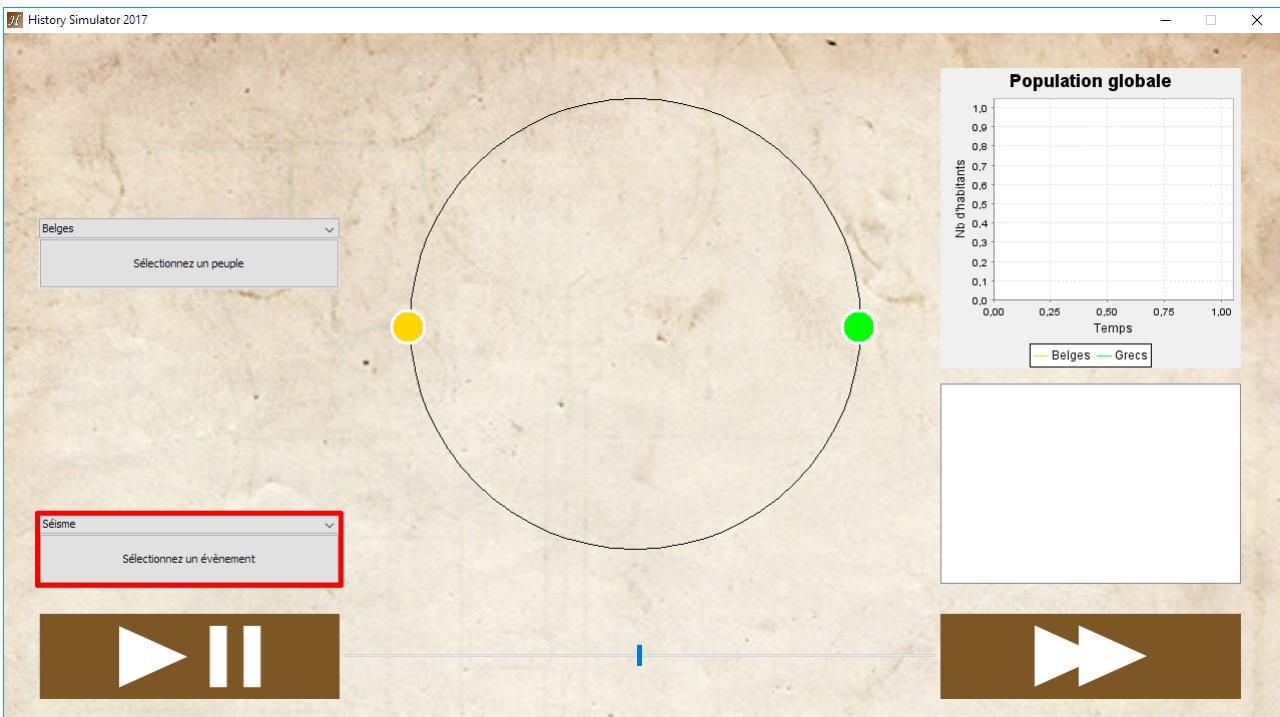


FIGURE 15 – Sélection des évènements

3.7 Logs

Une zone de texte, en bas à droite de la fenêtre principale, présente les logs. Tout ce qu'il se passe dans la simulation est retranscrit ici, avec un code couleur permettant de différencier les différents événements :

- Rouge : guerre
- Bleu : commerce
- Vert : évènements

Ainsi, il est possible de conserver un historique de tout ce qu'il s'est passé dans les derniers cycles. Initialement, nous avions prévu de gérer les couleurs en utilisant un formatage HTML et en affichant ensuite la page ainsi créée, mais nous avons ensuite décidé d'utiliser les objets Document et Style de Swing pour des raisons techniques.

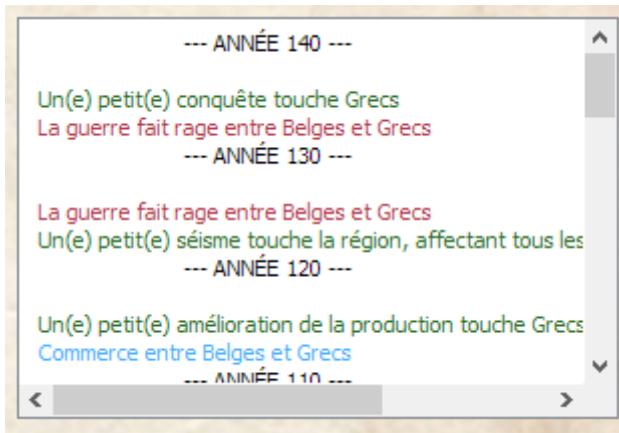


FIGURE 16 – L'affichage des logs

3.8 Système de thèmes

Lors de la sélection initiale des peuples, il est possible de choisir un thème via un menu déroulant. Ces thèmes ne modifient que l'aspect graphique du programme et permettent ainsi une forme de personnalisation de l'interface par l'utilisateur.

Les thèmes sont simplement différents dossiers contenant les images utilisées dans le GUI. Ainsi, changer de thème revient à changer de dossier source pour les images, il est donc possible d'ajouter un grand nombre de thèmes facilement.

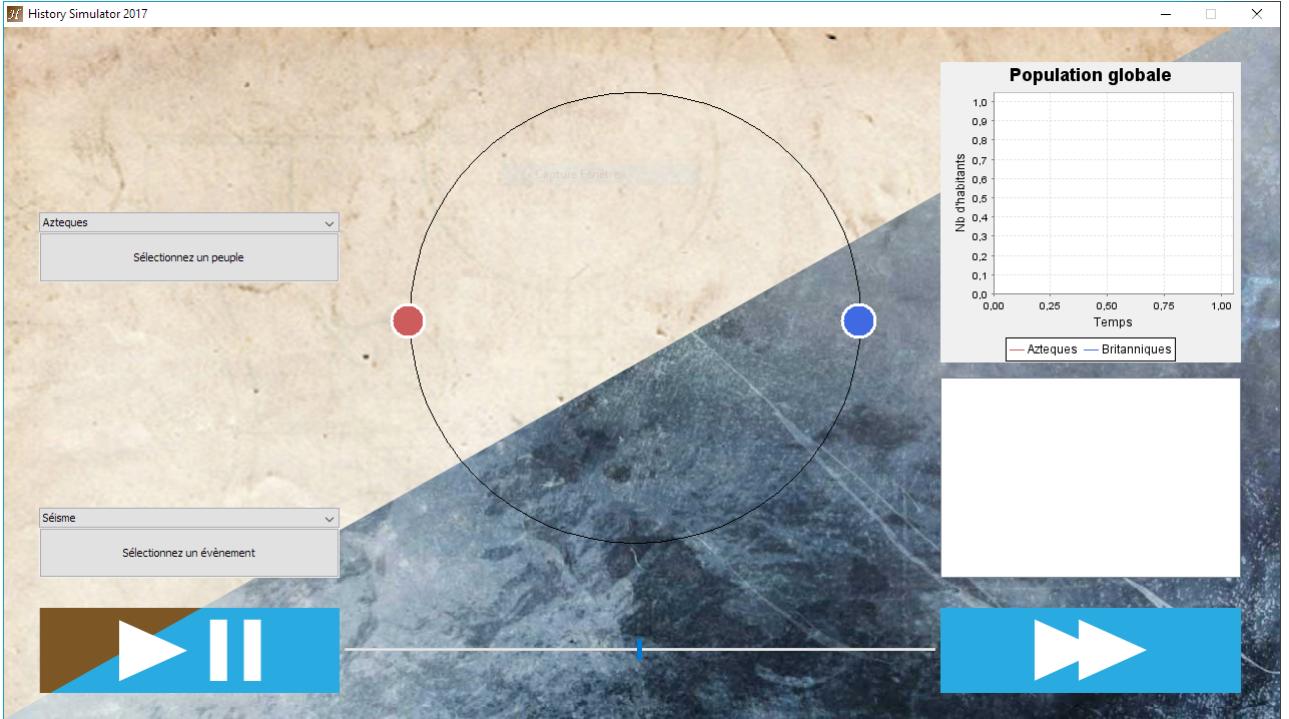


FIGURE 17 – Différences entre deux thèmes

3.9 Tray Icon

Le programme dispose d'une icône dans la barre des tâches.
Celle-ci est composée d'un H stylisé sur un fond dégradé pouvant changer en fonction du thème sélectionné.
On peut utiliser cette icône pour réduire ou restaurer la fenêtre.



FIGURE 18 – L'icône du programme

4 Conclusion

Au terme de ce projet, nous avons beaucoup appris et sommes fiers du programme final. Néanmoins, celui-ci est loin d'être parfait, et plusieurs améliorations seraient encore possibles. Il serait par exemple possible de permettre à l'utilisateur de créer lui-même les peuples qu'il souhaite voir en jeu, ou le thème qu'il souhaite utiliser. L'équilibrage n'est lui non plus pas parfait, et il serait intéressant de créer un système d'équilibrage automatique en fonction des résultats des dernières simulations.

Toutefois, nous sommes parvenus à nous organiser correctement de manière à ce que l'interface utilisateur et le moteur avancent de manière parallèle et synchronisée. Les objectifs ont été réalisés dans les délais que nous avions convenu et nous avons tous deux consolidé nos connaissances générales et nos méthodes de travail.