Protocole serveur du projet Artisamine 3D

Cette documentation contient un ensemble exhaustif des éléments qui composent le protocole du jeu Artisamine3D : la façon de se connecter au serveur, la syntaxe des commandes échangées, et ce qui peut déclencher une erreur ou une déconnexion, de façon accidentelle ou non.
Le protocole d'Artisamine3D ne repose sur aucun chiffrement, et est encore en phase de test. Néanmoins, il reste cependant fonctionnel et permet à des joueurs de se connecter à un serveur de jeu depuis le client officiel, de préciser un pseudonyme, et de poser des blocs à la volée.
Artisamine3D est un jeu codé en Visual C# .NET 2022. Il exploite la plateforme Monogame, qui est une continuation non officielle de Microsoft XNA. Le serveur, quant à lui, est codé en Visual Basic .NET 2022. C'est un projet Windows Forms qui utilise la librairie socket de .NET, en mode TCP. Il pourrait fonctionner en théorie sur Mono .NET, mais je ne l'ai pas testé sur cette plateforme.

Durant cette documentation, le texte à l'intention directe de l'utilisateur sera écrit en police Times New Roman. Les commandes envoyées entre le serveur et le client seront écrites avec la police `Courier New` pour bien distinguer les commandes du texte réservé au lecteur. Dans le chapitre III, afin de distinguer les commandes envoyées par le serveur de celles envoyées par le client, les commandes côté client seront précédées de >>>, et si c'est envoyé par le serveur (et donc reçu par le client), ce sera <<<. Dans la réalité, les signes >>> et <<< ne sont pas envoyés.
Il est intéressant de noter que chaque commande est suivie d'un caractère d'échappement. Il s'agit de \n en C#, Java et en C++, de vbLf en Visual Basic, en outre, le caractère 10 en ASCII. C'est ce qui permet de séparer chaque commande au moment de la réception de données. À chaque fois qu'un packet est reçu, un tampon ajoute la commande à une liste lorsque le caractère \n est rencontré.

Table des matières

I. Fonctionnement du serveur Artisamine3D....3
a) La procédure de handshake....3
b) La procédure d'authentification du joueur....3
c) La procédure de téléchargement de la map....4
d) Apparition du joueur sur la carte....5
e) Interaction du joueur sur la carte....6
f) Challenge ping....8
g) Déconnexions et bottages....8
h) Messages d'erreur....9
II. Liste exhaustive des commandes....10
a) Les commandes de connexion....10
b) Téléchargement de la map....10
c) Interaction avec l'environnement....10
d) Autres commandes....11
e) Messages d'erreur....11
III. Interaction typique en conditions normales....13
IV. Lexique technique....15
V. Blocs intégrés....16

I. Fonctionnement du serveur Artisamine3D

Il existe une liste de commandes implémentées et interprétées par le serveur. Elles peuvent être suivies de paramètres, ou n'en avoir aucun. Au cas où une commande serait malformée ou inexistante, le serveur tout comme le client procèdent à une déconnexion immédiate, par sécurité. Ceci afin d'éviter tout dysfonctionnement et toute tentative de détournement/hack/piratage. Lors de la première connexion par socket au port spécifié, le serveur accueille la connexion et mémorise le client dans un tampon mémoire.
À ce moment-là, le client doit procéder à un « handshake », c'est-à-dire un serrage de main entre le client et le serveur pour montrer qu'ils se reconnaissent, et qu'ils voudraient « travailler ensemble » pour créer une partie en ligne, que le client désire rejoindre ce serveur dans le seul but de traiter les commandes.

a) La procédure de handshake

La toute première commande à envoyer au serveur est « `helo` », suivie du numéro de version du client en cours d'exécution. À partir de la Bêta 0.1.0, le jeu prend en charge les connexions à distance, dans le cadre du mode multijoueurs.

```
helo 0.1.2
```

Lorsque le client envoie cette commande, il annonce au serveur qu'il est un client Artisamine et qu'il souhaite rejoindre une partie en ligne. Envoyer autre chose que la commande `helo` se traduira par un kick immédiat. Même une commande connue, puisque la procédure de handshake est obligatoire et préliminaire à tout échange avec un serveur Artisamine3D.

Le serveur devrait normalement répondre par un `helo ok` pour dire que le client est désormais accepté. Mais dans le cas où une autre commande serait envoyée, une erreur 01 serait renvoyée, pour dire que la version du client et du serveur diffèrent.
Si toute autre commande est envoyée à ce moment-là, le serveur fermera aussi la connexion, en renvoyant une erreur 00 disant que la procédure de handshake n'a pas été accomplie correctement, parce qu'il faut débuter tout échange par un helo.
Soit dit en passant, la commande helo est directement inspirée du protocole SMTP qui permet, quant à lui, d'échanger des mails. Mais ceci est uniquement précisé à but didactique, puisque le protocole d'Artisamine permet de jouer en ligne avec ce jeu uniquement.

b) La procédure d'authentification du joueur

En cas de succès, le client envoie automatiquement la commande suivante :

```
nick <pseudonyme>
```

Le nom provient de « nickname » en anglais. À la place de <pseudonyme>, sans le < ni le >, il sera précisé le pseudonyme voulu par le joueur dans le cadre d'une partie en ligne. Le serveur ne mémorise pas les pseudonymes, et n'utilise pas un système de comptes. Ainsi, n'importe qui peut prendre n'importe quel pseudonyme, tant qu'il fait au minimum 1 caractère, et au maximum 16 caractères, avec uniquement des chiffres et des lettres. Aucun paramètre utilisateur n'est mémorisé par le serveur, excepté les coordonnées pendant la connexion, le pseudonyme, et quelques propriétés inhérentes à la connexion. Elles sont perdues à chaque déconnexion.
Vous l'avez probablement vu venir, mais si le pseudonyme est déjà utilisé par un autre joueur, celui qui demande le même pseudonyme est expulsé. Le serveur renverra alors une commande `error 04`. Si le pseudonyme est trop long, ou invalide, un `error 07` sera renvoyé. Dans le cas où le pseudonyme serait valide, le serveur renvoie `nick ok`.

c) La procédure de téléchargement de la map

Puisqu'Artisamine3D est un jeu en voxel, il y a une carte en 3D avec plein de blocs stockés en mémoire. Cette carte est interchangeable par l'administrateur du serveur, et fait une taille précise (64x64x64, 128x64x128, 256x64x256, 512x64x512). Elle est sauvegardée de façon périodique, et les joueurs peuvent la modifier en temps réel de façon collective (d'où l'intérêt du serveur en ligne). Le transfert de cette map fait donc l'objet d'un alignement de commandes très précis pour que le client puisse la télécharger, et la mettre en mémoire de son côté, pour pouvoir l'afficher, que le personnage se déplace dessus, et qu'il casse/construise des blocs.
Ainsi, aussitôt que le pseudonyme est confirmé, le client envoie la commande suivante, pour obtenir les propriétés de la carte ouverte :

```
mapd size
```

Le serveur renvoie les propriétés de la carte avec la réponse suivante :

```
mapd size x y z t md5
```

Il reprend, en outre, la syntaxe `mapd size`, et rajoute les paramètres suivants :

`x` = Taille en longueur de la carte (Exemple : 256)
`y` = Taille en hauteur de la carte (Toujours 64 officiellement)
`z` = Taille en profondeur de la carte (Exemple : 256)
`t` = Type de la carte (0 = île flottante, 1 = île maritime, 2 = terrain plat sur un océan)
`md5` = Hash MD5 qui représente la totalité de la map, pour vérifier si elle n'est pas corrompue, ou si le transfert n'a pas échoué. Inusité côté client, puisque la map peut être modifiée en cours de transfert, elle indiquera une donnée incorrecte.

Après avoir reçu cette commande, le client envoie immédiatement celle-ci au serveur :

```
mapd down
```

Cette commande n'est à exécuter qu'une fois, après avoir eu un pseudonyme confirmé avec succès, sinon elle entraîne un kick immédiat. En effet, cette commande déclenche le transfert de la carte entière. C'est la commande qui exige le plus de bande passante. Elle est traitée d'une façon différente côté serveur. Alors que certaines commandes peuvent être envoyées à la suite, celle-ci met en tampon tous les tronçons à envoyer en mémoire, puis les transfère à chaque cycle du serveur, et non d'un coup. Ceci afin de garantir la stabilité des échanges, et d'empêcher au serveur de geler en tentant de tout transférer d'un coup.

La carte arrive au client sous forme de commandes successives qui font transmettre toutes les couches en longueur, comme une pomme coupée en tranches. La commande `block` est utilisée. En premier paramètre, elle contient la destination du tronçon (`x`), ainsi que l'ensemble des blocs stockés en mémoire. Elle se présente sous cette forme :

```
block x b,b,b,b,b,b,…,b,b,b
```

Où x est l'abscisse du tronçon de destination. Les b représentent le code du bloc utilisé, sous forme numérique. Un bloc d'air, un bloc de roche, un bloc de sable, etc. En y et en z affichés de façon linéaire. Si on sniffe le réseau local avec un logiciel adéquat, on pourra donc voir :

```
block 0 0,0,0,0,0,…,0,0,0
block 1 0,0,0,0,0,…,0,0,0
block 2 0,0,0,0,0,…,1,1,1
```

Jusqu'à ce que x atteigne la longueur totale de la carte. Il n'y a donc aucune compression utilisée, et il se trouve que les joueurs déjà connectés peuvent modifier des blocs, alors qu'un transfert est en cours. Pour y remédier, le serveur mémorise pour chaque client connecté tous les blocs modifiés en cours de route, pour les envoyer après le transfert de la map, afin que la carte reçue par le client soit le plus fidèle possible à la carte stockée sur le serveur.

Après transfert de la carte, la commande `mapd ok` est envoyée au client pour signaler que le transfert est accompli. `block` indique juste la destination des blocs dans la mémoire, et contient la ribambelle de blocs qui sont stockés côté serveur.

d) Apparition du joueur sur la carte

Dès lors que la carte est transférée, le serveur transmet les blocs modifiés en cours de transfert et qui ont possiblement été ignorés au tout début. Il envoie pour cela une succession de commandes `setblock` avec les coordonnées et le type de bloc sous la forme de paramètres.
Cette commande sera expliquée plus bas. Elle permet non pas d'envoyer tout un tronçon d'un coup, mais de modifier la carte bloc par bloc, comme lorsqu'un joueur casse ponctuellement un bloc, ou en pose un. Des procédures anti-triche sont mises en place pour éviter toute pose excessivement rapide, ou trop lointaine par rapport au joueur.

Le joueur, lorsque la carte est transférée, a le droit de s'annoncer sur le serveur, d'apparaître en tant que joueur visible aux yeux des autres, et d'interagir avec la carte. Le client envoie donc au serveur la commande suivante :

```
spawn
```

En retour, le serveur annoncera que l'utilisateur vient de se connecter aux autres joueurs, et lui enverra les coordonnées où il devra apparaître sur la carte. La commande prendra la forme suivante :

```
spawn x y z
```

Où x, y et z sont des nombres qui représentent l'endroit où doit apparaître le joueur. En l'absence de données, les coordonnées sont normalement fixées à -1, -1, et -1 pour chaque nouveau client. Il y aura donc un changement important de position, et le client serait probablement kické.
Mais le serveur met un flag secret à true pour autoriser pour une fois tout déplacement lointain, tout comme en cas d'usage de tp. Ainsi, il aura temporairement le droit de changer de coordonnées de façon soudaine, pour apparaître au point voulu par le serveur, sans craindre d'être kické.
Une fois cette commande reçue, le serveur va envoyer la commande suivante à tous les autres joueurs connectés, afin qu'ils soient au courant que le joueur vient d'arriver, et qu'il doit être instancié côté client (mis en mémoire) avec ses coordonnées :

```
newplayer <pseudonyme> x y z true
```

Là encore, x, y et z sont les coordonnées d'apparition, les mêmes fournies par la commande spawn. Il y a aussi un pseudonyme précisé, ainsi qu'un booléen mis constamment à « true » pour indiquer que le joueur vient d'arriver. J'ai utilisé ce booléen pour distinguer les joueurs à ajouter en liste mais qui étaient déjà là, de ceux qui viennent d'arriver. De sorte que l'on ne voie pas « jean vient de se connecter au serveur » alors qu'il était déjà là.

De son côté, le client va recevoir également autant de newplayer que de joueurs déjà connectés. Sauf que le booléen sera mis à false, puisqu'il ne doit pas déclarer dans le tchat qu'ils viennent de se connecter. Cette fonctionnalité évoluera dans la prochaine version, et j'ajouterai plutôt une fonction comme playerlist pour envoyer une liste complète, au lieu de nombreuses lignes (autant qu'il pourrait y avoir de joueurs) mais ça viendra. En cas de déconnexion, le serveur envoie à tous les joueurs la commande `oldplayer <pseudonyme>` pour dire qu'il vient de se déconnecter (sans les signes <, ni >, évidemment).

e) Interaction du joueur sur la carte

À ce moment-là, le joueur est visible par les autres joueurs. Il peut se balader sur la carte, puisqu'elle est téléchargée côté client, et ses mouvements sont transférés en temps réel au serveur. Il peut casser des blocs, en poser, et apercevoir lui-même les autres joueurs. Ils apparaissent sous la forme d'images de fantômes, les mêmes que dans Game Maker. Leur nom surmonte cette sprite.

Le joueur transmet ses déplacements en temps réel par la commande `playermove`.

```
playermove x y z
```

x, y et z sont les nouvelles coordonnées en tant que nombres à virgule flottante. En gros, ce sont des nombres à virgule pour avoir une place très précise sur la carte, sinon les joueurs apparaîtront uniquement bloc par bloc, et ce serait peu réaliste, voire saccadé.
Si le joueur change de coordonnées trop vite, il est expulsé, car il serait soupçonné d'avoir un client hacké, qui lui permettrait de se déplacer très vite ou de se téléporter sans autorisation. Pareil lorsque le joueur fournit des coordonnées en dehors de la map. Il y a un kick pour ça.

Après réception de cette commande, si elle est correcte, si les coordonnées sont correctes et qu'elles ne déplacent pas le joueur de plus de 5 blocs environ d'un coup, le serveur envoie la commande suivante à tous les joueurs connectés, sauf celui en cours de déplacement (puisque le déplacement est géré côté client) :

```
playermove <pseudonyme> x y z
```

Encore une fois, les < et > ne sont pas inclus, et x, y et z correspondent aux nouvelles coordonnées du joueur. En revanche, comme vous pouvez le voir, l'orientation du joueur n'est pas encore implémentée, et donc pas transférée, comme la rotation verticale et horizontale (yaw et pitch). Ceci viendra probablement dans les futures versions.

Pour interagir avec les autres, et envoyer des messages, le joueur envoie la commande suivante :

```
chat <message>
```

Où <message> contient le message à transmettre, sans les signes <, ni >. Il doit faire minimum 1 caractère et maximum 150 caractères, en mode Twitter (ou X, si vous vivez en 2026 et plus). Il sera ensuite transmis aux autres joueurs via le serveur sous la forme suivante :

```
chat <pseudonyme> <message>
```

Encore une fois, sans les signes <, ni >. Le client va donc afficher le message dans le tchat, en affichant le pseudonyme, et le message lisible par les autres joueurs. Parfois, le serveur peut envoyer un message automatique. Il utilisera donc le pseudonyme « sys ». Il est interdit d'usage par tous les clients.

Pour poser un bloc, la commande utilisée par le serveur autant que par le client est setblock. Il est suivi de 4 paramètres : les coordonnées, suivies du type de bloc. Détruire un bloc revient à poser un bloc d'air, c'est-à-dire un bloc d'identifiant 0. La liste des blocs implémentés dans le jeu se situe au chapitre IV de ce document.

```
setblock x y z id
```

id est un entier qui représente le type du bloc. Il peut être un bloc de gazon (1), un bloc de terre (2), un bloc de pierre (3), etc. Si un bloc excède l'identifiant implémenté, il peut entraîner un kick immédiat du joueur pour tentative de hack. De même, un bloc ne peut pas avoir un identifiant inférieur à 0 (le bloc d'air).

Un bloc posé trop loin par rapport aux coordonnées du joueur entraîne un kick, puisque la portée maximale du client officiel est de 5 blocs dans toutes les directions. Tenter de poser un bloc d'eau entraîne également un kick, pour tentative d'inonder la carte. De toute façon, les mécaniques de l'eau ne sont pas encore implémentées côté serveur à ce jour.
Tenter de poser un bloc en dehors des limites de la carte entraîne aussi un kick immédiat, puisque le client officiel ne peut faire de telles actions. Non pas pour écarter les développeurs indépendants qui veulent faire des clients non officiels, mais plutôt pour éviter les tentatives de hack, ou de plantage.

f) Challenge ping

En plus de toutes ces sécurités, pour déceler les joueurs inactifs, j'ai mis en place un challenge qui permet d'envoyer une commande aux clients environ toutes les minutes. Il s'agit de la commande ping. Elle est suivie d'un hash unique, une séquences de caractères hexadécimaux générés à partir de la date + heure + nom du joueur, hashés en MD5. La commande est totalement inspirée du protocole IRC.

```
ping <hash_md5>
```

… Est donc envoyé au client de façon périodique, sans les signes <, ni >. Le client a environ une minute pour renvoyer la commande suivante :

```
pong <même_hash_md5>
```

Si aucune requête ping n'a été envoyé et que le client envoie une réponse pong sans raison, il est kické. Si le délai est expiré, il est kické. Si le hash est différent de celui envoyé par le serveur, pareil. Ceci afin d'éviter les clients frauduleux et d'économiser de la bande passante en écartant les joueurs inactifs, ou dont le socket a brutalement été fermé sans signalement.

g) Déconnexions et bottages

En effet, le client, avant de quitter la partie, doit envoyer `shut` au serveur afin de signaler qu'il s'en va. C'est à ce moment-là que la commande `oldplayer` est envoyée pour signaler une déconnexion, et que le serveur ferme le socket et retire le joueur de la liste des joueurs connectés.
Parfois, si le jeu ou le réseau plantent, le joueur ne peut pas signaler sa déconnexion. C'est pour cela que certains mécanismes, comme le ping challenge, sont mis en place.
C'est l'équivalent du heartbeat dans Minecraft, même si je ne connais pas vraiment son fonctionnement sur le bout des doigts.

N.B. : Si c'est le serveur qui envoie shut aux clients, c'est parce qu'il annonce qu'il est en cours d'arrêt. À ce moment-là, le client officiel déconnecte le socket, et affiche un message au joueur, qui précise que le serveur était en cours d'arrêt, et que la partie en ligne est finie. *Ouin !*

Le joueur peut aussi être kické, par l'administrateur, par le serveur de façon automatique, et, dans les futures versions, par des modérateurs qui auront le droit de botter les gens en dehors du serveur, en cas de faute, par exemple. La commande suivante est envoyée par le serveur :

```
kick
```

La commande peut être suivie d'une raison écrite sous forme textuelle, et apparaîtra donc sous la forme suivante :

```
kick <reason>
```

J'espère que je ne le rappelle pas, car je l'ai déjà beaucoup précisé, les caractères < et > ne sont pas envoyés, ils servent d'illustration pour dire qu'à la place, il faut indiquer une raison quelconque. Par exemple, si le joueur casse trop la carte, il pourra recevoir ça de l'administrateur :

```
kick Casse trop de blocs d'un coup
```

Le client sera expulsé, il sera déconnecté, et le client affichera la raison dans la fenêtre, dans un joli petit message sur fond terreux. Le kick peut aussi prendre la forme d'un `error 24` selon les circonstances.

h) Messages d'erreur

Le serveur ne transmet pas directement les messages d'erreur liés à des fonctionnements internes, des problèmes de connexion ou des erreurs récurrentes, également liées à des données corrompues ou des paramètres incorrects.
Généralement, quand une commande envoyée ne respecte pas le nombre de paramètres attendus, qu'il n'y en a aucun là où ils sont obligatoires, ou que des données textuelles sont envoyées à la place des chiffres, par exemple, le serveur envoie un `error 14` et déconnecte immédiatement le client à l'origine de la commande malformée.
Pareil pour le client lorsqu'il reçoit une commande côté serveur, il déconnecte immédiatement, sans en chercher l'origine. Ceci, encore une fois, afin d'éviter les dysfonctionnements, les abus ou les serveurs piégés.
La liste des messages implémentés est indiquée dans le chapitre II, petit e. Si le serveur renvoie un nombre inconnu, le client affiche un message générique de type « Erreur inconnue » dans la fenêtre.

II. Liste exhaustive des commandes

a) Les commandes de connexion

`helo` : Suivie du numéro de version du client, initie une procédure de handshake.

`nick` : Spécifie le pseudonyme voulu par le joueur. Paramètre : pseudonyme voulu.

b) Téléchargement de la map

`mapd` : Suivi de `size`, permet d'obtenir les propriétés de la carte (sa taille, son type, etc.) auprès du serveur. Celui-ci répond avec une commande identique, suivie de la taille, du type et d'un hash MD5. Cette commande est envoyée par le client. Si `mapd` est suivi de `down`, permet de télécharger la carte tronçon par tronçon. Ce transfert peut durer un certain temps. Si cette commande suivie de `ok` est envoyée par le serveur, c'est pour annoncer au client que le transfert et complet.

`block` : Représente un tronçon de la carte envoyé par le serveur. La carte n'est jamais envoyée d'un coup, seulement morceau par morceau. Un peu comme un ADN, `block` est suivi de l'abscisse puis des données des blocs présents dans ce tronçon sur une dimension. Paramètres : x, et la suite des blocs à intégrer dans le tampon mémoire qui représente les blocs du monde. Ces blocs sont à ajouter aux axes y et z subséquents au tronçon localisé avec x.

c) Interaction avec les utilisateurs et l'environnement

`spawn` : Demande au serveur où faire apparaître le joueur. Les mouvements du joueur sont gérés côté client. Le serveur renvoie ainsi la même commande, mais avec des paramètres, qui sont les suivants : x, y, z (coordonnées en 3D).

`motd` : Envoi d'un message de bienvenue à tout utilisateur qui vient de se connecter.

`setblock` : Poser un bloc avec l'identifiant voulu. Poser un bloc d'air (absence de bloc) revient à casser un bloc. Paramètres : `setblock x y z id`. Le client ne modifie jamais la map tout seul, il attend que le serveur renvoie à tout le monde, y compris le joueur, le bloc cassé ou posé. Le serveur répond ainsi, en cas de succès, avec la commande suivante : `setblock x y z id`. En cas d'échec, le serveur peut ignorer la pose, ou peut renvoyer les erreurs suivantes : `error 26` (trop de blocs posés en un coup), `error 27` (identifiant du bloc invalide), `error 28` (le bloc est en dehors des limites de la map), ou alors lorsque le bloc est posé trop loin : `error 09`.

`newplayer` : Envoyé par le serveur pour indiquer qu'un nouveau joueur arrive. Il est envoyé aux autres clients quand un client se connecte, pour qu'il soit ajouté à la liste des joueurs connectés. Une

succession de lignes est envoyée au joueur lorsqu'il vient de se connecter, et que d'autres joueurs sont déjà présents. `newplayer <pseudonyme> x y z <bool:new_or_not>`
`oldplayer` : Envoyé aux clients connectés pour signaler qu'un joueur se déconnecte. Le seul paramètre communiqué est le nom du joueur en toutes lettres. Les clients qui reçoivent cette commande vont retirer le joueur visé de la liste des joueurs connectés.

`playermove` : Envoyé par le client au serveur, pour indiquer que le personnage bouge. La syntaxe est la suivante : `playermove x y z`. Les paramètres x, y et z sont des nombres à virgule flottante. Si le serveur accepte de bouger le joueur, il envoie cette commande à tous les joueurs, excepté celui qui demande à bouger : `playermove pseudonyme x y z`, afin qu'ils réassignent une nouvelle coordonnée. Si le joueur se déplace trop vite, un `error 29` est envoyé. Si les coordonnées du joueur sont invalides, un `error 08` est envoyé au client, avant déconnexion sans préavis.

`chat` : Côté client, la commande est suivie d'un message entier. Il sera ensuite transmis à tout le monde sous la syntaxe : `chat <pseudonyme> <message>`

d) Autres commandes

`ping` : Envoyé par le serveur pour vérifier si un client est connecté. Le ping est suivi d'une suite de caractères uniques. Le client doit répondre avec pong, suivi du hash unique, pour montrer qu'il est encore en fonctionnement. Sinon, c'est le kick !

`shut` : Envoyé par le client, annonce qu'il se déconnecte. Envoyé par le serveur, annonce la fermeture de ce dernier.

`kick` : Envoyé par le serveur pour annoncer qu'il a été expulsé du serveur. Peut être suivi de la raison en paramètre.

N.B. : Toute commande inconnue, non documentée, ou malformée entraîne une expulsion immédiate du client. Si le serveur envoie une commande inconnue, malformée ou non documentée, le client se déconnectera immédiatement en précisant la nature du problème. Une expulsion automatique peut être opérée par le serveur si les commandes ne sont pas envoyées de façon synchrone, ou au mauvais moment.

e) Messages d'erreur

En cas d'erreur, les commandes suivantes pourront être envoyées au client pour signaler une erreur de transmission, une erreur dans les paramètres, une action invalide, ou une commande erronée. Certaines de ces erreurs ne sont pas envoyées par le serveur, mais font partie de la liste, puisque si le serveur commet lui-même des erreurs de transfert, certaines erreurs seront affichées côté client pour désigner ce dysfonctionnement.

error 00 : Handshake invalide
error 01 : Version différente entre le serveur et le client
error 02 : Trop de connexions simultanées
error 03 : IP bannie ou non autorisée
error 04 : Pseudonyme déjà en usage
error 05 : Carte déjà transférée
error 06 : Erreur de transmission (packet mal arrivé)
error 07 : Pseudonyme invalide (y compris utiliser "sys")
error 08 : Coordonnées du joueur invalides
error 09 : Set block hack (bloc posé hors de portée du joueur)
error 10 : Water hack (tentative d'inonder une carte en posant un bloc d'eau)
error 11 : Absent de la liste blanche
error 12 : Aucune réponse pour la requête ping
error 13 : Commande invalide
error 14 : Commande malformée (trop tôt, ou arguments invalides)
error 15 : Erreur lors du téléchargement de la carte (Hash invalide après téléchargement)
error 16 : Timeout
error 17 : Packet trop grand
error 18 : Flood détecté
error 19 : Limite de joueurs connectés atteinte.
error 20 : Déconnexion intempestive sans motif valable.
error 21 : Le serveur a envoyé une commande inconnue (uniquement côté client).
error 22 : Le serveur a été arrêté par l'administrateur.
error 23 : Le serveur n'accepte plus de connexions entrantes.
error 24 : Vous avez été kické (peut parfois se substituer à « kick »).
error 25 : Le serveur a envoyé une commande mal formée (uniquement côté client).
error 26 : Vous cassez trop de blocs à la seconde.
error 27 : Le bloc posé n'existe pas dans le jeu.
error 28 : Le bloc posé correspond à des coordonnées invalides.
error 29 : La position du joueur a évolué trop rapidement d'un coup.
error 30 : Hash de ping invalide.
error 31 : Adresse IP déjà connectée ailleurs. ← Cette erreur est inusitée
error 32 : Idle player (Le joueur est resté trop longtemps sans bouger)

N.B. : Le nombre d’erreurs pourra évoluer dans les futures versions, pour anticiper les autres scénarios pouvant susciter des erreurs.

III. Interaction typique en conditions normales

Les lignes suivantes affichent un échange typique entre le client et le serveur, avec la succession attendue par le serveur. Tout se fait du point de vue du client. >>> signifie que c'est le client qui envoie, <<< signifie que c'est le serveur qui envoie la commande. Car certaines commandes peuvent être communes aux deux, il est important de distinguer qui envoie quoi. Dans la réalité, évidemment, ni le serveur ni le client n'envoient <<< ou >>>. C'est juste pour mieux comprendre.

```
>>> helo 0.1.2
<<< helo ok
>>> nick Jean1
<<< nick ok
>>> mapd size
<<< mapd 256 64 256 1 d27a834f33dbd0cf68730752870ffbe5
>>> mapd down
<<< block 0 10,10,10,10,…,10,10,10,10
<<< block 1 3,3,3,3,3,3,…,3,3,3,3,3,3
(Etc.)
<<< block 255 0,0,0,0,…,0,0,0,0
<<< mapd ok
>>> spawn
<<< spawn 6 36 6
<<< motd Bienvenue sur le serveur !
<<< newplayer Patrick2 8.234 33.554 6.374 true
<<< newplayer Shadow1 12.563 33 8.235 true
<<< chat Patrick2 Bonjour à tous
>>> chat Bonjour les gens
<<< playermove Patrick2 9.234 33 7.532 true
<<< playermove Patrick2 10.543 33 8 true
>>> shut
```

Jusqu'à la déconnexion, avec la commande `shut`. Pour la commande `block`, volontairement tronquée avec des « … » pour des raisons de lisibilité, les identifiants des blocs à la suite sont codés de façon linéaire. La carte est un tableau de valeurs en 3D, qui accepte trois coordonnées, au risque de me répéter : x, y et z.

Pour intégrer les blocs de façon à prendre en compte le décalage de la valeur y et z, il est important de faire deux boucles for pour y et z, puisque la carte est représentée par un tableau en 3 dimensions. Le code utilisé dans mon jeu, dans la version 0.1.1, est le suivant, en C# :

```
int dest_x = Convert.ToInt32(parameters[0]);
string[] fullparcel = parameters[1].Split(",");
int x_offset = 0;

for (int y = 0; y < chunkSizeY; y++) {
   for (int z = 0; z < chunkSizeZ; z++) {
      chunk[dest_x, y, z] = Convert.ToInt32(fullparcel[x_offset]);
      x_offset++;
   }
}
```

dest_x est la destination du tronçon dans le tableau de valeurs de blocs. Le paramètre 0 est converti vers un entier. **fullparcel** contient la liste des blocs séparés par des « , » précisée en second paramètre. x_**offset** est incrémenté à chaque ligne et « convertit » les blocs stockés sur une seule dimension, vers un array en trois dimensions, aux coordonnées indiquées.

Ceci est un code fourni à titre d'exemple, le projet complet fonctionne sur plusieurs milliers de lignes de code, et ce fragment peut sembler sibyllin à quiconque n'ayant jamais touché à la programmation. Mais à tous ceux qui ont des notions, elle permet de mieux comprendre comment la liste des blocs envoyés par la commande `block` permet d'intégrer les blocs reçus sur une dimension dans un environnement en 3D.

IV. Lexique technique

Protocole : Règles et conventions régissant l'échange de données entre ordinateurs.

Serveur : logiciel permettant d’accueillir des connexions clientes opérées par un logiciel compatible avec le protocole voulu.

TCP : Protocole de transfert de données à travers Internet (Transmission Control Protocol), qui fait usage d’une vérification de chaque packet pour voir s’ils ne sont pas corrompus.

Kick : Mot anglais signifiant « botter ». Arrive lorsqu’un joueur est expulsé d’une partie en ligne un d’un salon, pour des raisons plus ou moins évidentes.

Motd : Message of the day, ou message du jour.

V. Blocs intégrés

Identifiant	Intitulé
0	Air
1	Pelouse
2	Terre
3	Roche
4	Roc terreux
5	Sable
6	Tronc
7	Feuillage
8	Eau
9	Fleur
10	Adminium
11	Parterre de fleurs
12	Planches de bois
13	Verre
14	Fleur violette
15	Fleur rouge
16	Fleur bleue
17	Bloc rouge
18	Bloc orange
19	Bloc jaune
20	Bloc vert
21	Bloc cyan
22	Bloc bleu
23	Bloc violet
24	Bloc blanc
25	Bloc en or
26	Bloc en argent
27	Lanterne
28	Balise rouge
29	Grillage
30	Bloc de Game Maker
31	Pelouse enneigée
32	Bloc de glace
33	Aiguilles de sapin
34	Bloc de neige
35	Briques

Notez bien :

Les blocs en vert ne sont pas ou plus accessibles dans l'inventaire pour des raisons fonctionnelles.

L'adminium est indestructible. Il sert à délimiter les cartes, et particulièrement à tapisser le fond de la carte.

Casser un bloc revient à placer un bloc d'air. Le client fait apparaître des particules dans ce cas-là, partant du principe que lui ou un autre joueur en est à l'origine.

L'eau est normalement inaccessible dans l'inventaire, et se répand normalement de façon rectiligne, comme dans Minecraft Classic. Un bloc se répand d'un bloc à gauche, à droite, devant, derrière, et en dessous. Pas au-dessus ni en diagonale. Le cycle de calcul a lieu toutes les 500 millisecondes environ.

Dans la version cliente, depuis la Bêta 0.1.2, appuyer sur F7 fait apparaître les blocs techniques dans l'inventaire quand on appuie sur « I ». Mais cette manœuvre est risquée, car l'eau et l'adminium posés de façon répétitive peuvent casser la map ou la rendre injouable.

Merci d’avoir lu cette documentation technique.

Elle est proposée à but pédagogique par l’auteur d’Artisamine3D.

---

https://www.monokeros.ovh/art3d/