Rapport de projet

Sina Monnet
Alan Quinton
Keryann Orain
Baptiste Crémieux
Lucile Vignon - - Prost

Sommaire

1 Introduction

1.1 Entreprise

Fondée en octobre 2025, Quantum Tactics est un studio de développement indépendant né d'une synergie parfaite entre cinq passionnés d'informatique et créateurs de mondes virtuels. L'entreprise s'est forgée autour d'une ambition claire : concevoir des expériences ludiques exigeantes qui repoussent les limites de la réflexion en fusionnant la profondeur stratégique avec les fascinantes mécaniques du développement scientifique.

Cette double identité, à la fois analytique et tactique, est l'essence même de notre studio. Elle se reflète d'ailleurs directement dans notre nom :

- "Quantum" incarne notre fascination pour les sciences et la technologie. Il représente notre volonté d'intégrer des systèmes profonds, comme la recherche avancée ou la génération procédurale d'univers, au cœur de notre jeu.
- "Tactics" souligne notre engagement envers le genre de la stratégie. Nous croyons en des jeux où l'intellect, l'optimisation des ressources et l'anticipation priment sur la force brute, offrant aux joueurs un contrôle total sur leur destinée.

1.2 Equipe

Sina Monnet

Sina dirige l'équipe et la vision globale de Quantum Tactics. Passionné par la gestion d'équipe et l'innovation, il coordonne la stratégie et les objectifs de l'entreprise. Il est également chargé de réfléchir à la conception de certaines mécaniques principales du jeu afin d'en expliquer le fonctionnement aux membres responsables de leur création. Enfin, Sina harmonise le code produit par l'équipe et corrige certains défauts de programmation empêchant le bon fonctionnement du jeu.

Alan Quinton

Alan conçoit l'identité visuelle du jeu et de la marque. Son sens du détail donne vie à l'univers graphique de Quantum Tactics. Il imagine ainsi l'interface du jeu et participe activement au développement de ce dernier, notamment en créant

différents menus pour le jeu afin de garantir une cohérence entre technique et direction artistique. De même, il est responsable de l'implémentation d'un système de recherches pour une progression fluide au cours de la partie.

Keryann Orain

Keryann teste, remet en question et affine les expériences de jeu. Il garantit l'intuitivité et la qualité de l'expérience utilisateur. Il s'assure de la qualité des fonctionnalités implémentées par les autres membres de l'équipe. De plus, Keryann développe et supervise les aspects réseau et multijoueur du jeu. Enfin, il participe activement à la création de fonctionnalités utilitaires telles que les boutons ou les zones de renseignement de texte.

Baptiste Crémieux

Baptiste gère le planning et les jalons du projet. Il est garant de la rigueur temporelle de ce dernier, permettant une organisation optimale. De plus, il s'occupe du développement du site internet et de la base de données servant à gérer les comptes des utilisateurs. Pour finir, il est responsable du système d'attaque entre les joueurs.

Lucile Vignon - - Prost

Lucile assure la traçabilité des actions et décisions de l'équipe, elle produit ainsi des documents clairs et exploitables sur le travail effectué par l'ensemble de l'équipe. De plus, elle conçoit et met en place les mécaniques de base du jeu, assurant une expérience de jeu cohérente, équilibrée et fonctionnelle. Enfin, elle a imaginé l'histoire du jeu et s'est occupée de fournir un tutoriel pour les nouveaux joueurs.

2 Exigences du cahier des charges

2.1 Exigences globales

La définition initiale du projet impose une compatibilité avec le système d'exploitation Windows. Développé en Python à l'aide de la bibliothèque Pygame, il s'agit d'un

jeu en deux dimensions en temps réel se déroulant sur une carte interactive générée de manière procédurale. Le jeu devra être jouable aussi bien en solitaire qu'en multijoueur en temps réel, tout en intégrant un système d'intelligence artificielle afin d'enrichir les possibilités de jeu. Enfin, il devra inclure plusieurs bandes-son, comprenant à la fois une musique d'ambiance et des effets sonores interactifs liés aux actions des joueurs.

2.2 Fonctionnalités internes au jeu

Le tableau suivant indique la présence des fonctionnalités prévues au début du projet ainsi que leur statut (obligatoires ou souhaitées).

Domaine	Fonctionnalité	Statut	Réalisée	Non réalisée
Mode multijoueur	Salon de jeu	Obligatoire	X	
	Messagerie textuelle	Obligatoire	X	
	Comptes utilisateur	Obligatoire	X	
Mécaniques de jeu	Carte interactive	Obligatoire	X	
	I.A. joueuse	Obligatoire	X	
	Attaque et défense	Obligatoire	X	
	Gestion des ressources	Obligatoire	X	
	Bâtiments	Obligatoire	X	
	Arbre de recherche	Obligatoire	X	
	Alliance entre joueurs	Souhaitée		X
	Commerce entre joueurs	Obligatoire	X	
	Victoire	Obligatoire	X	
	I.A. pirates	Souhaitée	X	
	Personnage du "Supreme Leader"	Souhaitée		X
Menu	Solitaire	Obligatoire	X	
	Multijoueur	Obligatoire	X	
	Lancement de partie	Obligatoire	X	
	Paramètres	Obligatoire	X	
	Connexion	Obligatoire	X	
	Crédits	Obligatoire	X	

3 Réalisation du projet

3.1 Répartition des tâches

Domaine	Fonctionnalité	Sina	Alan	Keryann	Baptiste	Lucile
Carte	Génération	X				X
	Interaction	X				
Interface utilisateur	Musique	X				
	Apparence		X	X		
	Histoire			X		X
Fonctionnement du jeu	Intelligences Artificielles	X		X		
	Arbre de recherche		X	X		
	Victoire	X	X			
	Multijoueur	X		X		
	Gestion des ressources	X				X
	Base de données				X	
	Système d'attaque	X			X	
Mécaniques de développement	Console	X				

En dehors du jeu en lui-même, Alan a été chargé de l'identité graphique de l'entreprise. Baptiste a développé le site internet et Lucile s'est occupée de la partie communication.

3.2 Carte

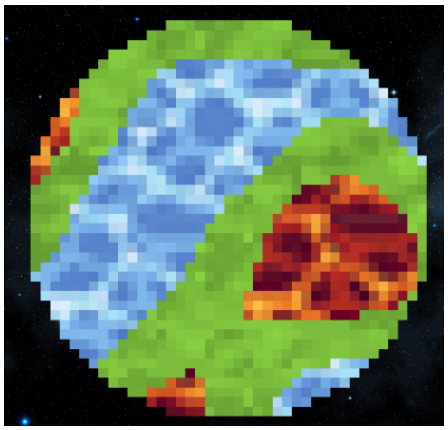
a. Génération

La génération de la carte repose sur une graine, c'est-à-dire un entier compris entre 10 000 et 99 999. Cette graine détermine la position des planètes, leur taille ainsi que leurs biomes.

Les planètes sont générées de manière à rester relativement proches les unes des autres tout en respectant une distance minimale. Pour cela, elles sont créées une par une. À chaque nouvelle planète, le programme tente de la positionner autour de la précédente, puis vérifie qu'elle n'entre pas en collision avec un lien existant et qu'elle n'est pas trop proche d'une autre planète.

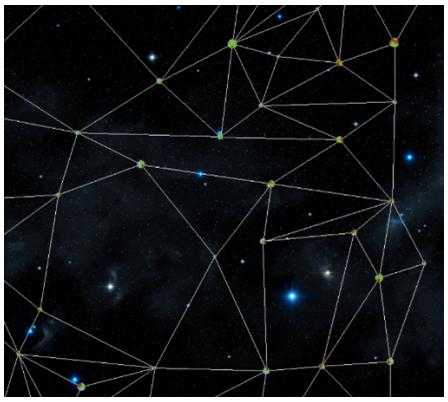
Le nombre total de planètes dépend de deux facteurs : le nombre de joueurs et un coefficient multiplicateur. Ce coefficient est actuellement fixé à 20, ce qui permet une génération rapide tout en garantissant une taille de carte suffisante pour proposer une partie divertissante.

La création de l'apparence de chaque planète ainsi que de ses écosystèmes s'effectue par bruit de Perlin. Pour rendre le processus plus rapide, une seule couche est générée et les planètes sont prédéfinies comme des images de 40 pixels par 40 pixels. Il n'existe pour l'instant que trois environnements, à savoir l'océan, les plaines et les zones volcaniques, mais il est possible d'ajouter facilement autant de biomes que souhaité. La figure ci-contre est un exemple d'apparence de planète.

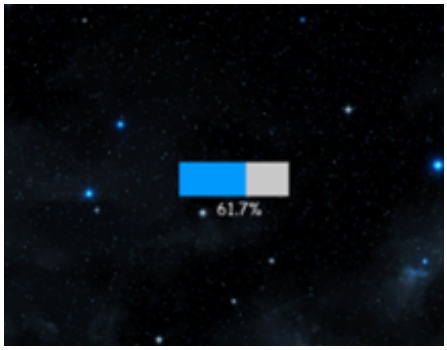


Chaque planète possède des liens avec d'autres planètes. Ceux-ci sont générés aléatoirement avec une distance maximale et ne peuvent pas croiser un autre lien ou traverser une planète. La capture d'écran ci-contre montre l'affichage de ces liens.

Pour rendre la génération plus rapide, le programme ne vérifie pas les croisements entre les liens trop éloignés. La méthode de calcul des croisements détecte directement certains cas et ne les calcule pas. Le temps de génération est divisé par deux au minimum.



Lors du temps d'attente de la génération une barre de chargement correspondant au nombre de planètes créées par rapport au nombre total de planète est affichée, permettant à l'utilisateur de voir la progression de ce processus (voir la figure ci-contre).



Le calcul et l'affichage des éléments de la carte s'effectue uniquement lorsqu'une action modifie cette dernière, comme une attaque de joueur ou un déplacement.

b. Interaction

Le joueur peut se déplacer grâce à sa souris lorsqu'il est sur la carte. A chaque déplacement détecté, le programme calcule le vecteur déplacement de la souris et « l'ajoute » aux coordonnées des planètes. Les liens entre planètes sont déplacés en conséquent.

L'implémentation d'un système d'agrandissement de la carte permet au joueur d'agrandir ou de réduire la taille de la carte grâce à la molette de sa souris lorsque la fenêtre active est la carte de jeu. Chaque fois que la molette est utilisée, le coefficient d'agrandissement ainsi que les coordonnées des planètes sont mis à jour en fonction de la position de la souris. Le coefficient d'agrandissement est utilisé lors de l'affichage des planètes pour calculer la taille de celles-ci.

3.3 Interface utilisateur

a. Musique

La gestion de l'audio est centralisée dans un fichier. La fonction asynchrone musique s'appuie sur le système de canaux de Pygame Mixer pour lire des fichiers audio, avec la possibilité d'utiliser jusqu'à huit canaux en simultané. Cela permet de superposer librement des musiques d'ambiance et des effets sonores ponctuels sans qu'ils ne s'interrompent mutuellement. Un mécanisme de réinitialisation permet de stopper instantanément l'ensemble des sons en cours, utile lors des transitions entre les menus et le jeu.

Les effets sonores courts, notamment le son de survol et le son de clic associés aux boutons qui sont gérés séparément. Les fichiers sont chargés une seule fois au démarrage et conservés en mémoire dans un dictionnaire de cache, ce qui évite toute latence lors de la lecture. Toute la logique de lecture est conçue pour

fonctionner en parallèle de la boucle de jeu grâce à la bibliothèque asyncio, sans jamais bloquer le rendu ni les entrées utilisateur.

b. Apparence

Une image de l'interface de jeu est disponible figure 1 pour une meilleure compréhension des descriptions suivantes.

Découpage de l'écran en zones

L'une des premières problématiques rencontrées lors de la conception de l'interface était de rendre l'affichage cohérent quelle que soit la taille de la fenêtre. Une fonction répond à ce besoin en calculant à chaque rafraîchissement d'image les dimensions et positions de toutes les zones de l'interface à partir de la taille réelle de la fenêtre courante.

L'écran est découpé en quatre zones principales : une barre supérieure représentant 8 % de la hauteur, réservée à l'affichage des ressources du joueur ; une colonne de gauche occupant 12 % de la largeur, qui accueille les boutons de navigation ; un panneau de droite prenant 18 % de la largeur, dédié aux informations sur les planètes ; et enfin la zone centrale, qui constitue le reste de l'espace et affiche la carte du jeu. Ce découpage proportionnel garantit que l'interface reste lisible et fonctionnelle à toutes les résolutions, sans aucun ajustement manuel.

Affichage des ressources

La barre supérieure est prise en charge par une fonction qui affiche en temps réel les quatre ressources du joueur : l'or, l'eau, les armées et les points de recherche. Chaque ressource est rendue dans un cadre individuel contenant une icône colorée, un libellé traduit dans la langue choisie par le joueur, la valeur numérique actuelle et une mini-barre de progression qui donne une lecture visuelle rapide du niveau de la ressource par rapport à un plafond de référence. La couleur de chaque cadre est propre à sa ressource, ce qui permet d'identifier d'un coup d'œil l'état général de l'économie sans lire les chiffres.

Colonne de boutons gauche

La colonne de gauche présente une liste de boutons cliquables qui permettent au joueur d'accéder aux différentes interfaces secondaires du jeu, comme la carte, la gestion de la flotte ou l'arbre de recherche.

La fonction retourne le nom du bouton sur lequel l'utilisateur a cliqué, ce qui permet à la boucle principale de réagir sans avoir à gérer elle-même la détection. Chaque bouton distingue trois états visuels : par défaut, survolé et actif afin d'offrir un retour immédiat sur l'interface en cours d'utilisation. Le bouton actif est notamment marqué par un indicateur coloré sur son bord gauche.

Panneau d'informations droite

Le panneau de droite est l'interface principale par laquelle le joueur consulte les détails de ses planètes. Ce panneau affiche d'abord un titre de section, suivi d'une liste de lignes clé/valeur dont la couleur s'adapte au contenu : vert pour les valeurs positives, rouge pour les éléments ennemis, blanc pour les informations neutres (en théorie). En dessous, un sous-panneau défilable liste les bâtiments disponibles sur la planète sélectionnée.

Chaque bâtiment est présenté sous forme de carte individuelle indiquant son nom, son statut (déverrouillé ou non), son coût en ressources, une mini-barre représentant son niveau actuel, et un bouton d'achat dont l'état dépend de la disponibilité des ressources et du déverrouillage par la recherche. Ce panneau est conçu pour centraliser les décisions de construction sans quitter la vue de jeu principale.

Panneaux génériques et effets visuels

Trois fonctions utilitaires complètent le système d'interface. La première dessine un fond semi-transparent sombre avec une bordure fine, utilisé comme base visuelle pour chaque zone de l'écran. La seconde permet d'appliquer un dégradé progressif entre deux couleurs sur toute la hauteur de la fenêtre, ce qui améliore la lisibilité sur les arrière-plans chargés. Enfin, la troisième génère un effet de halo autour d'un élément en traçant plusieurs rectangles semi-transparents de taille

croissante ; l'intensité du halo est paramétrable, ce qui permet de l'utiliser aussi bien pour des éléments passifs que pour des effets de survol dynamiques.

Système de boutons

La fonction draw_button est le composant d'interaction le plus central du projet : elle est utilisée dans tous les menus et toutes les interfaces du jeu. Son rôle est double : afficher visuellement le bouton et détecter si l'utilisateur a cliqué dessus et elle expose un paramètre style qui permet d'adapter son rendu au contexte sans dupliquer de code.

Quatre styles ont été définis. Le style *classic* produit un bouton plein dont la couleur change au survol, adapté aux usages génériques. Le style *space*, qui est le style principal des menus, combine un remplissage semi-transparent, une bordure fine avec des coins décorés par de courtes lignes angulaires, et une barre latérale gauche colorée : il donne une esthétique science-fiction cohérente à l'ensemble des menus. Le style *better* ajoute un effet de halo pulsant autour du bouton via une fonction d'effet visuel, avec une intensité qui augmente au survol. Le style *search* est une variante sobre utilisée pour les nœuds de l'arbre de recherche.

La détection de clic repose sur une comparaison entre l'état de la souris à l'image affichée précédemment et à la l'image courante. Cette approche évite les déclenchements multiples qui surviendraient si l'on se contentait de tester si le bouton est enfoncé. Le système intègre également des retours sonores : un son de survol est joué dès que le curseur entre sur un bouton, et un son de clic est déclenché à la validation.

Ces sons sont chargés et mis en cache au démarrage.

Panneaux déroulables

Des panneaux déroulables ont également été créés. En d'autres termes, ce sont des rectangles dans lesquels il est possible d'ajouter des éléments comme des boutons, textes ou images. Si le contenu est trop grand, il est possible de faire défiler à l'aide de la molette pour afficher le contenu initialement non affiché.

Plus précisément, le fonctionnement est le suivant : on crée une zone déroulable et on y ajoute le contenu souhaité. De plus, la taille du panneau change en fonction du contenu qu'on y ajoute. Ce qui dépasse du rectangle est simplement 'coupé' comme s'il dépassait de l'écran, il n'est donc pas affiché. Lorsque l'utilisateur utilise la molette, la partie contenant les éléments du panneau est déplacée à l'intérieur de ce dernier. Ni le panneau ni les éléments ne sont déplacés en eux-mêmes mais plutôt l'endroit où ils sont affichés.

Arrière-plans animés

Champ d'étoiles

La classe SpaceBackground génère un arrière-plan animé utilisé dans les menus et pendant les parties. À l'initialisation, une méthode produit un ensemble de 260 étoiles de façon entièrement déterministe grâce à une graine aléatoire fixée, ce qui garantit que le fond est toujours identique pour une même graine mais visuellement varié. Chaque étoile est définie par sa position, sa taille, sa vitesse de défilement horizontal, et plusieurs paramètres de scintillement , phase, fréquence et alpha de base qui sont modulés à chaque rafraîchissement d'image selon une fonction sinusoïdale pour créer un effet de clignotement naturel.

En complément, une autre méthode fait apparaître périodiquement des traînées d'étoiles filantes dont la position de départ, la direction, la vitesse et la durée de vie sont tirées aléatoirement.

Chaque traînée est rendue avec une transparence croissante vers sa fin de vie pour simuler une disparition progressive. Le fond est régénéré automatiquement si la taille de la fenêtre change.

Fond de carte

Pour les phases de jeu sur la carte, une fonction charge une image de fond et l'adapte à la fenêtre courante par une mise à l'échelle proportionnelle suivie d'un recadrage centré. Le résultat de cette opération est mis en cache par résolution, de sorte que le calcul ne s'effectue qu'une seule fois par taille de fenêtre et non à chaque rafraîchissement d'image, ce qui préserve les performances.

Charte graphique des boutons

Le travail sur les boutons a été conduit en cherchant une esthétique cohérente avec l'univers du jeu, tout en restant lisible et fonctionnelle. Le style space, qui est le style dominant dans les menus, a été pensé pour évoquer une interface de terminal spatial : fond très légèrement opaque, bordure fine avec de petites marques angulaires aux coins, barre latérale gauche qui s'élargit au survol et typographie claire sur fond sombre. Ces détails sont subtils mais contribuent à l'immersion générale.

Le style better, utilisé pour les boutons d'action importants, ajoute un halo lumineux dont l'intensité augmente à l'approche du curseur, ce qui oriente naturellement l'attention du joueur vers les éléments interactifs prioritaires.

L'ensemble de ces styles a été implémenté directement dans draw_button de façon à ce qu'un simple paramètre suffise à changer complètement l'apparence d'un bouton sans toucher au reste du code.

Ecran de lancement

Une fonction prend en charge l'affichage d'une image plein écran au lancement de l'application. L'image est mise à l'échelle pour occuper toute la fenêtre, puis disparaît progressivement par fondu sur une durée configurable. Le joueur peut ignorer cette séquence à tout moment en appuyant sur la touche Échap, ce qui déclenche une disparition instantanée.

c. Histoire

Les scènes de dialogues sont imaginées par l'équipe de développement et leur implémentation en jeu se fait par le biais de fichiers textes. Ces fichiers doivent respecter une certaine mise en forme pour pouvoir être interprétés par le programme python. Durant les cinématiques, il est ainsi possible de modifier le fond d'écran, que ce soit par une couleur ou une image, de changer l'interlocuteur, le texte affiché ainsi que sa couleur.

Dès qu'un personnage parle, ses dires sont affichés sous forme de texte (figure

2) et le joueur doit interagir avec le jeu, typiquement en cliquant, pour passer à l'instruction suivante. Les instructions comprennent donc les différentes fonctionnalités citées précédemment. Seules deux d'entre elles requièrent une interaction du joueur pour continuer : l'affichage du texte de l'interlocuteur et la possible modification de la fenêtre de cinématique, permettant par exemple d'afficher un fond d'écran différent seulement après une interaction du joueur.

L'histoire créée pose le contexte et l'univers du jeu, rendant l'expérience de l'utilisateur plus immersive. Un tutoriel y est également associé afin d'améliorer la fluidité de la prise en main.

3.4 Fonctionnement du jeu

a. Intelligences Artificielles

Pour pouvoir rendre l'expérience de jeu en mode solitaire intéressante, des intelligences artificielles jouent contre le joueur en simulant un comportement humain.

Chacune d'entre elle possède trois caractéristiques avec une valeur de 1 à 5 : agressif, constructeur et chercheur. Ces caractéristiques définissent le comportement des intelligences artificielles pendant la partie. Par exemple, si sa caractéristique d'agressivité est à 5, alors l'intelligence artificielle aura tendance à plus attaquer. Chaque intelligence artificielle fonctionne par tour et objectif. Si elle n'a pas encore défini d'objectif, alors elle en choisit un selon les conditions actuelles dans la partie et ses caractéristiques en un tour.

Après ce choix, l'intelligence artificielle va avancer vers cet objectif à chacun des tours, jusqu'à qu'il soit rempli et qu'elle en choisisse un nouveau. Les trois objectifs principaux sont l'attaque d'un joueur, la construction d'un bâtiment et la recherche d'une technologie. L'intelligence artificielle choisit aussi les paramètres de l'objectif (la personne à attaquer, le bâtiment à construire) en fonction de ses caractéristiques.

Des "pirates" sont implémentés dans les deux modes de jeu. Sous forme d'intelligence artificielle, ils se déplacent de planète en planète pour attaquer les joueurs qui s'y trouvent. Chaque seconde, ils ont une chance d'attaquer le joueur propriétaire de la planète, et ainsi de lui voler un pourcentage de ses ressources. Enfin, après un temps imparti, les pirates se déplacent vers une planète voisine et recommencent ce processus.

b. Arbre de recherche

Conception du modèle de données

L'arbre de recherche repose sur deux classes.

La classe TechNode représente un nœud technologique individuel. Chaque nœud possède :

- un identifiant unique,
- un nom d'affichage,
- un numéro indiquant sa position dans la hiérarchie de l'arbre,
- des coordonnées en grille pour son placement visuel,
- un coût en ressources,
- un temps de recherche exprimé en tours,
- une liste d'identifiants de prérequis,
- une chaîne d'action à exécuter lors du déverrouillage,
- des attributs d'état suivant la progression : researched, in_progress et progress.

La classe ResearchTree regroupe un ensemble de nœuds sous un nom et une couleur d'identification.

Elle gère la logique de progression via trois méthodes : vérification de la validation des prérequis d'un nœud, déclenchement de la mise en recherche, et finalisation de la recherche, mise à jour des états des nœuds dépendants et enregistrement de l'action associée dans la file de tâches globale.

Arbres intégrés et chargement externe

Trois arbres de recherche sont fournis par défaut à l'initialisation de l'interface de recherche : un arbre Militaire, un arbre Économie et un arbre Science. Chacun comporte neuf nœuds répartis sur trois tiers de trois nœuds chacun, avec des dépendances qui obligent le joueur à progresser de façon cohérente dans chaque domaine avant d'accéder aux technologies avancées.

Pour offrir plus de flexibilité, une fonction permet d'importer un arbre personnalisé depuis un simple fichier texte. Le format repose sur des blocs délimités

par des marqueurs dans lesquels chaque attribut est défini sur une ligne distincte. Au démarrage, l'application parcourt automatiquement un dossier défini et charge tous les fichiers texte qu'il contient, ce qui permet d'enrichir ou de remplacer les arbres sans modifier le code source.

Interface d'affichage et d'interaction

Une méthode gère l'intégralité de l'interface affichée par-dessus le jeu en cours. En haut de l'interface, des onglets permettent de naviguer entre les différents arbres disponibles. Chaque onglet adopte la couleur de son arbre et est mis en évidence lorsqu'il est actif (voir figure 3).

La zone principale affiche les nœuds reliés par des lignes dont la couleur indique l'état des dépendances : vert si le prérequis est satisfait, gris sinon. Les cartes de nœuds changent également de couleur selon leur état propre : vert pour les technologies déjà recherchées, jaune pour celles en cours, bleu pour celles disponibles et gris pour celles verrouillées. Lorsqu'un nœud est sélectionné, un panneau latéral s'ouvre et présente toutes ses informations détaillées ainsi qu'un bouton de lancement de recherche, désactivé si les conditions ne sont pas réunies. Une bulle d'information contextuelle apparaît également au simple survol d'un nœud.

L'utilisateur peut faire défiler la carte de l'arbre en maintenant le clic gauche enfoncé, grâce à un système de caméra qui déplace l'ensemble du contenu. Toutes les dimensions et positions sont mises à l'échelle selon la résolution courante de la fenêtre.

c. Victoire

Conditions et vérification

Trois chemins vers la victoire sont définis, chacun représentant une stratégie de jeu distincte.
La victoire militaire s'obtient en contrôlant au moins 70 % des planètes de la carte.
La victoire économique est déclenchée lorsqu'un joueur accumule au moins 100 000 unités de ressources, en additionnant ses stocks d'or et d'eau.

La victoire scientifique récompense quant à elle le joueur qui parvient à compléter au moins quinze nœuds de recherche dans l'ensemble de ses arbres.

La vérification est effectuée à chaque tour en parcourant la liste des joueurs actifs et calculant le ratio de planètes contrôlées ainsi que la richesse accumulée. La victoire scientifique est vérifiée séparément en totalisant l'ensemble des nœuds marqués comme débloqués dans tous les arbres du joueur.

Écran de fin de partie

Une fois une condition de victoire déclenchée, un écran de fin de partie complet est affiché. Le fond reprend le champ d'étoiles animé du jeu, complété par un dégradé coloré en bas de l'écran dont la teinte varie selon le type de victoire : rouge pour le militaire, bleu pour le scientifique, doré pour l'économique (voir figure 4).
Au centre de l'écran, une icône symbolique propre à chaque type de victoire est animée avec un halo pulsant.

Le titre de la victoire s'affiche en grands caractères avec un effet de rayonnement, ou bien le mot « DÉFAITE » si le joueur n'est pas le vainqueur. En dessous figurent le nom du gagnant, une description de la condition remplie et une indication pour continuer.
L'ensemble apparaît progressivement grâce à un fondu d'entrée sur 1,2 seconde, et l'écran attend un clic ou une pression de touche avant de laisser la main à la suite du programme.

d. Multijoueur

La connexion entre plusieurs utilisateurs est implémentée. Pour réaliser cette connexion, un des joueurs doit lancer le fichier server.py qui héberge le jeu pour les autres joueurs et lui-même. La connexion se fait en LAN (Local Area Network) donc en réseau local. Les joueurs peuvent se connecter grâce à l'adresse IP du joueur qui héberge, en lançant le fichier client.py.

Le fonctionnement est le suivant : lorsqu'un joueur lance le fichier client.py (on appellera ce joueur le 'client'), l'adresse IP directement modifiée dans le fichier le met en relation avec l'hébergeur (qu'on appellera 'serveur').

Chez le serveur, un 'websocket' est créé, c'est-à-dire un protocole de communication bidirectionnelle grâce à un unique centre de contrôle. Le serveur gère tous les websockets créés, un par connexion de client. Cette communication permet donc l'envoi et la réception des messages pour le client et le serveur. Ainsi, lorsqu'un joueur envoie un message au serveur, ce dernier est en mesure de le renvoyer aux autres joueurs. Aux yeux des autres joueurs, un client est représenté par un identifiant déterminé lors de la connexion. Cela leur permet d'envoyer des messages qui contiennent les identifiants des destinataires afin que le serveur renvoie le message aux bons joueurs. Lorsque le client ferme le jeu, la connexion se termine.

Un système de salons pour le mode multijoueur a été ajouté. Ce sont des parties privées ou publiques dont le principe permet de gérer plusieurs parties en même temps. Lors du lancement du client, le joueur est par défaut dans une partie publique d'avant jeu. Ensuite, le joueur choisit entre créer une partie ou en rejoindre une déjà existante. Pour la création, il choisit un nom et peut mettre un mot de passe s'il désire créer une partie privée. Pour rejoindre, il choisit parmi la liste des parties déjà existantes ; si un mot de passe est requis, il sera demandé au joueur.

Du côté du serveur, ce fonctionnement est géré grâce à un dictionnaire. Chaque élément du dictionnaire représente une partie, avec pour clé le nom de la partie et pour valeur un tableau de différentes informations telles que le mot de passe et les joueurs présents dans ladite partie. L'ajout et la suppression d'un joueur d'une partie sont également possibles. Lorsqu'un joueur rejoint une partie, le serveur lui envoie une liste des joueurs déjà connectés et, à tous les joueurs déjà connectés, l'information qu'un nouveau joueur est arrivé. Une partie vide est automatiquement supprimée.

Lorsqu'un joueur lance « créer un salon », il en devient l'hôte. L'hôte peut choisir les paramètres de la partie et choisis quand lancer la partie. Lorsque l'hôte quitte le salon, un autre joueur le devient et hérite de ses permissions. Un joueur hôte n'as pas la même interface avant de lancer le jeu, il faut donc être capable de changer facilement entre les deux pour gérer le cas de changement d'hôte du salon.

Tous les joueurs peuvent quitter le salon avant que la partie ne soit lancé. Le joueur retourne alors dans le menu principal et peut à nouveau rejoindre/créer un autre salon, ou rejoindre le salon qu'il vient de quitter. Si le joueur qui vient de quitter est l'hôte du salon, alors un nouvel hôte est choisi parmi la liste de joueur restant. Le choix du nouvel hôte se fait de manière aléatoire. Même si l'ancien

hôte rejoint la partie, il ne redevient pas hôte.

L'hôte peut également décider de la graine de génération de la partie. Pour cela, un champ de texte apparaît dans le menu du salon, uniquement pour l'hôte. Si l'hôte ne veut pas choisir, il lui suffit de laisser le champ libre et le jeu généra une graine aléatoirement. Sinon, il doit entrer un nombre entre 10000 et 99999, sinon, le jeu refusera de lancer la partie.

e. Gestion des ressources

A chaque seconde, le serveur calcule les ressources à générer pour chacun des utilisateurs et envoie un message pour leur notifier les changements. Toutes les consommations ou générations de ressources passent par le serveur pour éviter la triche.

Toute les secondes, le serveur calcule les ressources à générer pour chacun des joueurs et envoie un message pour leurs notifier les changements. Toutes les consommations/générations de ressources passent par le serveur pour éviter la triche.

Tous les joueurs peuvent échanger des ressources entre eux par l'intermédiaire d'un menu. Pour y accéder, le joueur doit d'abord ouvrir la liste des joueurs, puis cliquer sur celui avec lequel il souhaite effectuer un échange.

Un menu présentant le joueur apparaît alors, contenant diverses informations sur ce joueur, telles que son pseudonyme ou les ressources qu'il possède. Le joueur peut alors décider d'une quantité de ressources à échanger et confirmer avec un bouton. Le serveur vérifie alors si le joueur possède suffisamment de ressources, puis procède à l'échange. La figure ci-contre représente l'interface d'échange.

Le principe de bâtiments a été ajouté. Un bâtiment est un objet présent ou non sur une planète. Chaque planète possède alors une liste de bâtiments comprenant

le nombre de chaque type de bâtiment. Les bâtiments sont déblocables via l'arbre de recherche. Un bouton de construction permet d'afficher un menu listant les bâtiments ainsi que leur nombre sur la planète sélectionnée et leur état, débloqué ou non. Chaque bâtiment génère une ressource qui lui est propre.

Les caractéristiques de chaque bâtiment sont également indiquées, à savoir son nom et son prix d'achat. De plus, ce menu permet l'achat de bâtiments : l'utilisateur peut visualiser les ressources nécessaire à cette action. Il doit ensuite confirmer son achat en cliquant sur le bouton acheter. Si le joueur possède suffisamment d'or, les bâtiments sont achetés et construits. Le prix correspondant est également retiré de la quantité d'or associée au joueur. Sinon, rien n'est fait.

f. Base de données

La base de données reliée au jeu permet de gérer les comptes utilisateurs ainsi que les ressources des joueurs. Pour le moment, certaines données sont toujours stockées côté serveur, mais elles seront progressivement migrées vers la base de données afin d'améliorer la gestion et la persistance des informations. La communication entre le jeu et la base de données est assurée par un fichier spécifique qui centralise les différentes opérations. Ce dernier contient notamment :

- Une méthode connecter, qui permet d'ouvrir la base de données et d'établir la connexion ;
- Une méthode executer_requete, utilisée pour exécuter des requêtes SQL (insertion, modification ou suppression de données) ;
- Plusieurs fonctions dédiées à la gestion des comptes (création et suppression), utilisées par l'interface utilisateur.

Cette organisation permet de séparer clairement la logique du jeu et la gestion des données, ce qui facilite la maintenance et les évolutions futures.

Interface de connexion et déconnexion :

Un système de connexion est présent dans le menu principal du jeu. Un bouton « connexion » permet d'accéder à une interface dédiée via la fonction connex-

ion_menu. Cette interface propose deux actions : créer un compte ou se connecter à un compte existant.

Les fonctions principales sont :

- connexion_login, qui permet à un utilisateur de se connecter avec ses identifiants ;
- connexion_register, qui permet de créer un nouveau compte.

Lorsque le joueur est connecté, le bouton « connexion » est automatiquement remplacé par un bouton « déconnexion ». Cela permet soit de quitter la session en cours, soit de changer de compte facilement.

Ce système rend l'expérience utilisateur plus fluide et introduit une gestion des profils persistants.

g. Système d'attaque

Un système d'attaque et de conquête de planètes a été implémenté. Les joueurs peuvent coloniser d'autres planètes en se déplaçant via les connexions existantes entre elles.

Pour lancer une colonisation, le joueur doit disposer d'au moins 30 troupes. Une fois la planète conquise, elle devient la propriété du joueur (privatisée). Elle peut cependant être reconquise par un autre joueur.

Pour reprendre une planète, un joueur doit disposer d'un avantage militaire : il doit attaquer avec au moins 30 troupes de plus que le propriétaire actuel. Ce mécanisme introduit une dimension stratégique, obligeant les joueurs à gérer leurs ressources et leurs armées avant de lancer une attaque.

3.5 Mécaniques de développement

a. Console

Le serveur possède une console avec des commandes, permettant de faciliter les tests lors de la création de nouvelles mécaniques. Lorsque le développeur appuie sur les flèches du haut ou du bas, la commande utilisée réapparaît dans la zone de texte, à la manière d'un terminal de commande. Les messages reçus par le serveur apparaissent dans la console afin de suivre les actions réalisées dans le jeu (voir figure 5).

Quelques commandes fréquemment utilisées sont :

- create client 3 : permet de lancer trois entités du jeu différentes, ou clients en même temps
- restart + : permet de relancer le serveur et les clients en prenant en compte les modifications faites au code
- give salonex all 50 : permet de donner 50 unités de chaque ressource à tous les joueurs du salon « salonex »

4 Difficultés rencontrées

4.1 Carte

a. Génération

Une problématique rencontrée lors de la génération de la carte fut de trouver un système de génération procédurale suffisamment rapide pour pouvoir générer un nombre élevé de planètes sans pour autant sacrifier la diversité de la génération, le tout le plus rapidement possible. La génération par bruit de Perlin a été la solution retenue pour correspondre à ces critères.

Pour être plus rapide, le programme n'utilise qu'une seule couche et n'utilise que quatre vecteurs aux coins des planètes.

Malgré la vitesse de la génération procédurale avec le bruit de Perlin, la génération des liens entre les planètes était encore trop lente. Jusque-là, lorsque le programme essayait de créer un lien entre deux planètes, il vérifiait si ce lien n'en croisait pas d'autre et s'il ne passait pas trop près d'une autre planète. Pour accélérer ce processus, le programme ne vérifie maintenant plus que les liens et les planètes suffisamment proches.

b. Interaction

Lors de la première version de l'agrandissement de la carte (zoom), cette interaction se faisait depuis le centre de l'écran. Cependant, l'idée était de le faire en fonction de la position de la souris. Il a fallu pour cela trouver la position de la souris sur la carte. Celle-ci étant déplaçable, il faut recalculer sa position en fonction des coordonnées de la carte par rapport à celle de la fenêtre du jeu. Il

a fallu ensuite calculer la nouvelle position des planètes après l'agrandissement. Plus la planète est loin de la souris et plus ses coordonnées sont affectées.

Le programme calcule les nouvelles coordonnées de la façon suivante :

$(x, y) = (x\_sou + (x\_pla - x\_sou) * (zoom/a\_zoom),$
$y\_sou + (y\_pla - y\_sou) * (zoom/a\_zoom))$

avec (x_sou, y_sou) les coordonnées de la souris sur la carte, (x_pla, y_pla) les coordonnées de la planète sur la carte, $zoom$ le nouveau facteur d'agrandissement et a_zoom l'ancien facteur d'agrandissement.

Cette méthode est suffisamment rapide pour permettre d'agrandir la carte sans délai, même lorsqu'elle est utilisée avec plus de 5000 planètes en même temps. C'est donc la méthode qui a été retenue.

4.2 Interface utilisateur

a. Apparence

La réalisation du fond étoilé animé a présenté une difficulté d'ordre technique liée aux performances. La première version dessinait chaque étoile sur la surface principale à chaque rafraîchissement de l'écran en créant systématiquement une nouvelle surface à transparence pour chaque cercle, afin de simuler le scintillement. Sur des machines moins puissantes ou lors d'animations chargées, ce procédé multipliait les allocations mémoire et provoquait des chutes de fréquence d'affichage visibles.

Plusieurs pistes d'amélioration ont été explorées.

La régénération des étoiles a été déplacée : elle ne se produit plus à chaque rafraîchissement, mais uniquement lorsque la taille de la fenêtre change, grâce à une comparaison entre la taille courante et la dernière taille mémorisée.

Les paramètres de scintillement (phase, fréquence, transparence de base) sont calculés une seule fois à la génération et réutilisés ensuite, seule la valeur de transparence courante étant recalculée à chaque image via une fonction sinusoïdale peu coûteuse. Ces ajustements ont permis d'obtenir une animation fluide tout en maintenant une fréquence d'affichage stable, même lorsque le fond étoilé est superposé à d'autres éléments animés comme les traînées d'étoiles filantes.

Lors du développement des panneaux déroulables, la position de la souris n'était

correctement détectée. Pour corriger cela, la position de la souris est calculée à l'intérieur du panneau selon sa position sur l'écran.

Après la création de la messagerie textuelle, les messages envoyés par les joueurs n'étaient pas reçus par les autres joueurs et leur affichage était chaotique. L'envoi et la réception des messages ont été entièrement retravaillés et l'affichage permet des messages de plusieurs lignes.

4.3 Fonctionnement du jeu

a. Intelligences Artificielles

Lors de la création des intelligences artificielles, de nombreuses problématiques se sont posées. Tout d'abord, la façon de les lancer. D'un côté, il serait plus simple de les lancer directement depuis le serveur. Ainsi les intelligences artificielles peuvent directement accéder aux informations et aux ressources côté serveur et les modifier facilement.

Cependant, cette méthode demanderait de nombreuses ressources pour faire fonctionner les intelligences artificielles côté serveur, alors que le joueur joue de son côté. Le choix final a donc été de les lancer depuis le client pour qu'il soit celui qui les fait tourner. Il a fallu donc adapter le programme pour qu'il simule celui d'un joueur, que ce soit par son comportement en jeu ou son comportement avec le serveur.

Pour éviter la triche, les intelligences artificielles n'ont pas de type de message spécial, qui pourrait être utilisé par un utilisateur mal intentionné. Les messages qu'elles utilisent sont exactement les mêmes que ceux des joueurs. Malgré les difficultés amenées par cette contrainte, cela a permis au final d'obtenir des comportements plus proches de ceux d'un humain.

b. Arbre de recherche

Isolation de la couche d'affichage par rapport au jeu

La première difficulté significative rencontrée lors du développement de l'arbre de recherche a été de s'assurer que lorsque le panneau est ouvert, le joueur ne puisse plus interagir avec la carte et les éléments de jeu situés derrière. Sans précaution particulière, les clics et les mouvements de souris destinés à l'arbre de recherche

traversaient la couche d'affichage et déclenchaient des actions non souhaitées sur la carte, comme des déplacement de flottes ou la sélection de planètes.

La solution a consisté à instaurer un mécanisme de capture des événements au niveau du panneau de recherche : tant que celui-ci est visible, il intercepte l'ensemble des entrées souris et clavier avant qu'elles ne soient transmises à la boucle de jeu principale. Cela a nécessité de revoir l'ordre de traitement des événements dans la boucle principale et d'ajouter un indicateur d'état indiquant si un panneau est actif.

Cette approche a ensuite été généralisée à tous les panneaux secondaires du jeu, ce qui a rendu la gestion des interactions bien plus prévisible et robuste.

Gestion d'une seule recherche active à la fois

Une autre difficulté liée à l'arbre de recherche concernait la contrainte qu'une seule technologie peut être en cours de développement à la fois par joueur. Sans mécanisme dédié, rien n'empêchait techniquement de lancer plusieurs recherches simultanément, ce qui aurait pu entraîner des conflits dans les états des nœuds et des incohérences dans la progression du joueur.

Résoudre ce problème a demandé de maintenir un pointeur vers le nœud actuellement en cours de recherche dans chaque arbre, et de vérifier son état avant d'autoriser le lancement d'une nouvelle recherche. Si une recherche est déjà en cours, le bouton de lancement est désactivé et un message explicatif est affiché dans le panneau de détail.

Par ailleurs, la progression de la recherche est mise à jour à chaque tour via un compteur, et le passage à l'état terminé déclenche automatiquement l'action associée au nœud. Coordonner ces transitions d'état de façon fiable, en particulier dans un contexte multijoueur où les tours sont synchronisés via le réseau, s'est révélé plus complexe que prévu initialement.

c. Système de victoire

Choix des conditions de victoire

La définition des conditions de victoire a soulevé une question de conception avant même d'être une question technique : quelles conditions rendre suffisamment accessibles pour ne pas allonger inutilement les parties, tout en étant assez exigeantes

pour constituer un vrai défi ?

Trop faciles, les victoires auraient rendu le jeu superficiel ; trop difficiles, elles auraient découragé les joueurs avant la fin.

Plusieurs pistes ont été envisagées et discutées en équipe. Les seuils retenus : 70 % des planètes possédées pour la voie militaire, 100 000 unités de ressources pour la voie économique, et 15 nœuds de recherche complétés pour la voie scientifique ont été ajustés par itérations successives lors des phases de test.

L'enjeu était de s'assurer que chaque type de victoire reste atteignable avec des stratégies de jeu différentes, sans qu'une voie ne domine systématiquement les autres.

Détection fiable des conditions en cours de partie

Une fois les conditions définies, leur détection en cours de partie a posé ses propres difficultés. La vérification doit être effectuée à chaque tour sans introduire de ralentissement perceptible, même lorsque le nombre de joueurs et de planètes est élevé.

La première version de la fonction de vérification parcourait l'ensemble des planètes et des joueurs à chaque appel, ce qui pouvait devenir coûteux sur les grandes cartes.

La solution a été d'optimiser les calculs en pré-agrégeant certaines données notamment le nombre de planètes possédées par joueur lors de la mise à jour de l'état du jeu, plutôt que de les recalculer depuis zéro à chaque vérification. La détection de la victoire scientifique a également nécessité une attention particulière, car elle dépend de l'état des arbres de recherche de chaque joueur, qui sont des structures distinctes de l'état global de la partie. Il a fallu établir un lien clair entre ces deux systèmes pour que la vérification reste cohérente.

d. Multijoueur

La possibilité de quitter un salon a été source de complications. Malgré la simplicité apparente de cette fonctionnalité, elle a en réalité amené de très nombreux problèmes et bogues. Le système des salons a été pensé de façon à économiser des échanges de données. Pour cela, le joueur ne reçoit d'information sur les salons qu'au moment du lancement, et reçoit les modifications faites tant qu'il n'a pas rejoint un salon.

Cependant, durant le temps passé dans le salon, il ne reçoit aucune information de modification, et, lorsqu'il quitte ce salon, est totalement désynchronisé avec le serveur. Il a donc fallu mettre en place une vérification lorsque le joueur quitte une partie pour qu'il soit de nouveau à jour.

Un autre problème fut celui du transfert du rôle d'hôte lorsque l'hôte actuel quitte la partie.

Les hôtes n'ayant pas le même écran que les autres joueurs, il faut pouvoir changer le menu d'attente rapidement et sans bogue. Pour cela, nous avons fusionné les deux menus en un, qui change son comportement selon une variable booléenne correspondant au fait que le joueur soit l'hôte ou non. Lorsque le joueur quitte le salon, le serveur envoie un message au joueur qui, lorsqu'il le reçoit, modifie cette variable.

4.4 Mécaniques de développement

a. Console

Cette fonctionnalité, bien qu'inutile pour les joueurs du jeu, permet au développeur de tester des situations spécifiques facilement et rapidement. Cependant, certaines commandes furent complexes à ajouter. Par exemple, la commande « restart + » permet de relancer le serveur et tous les clients déjà lancés pour prendre en compte les nouvelles modifications des fichiers.

Il faut pour cela exécuter le serveur puis arrêter l'instance actuelle du serveur, et faire en sorte que les nouveaux clients se connectent à ce serveur, tout en évitant d'arrêter le programme actuel avant de s'être assurer que le nouveau programme se soit bien lancé. La librairie « subprocess » combinée à la librairie « os » permet de faire tout cela, en créer un processus enfant qui remplace par la suite le processus parent.

Les clients sont lancés de la même façon, mais ne peuvent pas être relancés directement sur l'ordinateur du joueur.

5 Annexe

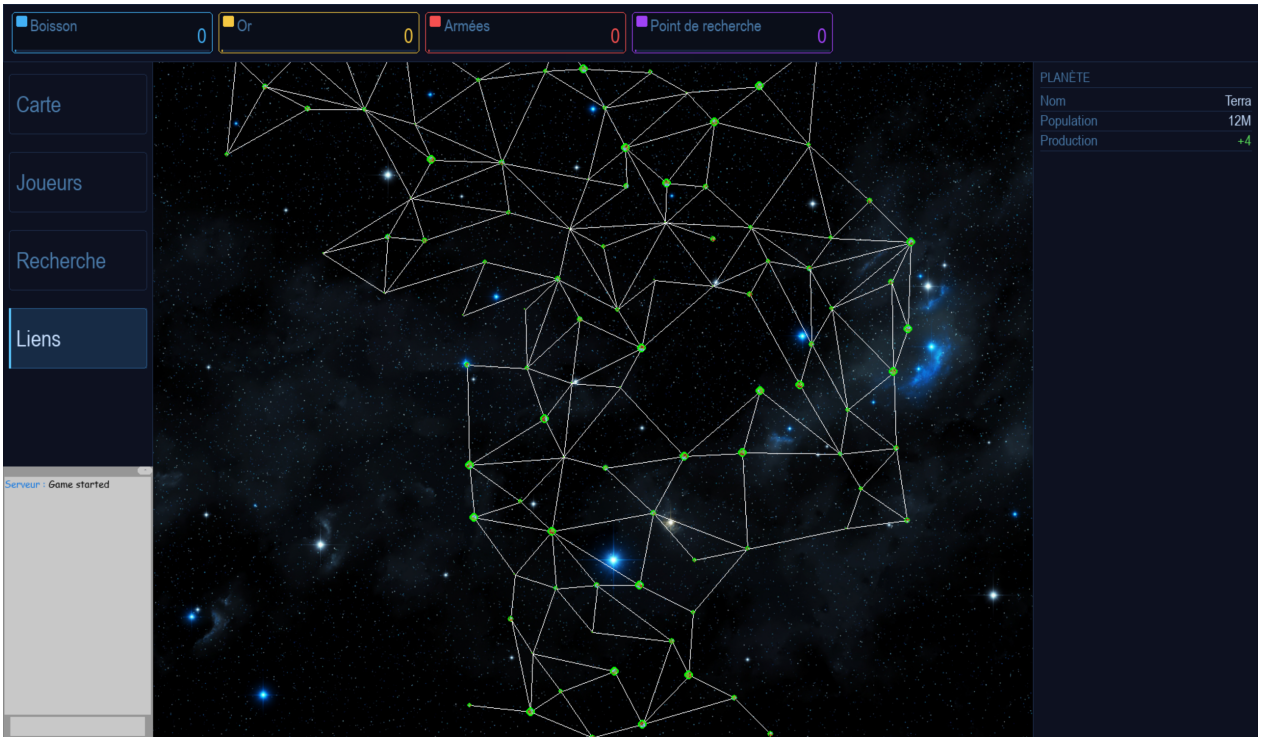


Figure 1: Interface en jeu

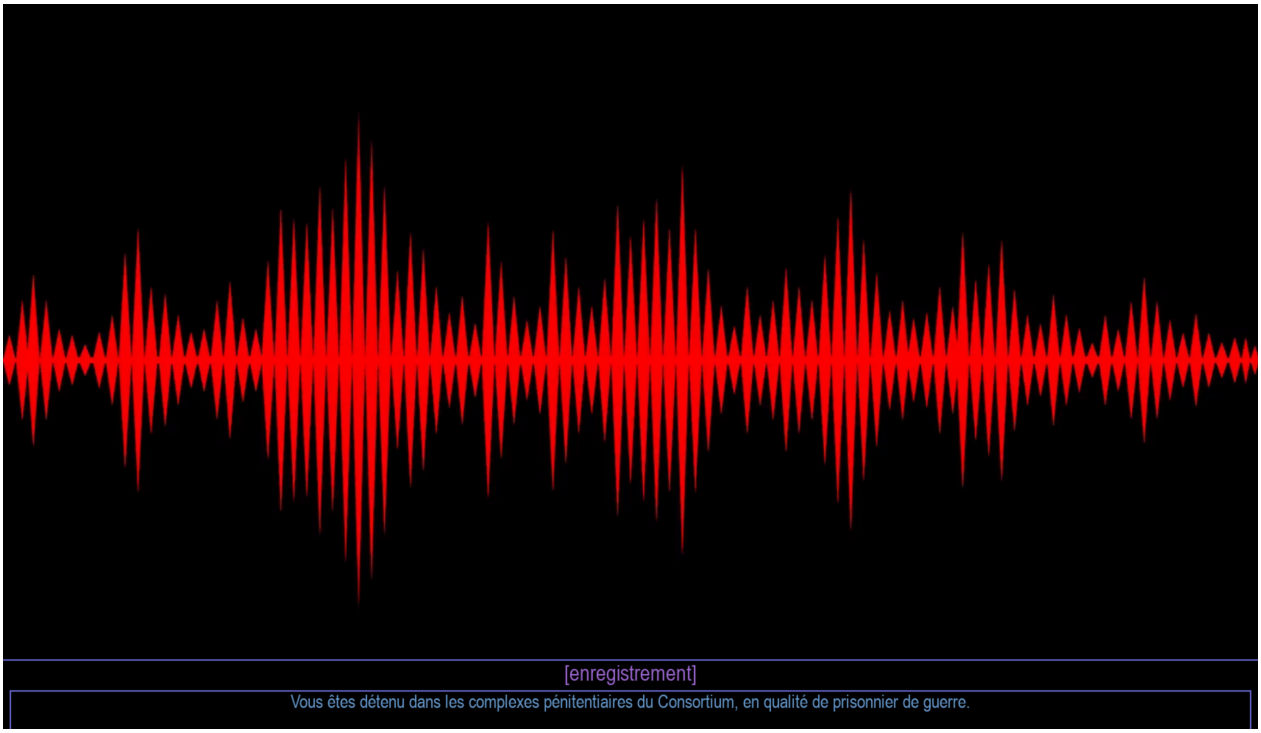


Figure 2: Boite de cinématique

Figure 3: Arbre de recherche militaire

Figure 4: Ecran de victoire économique

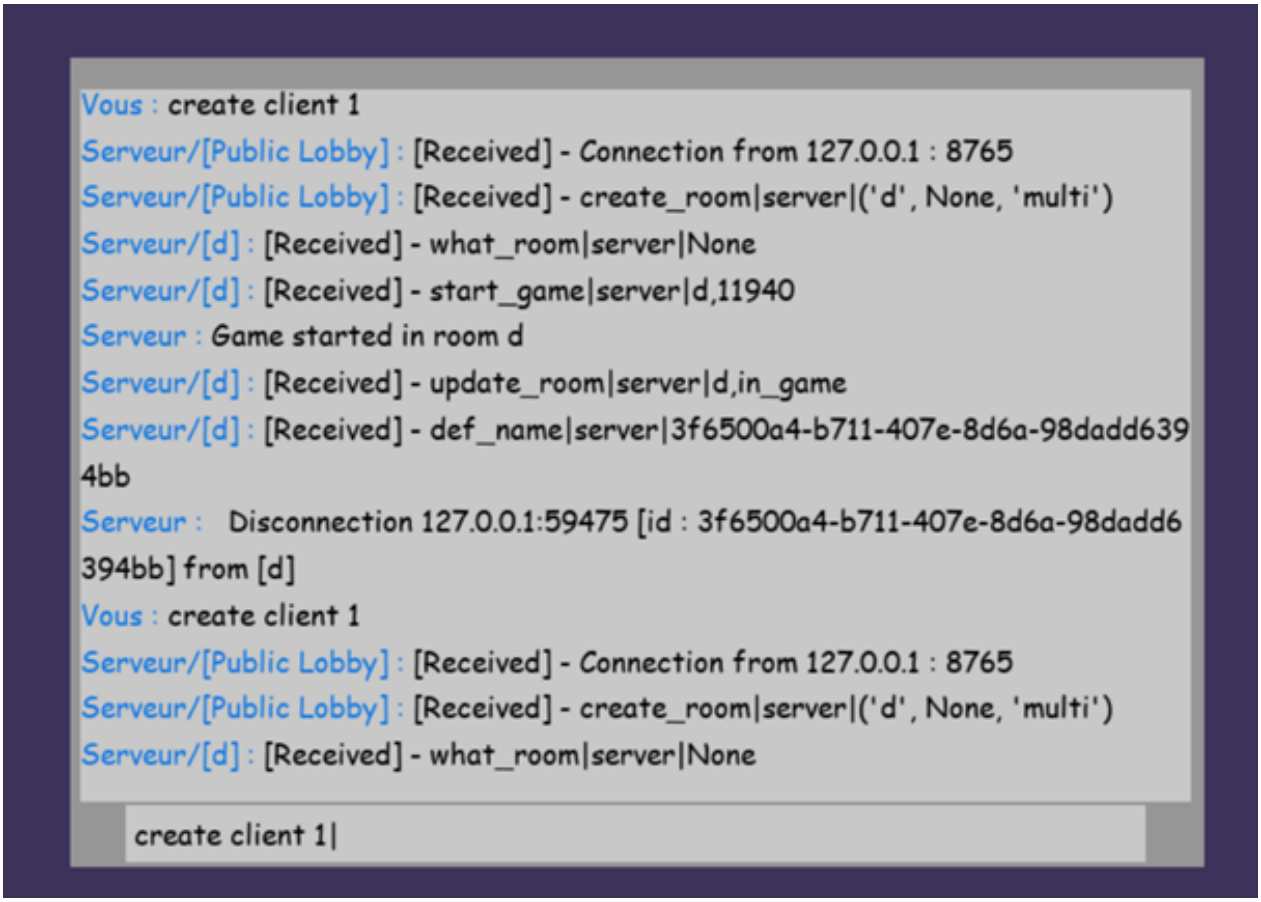


Figure 5: Console du serveur

6 Sources

opengameart.org

latex-project.org