Individu

* Planning de sa journée, programmable par le joueur
* Planning par défaut calculé une bonne fois par le système : on calcule à l’avance un maximum de ses tâches, si elles sont répétitives. Si son boulot est de couper des arbres il peut se faire la checkliste des arbres qu’il faudra qu’il coupe, calculer à l’avance ses path-findings quand la CPU a le temps, de façon à ne pas avoir à le calculer quand il est surchargé

Diversité génétique

* Construction des caractéristiques et de l’aspect de chaque individu par portrait robot
* Héritage des caractères génétiques : % de chances pour chaque caractéristique / trait que ça provienne d’un parent, ou d’un plus ancien ancêtre
* Mémorise les gênes des ancêtres
* Du coup si certains gênes rendent des individues plus lents (s’ils sont plus gros par exemple, où on un handicap important), ils pourraient tendre à ne plus être représentés, sauf à très faible proportion d’individus qui les récupèrent depuis leurs ancêtres
* Représentation graphique : construction des visages par portrait robot (à la faceyourmanga), construction des corps par portrait robot aussi et étirements pour obtenir des gens aux jambes / torses plus grands ou pas => variété dans la représentation, en partant d’éléments de base pourtant restreints aux éléments de portrait robot.

Multi-joueurs

* Monde généré procédural de taille infinie
* On pop en se mettant dans la peau d’un personnage déjà existant sur le monde (nomade / travaillant déjà chez un joueur qui n’a pas trop "donné" de ses personnages à de nouveaux joueurs), au hasard
* Les personnages nomades sont forcément pas trop loin de villages déjà existants, puisse qu’ils doivent avoir déjà été "générés"

Gameplay

* On n’est pas Dieu, mais un joueur leadeur qui se promène et gère son petit monde.

Interactions (en vrac)

Quand on parle à un minan :

- cliquer sur "va à ..."

- puis cliquer sur l'endroit où on veut qu'il aille

- attention : si c'est un endroit qu'il connaît, ça affichera "cet endroit" en tooltip, mais s'il ne connaît pas cet endroit, il affichage "vers le N/S/E/O" s'il est pas très bon pour se repérer, "N/NE/E/SE/S/SO/O/NO" s'il est un peu meilleur, voir avec un degré de précision supplémentaire s'il a une boussole.

- le tooltip indiquera également "pendant X pas", mais le bonhomme les respectera plus ou moins en fonction de son niveau d'intelligence, et plus la distance est grande, plus il risque de se tromper et de se perdre.

- on peut ainsi lui dire d'aller à un endroit plus précisément en lui indiquant d'abord un endroit qu'il connaît proche, puis la direction à prendre pendant X pas, ce qui diminue le risque de se tromper.

- on peut dire "pendant X pas" ou "jusqu'à ce que tu voies ..." et cliquer sur un objet qu'il est censé voir (des champignons par exemple, en en cliquant sur 5, ou sur 1 et en indiquant le nombre qu'il est censé voir, ou "des" pour qu'il s'arrête s'il en voie plus d'un, tout simplement).

- bien sûr le moyen le plus sûr est encore de lui dire de nous suivre et de l'emmener à l'endroit souhaité, comme ça il le connaîtra, ce sera plus clair.

Exécution

* Dans un premier temps :
* Exécution sur le serveur et envoi des informations au client, pour l’affichage
* Plusieurs instances peuvent tourner, gérant chacune une zone, pour tirer un parti efficace des processeurs multi-cœurs : chacun sa zone géographique
* A terme :
* Idéalement, quand un joueur est connecté, le jeu pourrait s’exécuter sur sa machine, en n’envoyant que le minimum d’informations de mise à jour au serveur
* Si plusieurs joueurs sont au même endroit, celui qui a la meilleure qualité réseau upload + cpu peut être automatiquement maître et gérer l’environnement pour tous
* Si personne n’est quelque part, il peut continuer à progresser en mode "résumé"

Programmation progressive :

* Déjà un monde tout blanc
* Les bonshommes peuvent être des carrés
* Et on commence par gérer leurs déplacements

Réalisation :

* Structure racine des "choses"
* Papier pour le dessin
* Boucle infinie de dessin des choses qui sont sur le papier
* Activités de base : rien (=> ennui), marcher vers un point précis
* Décision de base : en cas d'ennui, décide d'un point vers où marcher, et marcher

La suite :

* Pathfinding
* Gestion des collisions : carte de collision
* Pathfinding : contournement d'obstacles fixes
* Contournement d'obstacles mobiles (priorités ?)
* Activités
* Barre de faim
* Rechercher : se déplacer vers des directions plus vagues : points cardinaux, avec part de hasard, avec un but (genre "je cherche à manger")
* Manger
* Gameplay
* Minan contrôlé par le joueur : déplacements
* Première interaction avec une autre créature : clic sur lui, "follow me"
* Activité d'un Minan : suivre une autre créature
* Générateur
* Spread des plantes : planifier les "next spread cycles" de chaque plante
* Génération de terrain : générateur de formes vectorielles, par superposition (Perlin ?)
* Générateur de rivières

Scénario

* Basé sur un cliché : la terre étouffe sous la pollution, l'humanité cherche donc une autre planète où vivre
* Autre cliché : les envahisseurs extraterrestres qui se déguisent / "possèdent" une partie de la population en vue de la contrôler. Sauf que là les extraterrestres envahisseurs, ce seraient les humains !

Panse-bête canvas

* Animation
* requestAnimationFrame : équivaut à un waitvbl, en asynchrone
* Dessin vectoriel
* pen.lineWidth = 1 épaisseur du trait
* Pour avoir un vrai trait fin de 1px : épaisseur à 1, coordonnées + .5
* pen.lineCap = 'round' aspect des extrémités de lignes
* pen.fillStyle = '#ff0000' couleur de remplissage pour les formes pleines
* pen.strokeStyle = '#00ff00'
* pen.globalAlpha = .2 règle la transparence globale pour des prochains tracés
* pen.clearRect() vide un rectangle et le rend transparent
* pen.fillRect() trace un rectangle plein
* pen.strokeRect() trace un rectangle vide
* Contour
* pen.beginPath() à mettre avant un ensemble d’instructions de dessin (début)
* pen.fill() dessine le trajet en le remplissant (fin)
* pen.stroke() dessine le trajet en contour (fin)
* pen.closePath() termine le tracé, revient au contexte(fin)
* pen.lineJoin = 'square' aspect des jointures entre lignes successives dessinant une forme
* pen.arc() dessine un arc de cercle
* pen.line() trace une ligne
* pen.rect() trace un rectangle
* pen.moveTo() place le curseur
* pen.arcTo() trace un arc de cercle
* pen.lineTo() trace une ligne
* pen.bezierCurveTo() trace une courbe de Bézier
* pen.quadraticCurveTo() trace une courbe de Bézier quadratique
* Path2D() objet de tracé de chemin SVG ou de stockage de chemin réutilisable
* Texte
* pen.font = '10px Arial'
* pen.fillStyle = '#ff0000'
* paper.text() permet de positionner par rapport au haut du texte
* pen.text() permet de positionner par rapport à la ligne de base du texte
* Images
* img = new Image() ; img.src = '…' pour charger une image
* img.addEventListener(‘load’) pour attendre que l’image soit chargée
* pen.drawImage(img, x, y) pour dessiner l’image (marche aussi depuis un canvas)
* pen.drawImage(img, srcX, srcY, srcW, srcH, dstX, dstY, dstW, dstH) pour découper / zoomer
* Copier une partie d’image ou une image entière se fait globalement aussi rapidement
* Copier depuis un canvas ou depuis un Image se fait globalement aussi rapidement
* Copier une image de 10px ou de 100px se fait globalement aussi rapidement
* Mes tests sur RTX2080S : environ 88000 images dessinées par seconde
* Images point par point
* Img = createImageData() crée une image blanche
* Img = getImageData() récupère une image depuis le canvas
* putImageData(img) dessine une image sur le canvas
* img.data[i] = 5 : attention 4 octets pour chaque pixel, tout est linéarisé
* Images point par point optimisé
* data = new Uint32Array(new ArrayBuffer(img.data.length)) : manipulation 32 bits
* data[y \* width + x] = (alpha << 24) | (blue << 16) | (green << 8) | red : exemple stockage
* img.data.set(data) : replace les données
* Manipuler octet par octet est 38% plus lent que cette technique

<https://jsperf.com/canvas-pixel-manipulation>

<https://hacks.mozilla.org/2011/12/faster-canvas-pixel-manipulation-with-typed-arrays/>

* Couleurs
* 'black'
* '#ff0000'
* 'rgba(0, 0, 200, .5)'