SANDEV, Hristo

503

**ANALYSE SUR LE PROJET DE PROGRAMMATION DE FIN D’ANNÉE (QUARTO)**

Travail présenté à Monsieur Adam Desjardins

Dans le cadre du cours **Initiation à la science de l'informatique**

Collège Beaubois

Le 6 janvier 2020

Table des matières

Contenus

[Description générale du projet 3](#_Toc30067335)

[Analyse des parties du projet 4](#_Toc30067336)

[Interface 4](#_Toc30067337)

[Méthodes et fonctions 5](#_Toc30067338)

[Problèmes à résoudre : jeu contre l’ordinateur 6](#_Toc30067339)

[Échéancier de développement 16](#_Toc30067340)

# Description générale du projet

Mon projet vise à créer un jeu Quarto qui peut être joué contre un ordinateur. Quarto est un jeu joué sur un plateau de 4x4. Il comporte 16 pièces, chacune unique. Les pièces ont 4 caractéristiques différentes possibles. La pièce peut être grande ou petite, blanche ou noire, cercle ou carrée, trouée ou plate. Le premier à jouer va choisir la pièce que le deuxième va jouer. Après avoir placé la pièce, le deuxième va choisir la pièce que le premier va jouer et le cycle se répète. Le jeu finit quand des pièces avec au moins une caractéristique similaire sont alignées verticalement, horizontalement ou diagonalement. Le dernier à placer la pièce gagnante va gagner. Le jeu peut aussi finir si toutes les cases sont remplies et personne n’a gagné. Dans ce cas, la partie est une nulle. Je vise à faire ce jeu sur un ordinateur et créer un algorithme qui est capable de jouer de la façon la plus optimale.

Le jeu Quarto et les pièces:



Quarto est un jeu créé par Blaise Muller et distribué par Gigamic, WIKIPEDIA : <https://fr.wikipedia.org/wiki/Quarto>

# Analyse des parties du projet

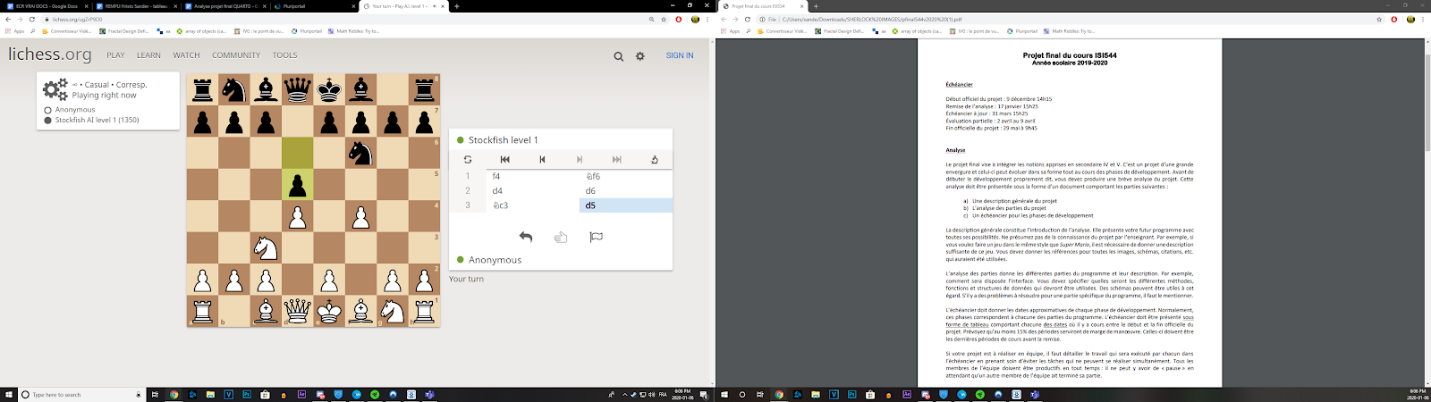
## Interface

L’interface au début va présenter l’usager avec des choix : jouer contre un humain, jouer contre un ordinateur, ou quitter le jeu. Au centre un peu vers l’haut sera placé un logo de Quarto et les options en-dessous, similaire à l’interface de Minecraft sur l’écran titre:



Minecraft est un jeu développé par le Suédois Markus Persson, alias Notch, puis par le studio de développement Mojang. WIKIPEDIA: <https://fr.wikipedia.org/wiki/Minecraft>

Une fois dans le jeu, l’usager sera présenté avec une interface qui contient au centre à gauche le jeu avec les pièces disponibles et à droite un historique des pièces jouées. L’usager aura l’option de transformer le jeu en mode 3D (BONUS). En haut à gauche, l’usager aura l’option de retourner au menu. L’interface ressemble à celle du site lichess.org lors d’une partie:



Lichess est un site-web d’échecs créé par un développeur français, Thibault Duplessis, WIKIPEDIA: <https://fr.wikipedia.org/wiki/Lichess>

## Méthodes et fonctions

Pour ce projet, je vais utiliser la programmation orientée objet. Je représente le jeu dans la mémoire par une matrice de 3 dimensions (5x4x4) qui est constituée de booléens. Le premier index va me servir de me repérer sur les coordonnées X. Le deuxième, sur les coordonnées Y et le troisième me permet d’avoir accès au 4 caractéristiques différentes, incluant la 5ième qui indique tout simplement s’il a une pièce sur la coordonnée indiquée. J’aurai une classe d’objet nommée pièce qui me permettra de mettre en mémoire les pièces que l’usager ou l’ordinateur voudront jouer. Le plateau sera un objet qui contiendra la matrice en mémoire et des fonctions.

*Fonctions et méthodes du plateau:*

* Fonction qui permet de vérifier si une case est disponible

Cette fonction aura en paramètre des coordonnées et retourna une valeur booléenne indiquant si la case est vide ou non. Cela sera accompli en vérifiant la 5ième position du dernier index de la matrice booléenne.

* Méthode qui permet de placer une pièce sur le jeu

Cette méthode aura en paramètre des coordonnées et l’objet de type pièce attribué et placera l’objet sur le plateau. Cela sera accompli en modifiant la mémoire de la matrice qui correspond au plateau.

* Fonction qui permet de recueillir les données d’une pièce sur le plateau

Cette fonction aura en paramètre des coordonnées et retournera un objet de type pièce contenant les caractéristiques de la pièce. Cela sera accompli en se référant à la mémoire de la matrice et copiant les valeurs qui correspondent à la pièce aux coordonnées indiquées.

* Fonction qui permet de vérifier s’il a un gagnant

Cette fonction vérifiera la mémoire de la matrice pour déterminer s’il a un gagnant. Cela sera accompli en vérifiant chaque dimension des 4 caractéristiques indépendantes et voir s’il a une suite verticale, horizontale ou diagonale et en retournant une valeur correspondant à l’état du jeu (rien, victoire ou perte, nulle).

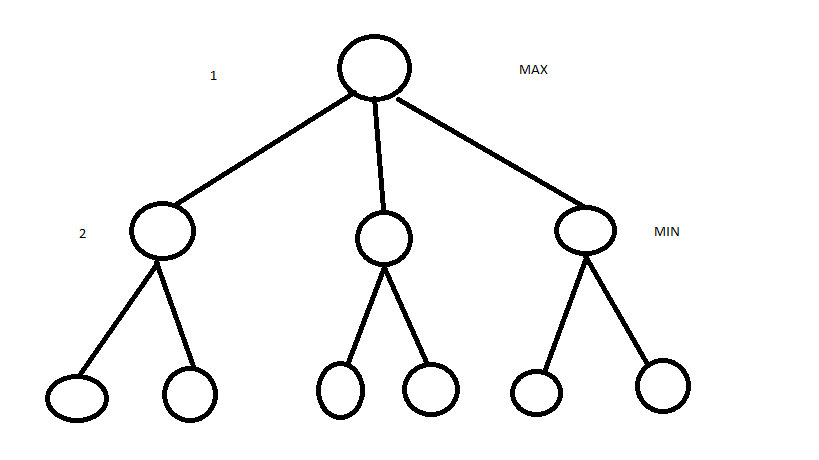
* Fonction permettant de retourner la valeur d’un mouvement (Minimax)

Cette fonction prendra en paramètre le plateau de jeu en mémoire et la possibilité qu’on est en train d’explorer pour retourner la valeur du mouvement auquel on est rendu. Cela sera accompli en utilisant un algorithme optimisé Minimax et une méthode d’évaluation de l’importance du mouvement.

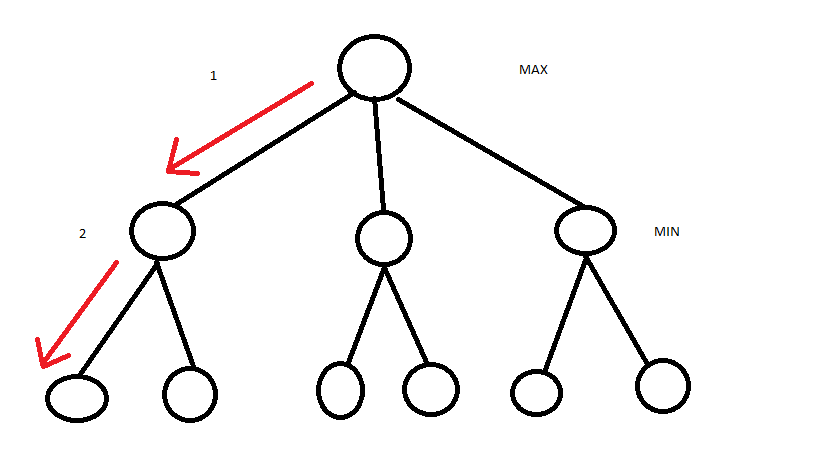
## Problèmes à résoudre : jeu contre l’ordinateur

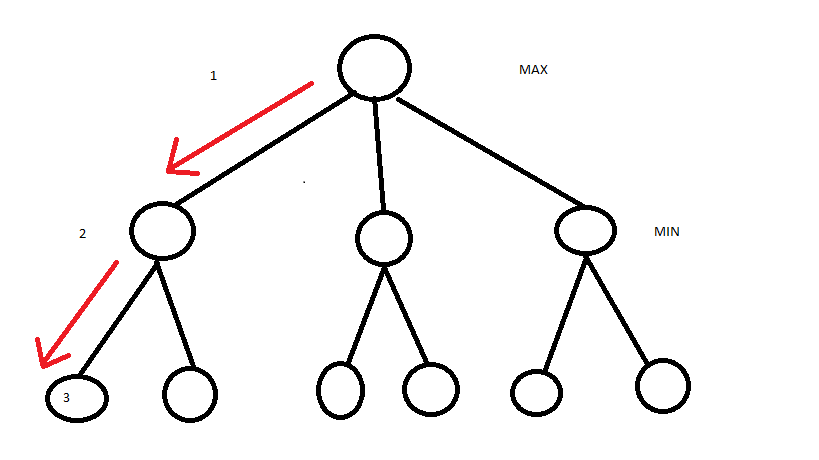
Le but de Minimax est de déterminer le meilleur mouvement possible en regardant comment ce mouvement va affecter le choix de mouvements qui déroulent de cela.

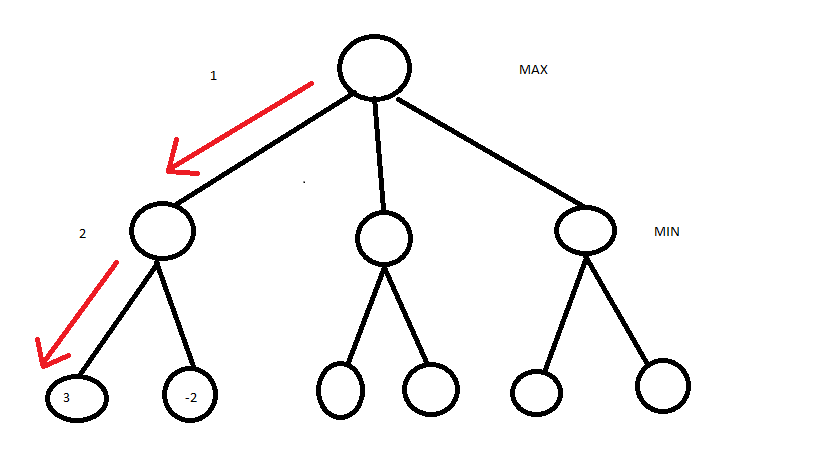
\*NOTE\* le nombre à gauche correspond à la génération (nombre de mouvements)

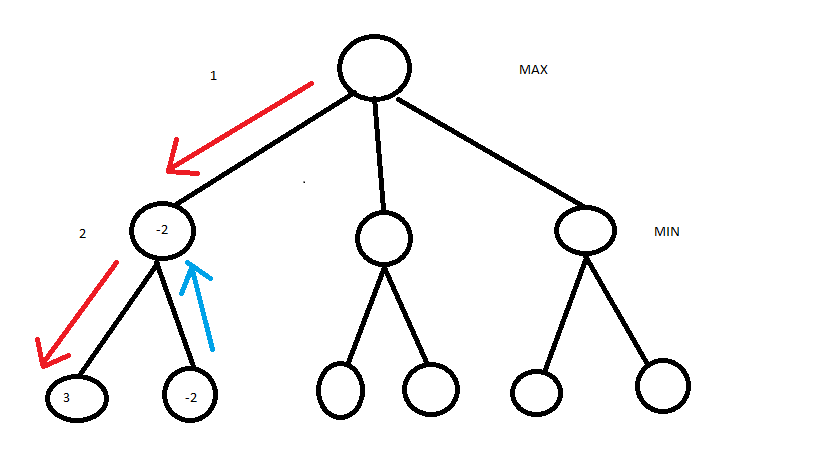


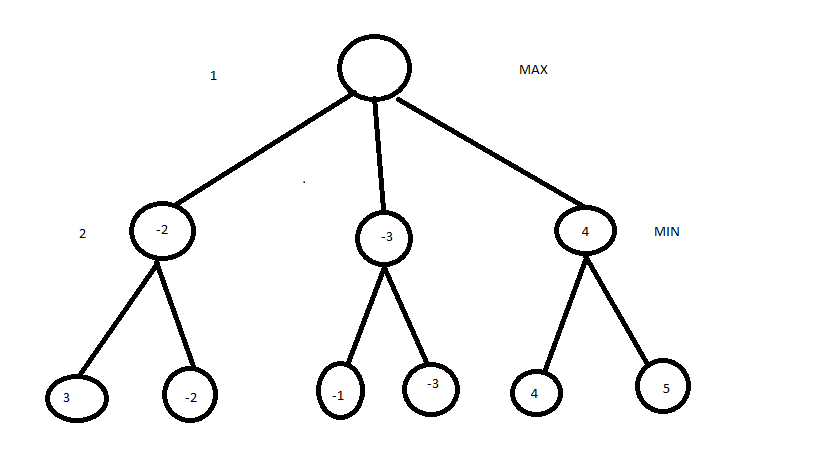
La stratégie que je vais employer sera d’essayer de maximiser le score quand c’est le tour de premier joueur à jouer et de minimiser le score quand c’est le tour au deuxième joueur à jouer. Quand un des joueurs gagne, la valeur retournée est la valeur maximale d’un Integer si c’est le premier joueur le gagnant ou la valeur minimale d’un Integer si c’est le deuxième joueur qui gagne. Mais pour faire cela, on doit décomposer le problème le plus possible : on doit atteindre la fin de l’arbre des possibilités, ce qui est beaucoup trop pour quarto. D’abord, on doit ajouter une limite. En ajoutant une limite, on doit trouver une autre façon d’évaluer la position du jeu pour déterminer qui est en faveur.



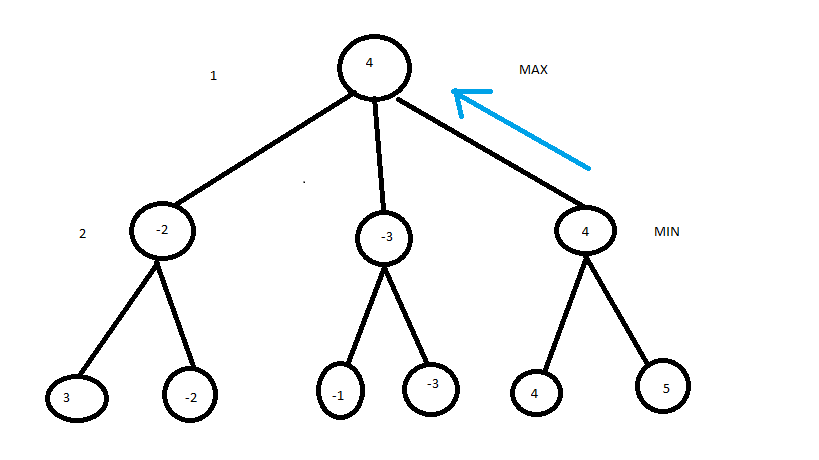
Pour la première génération, la fonction commence à explorer la première branche, mais celle-ci ne retourne pas de valeur tout de suite, d’abord elle doit à son tour chercher une valeur et le cycle se répète jusqu’à ce qu’on atteigne une valeur terminale ou on atteint la limite de la profondeur. 

On évalue cette position à être d’une valeur de 3 par exemple. Cependant, on ne peut pas la retourner tout de suite, car il reste dans le schéma encore une possibilité à explorer : la deuxième branche issue de la première possibilité de génération 2. 

On évalue cette position à être -2. Comme on veut minimiser le résultat parce que c’est le tour à l’adversaire, on va retourner la valeur qui est la plus petite. 

Le processus se répète pour les autres membres de génération 2. 

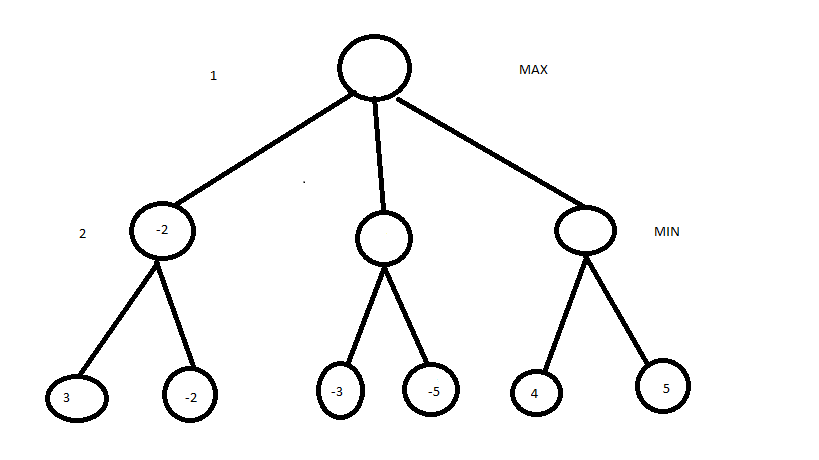
Et une fois que le processus est complété pour chaque membre de cette génération, on peut finalement maximiser le score. Comme on est rendu à choisir pour la première génération, et celle-ci maximise, on va naturellement choisir le score le plus grand des 3, ce qui correspond à 4.

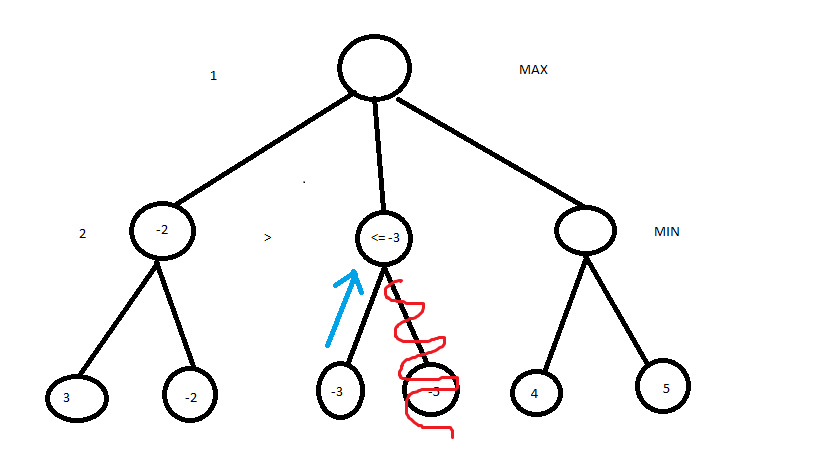


Le schéma de possibilité dans notre algorithme va commencer avec 16 membres qui s’attachent chacun à 15 membres, puis 14, 13 etc. Entre chaque membre, on doit en plus ajouter une série de d’autres membres, puisque le joueur doit aussi choisir une pièce à jouer. Finalement, il doit avoir une limite de profondeur, puisque 16! ^ 2 de possibilités prendra trop de temps pour l’ordinateur d’explorer. Si la limite correspond à *m* et le nombre de mouvements joués à *n*:

σ ≈ ((16-n)! / (16-n-m)!) ^ 2

Cependant, il a une façon d’optimiser l’algorithme Minimax. Comme le jeu Quarto est un jeu très contre-intuitif, on ne peut pas vraiment tendre des pièges au joueur. Il n’a pas de pièces qui appartiennent à un joueur comme dans les échecs par exemple. En plus, c’est notre adversaire qui choisit quelle pièce on va jouer. D’abord, dépendant du score de certains membres, on peut éliminer l’évaluation de certaines possibilités qui seront assurées d’être futiles pour la maximisation ou la minimisation du score.

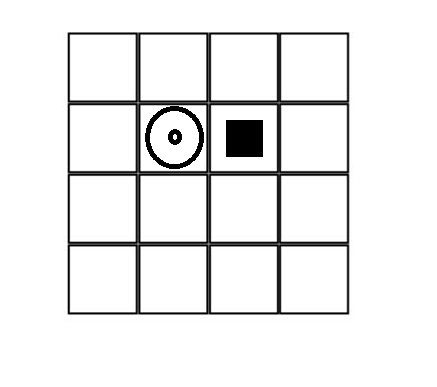


On est rendu à l’évaluation du deuxième membre de la deuxième génération. On explore la possibilité de -3 en premier, car on va de gauche à droite, mais si on a comme valeur -3, ça ne sert à rien d’explorer le -5, puisque le premier membre a une valeur de -2 et Le MAX va le piger au lieu du deuxième membre où on est sûr d’avoir -3 ou moins. D’abord, on n’a pas besoin d’évaluer le -5, car ça ne sert ultimement à rien. 

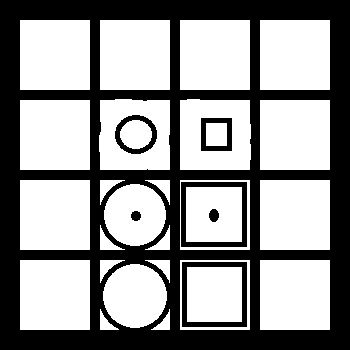
Pour la méthode d’évaluation des mouvements, on utilisera de la logique pour déterminer la faveur de la position.

Premièrement, quand le premier joueur aura gagné, la valeur du mouvement sera maximisée (valeur maximum d’un Integer) et quand le deuxième va gagner, la valeur sera minimisée (valeur opposée d’un Integer).

Deuxièmement, si on a un alignement de 2 pièces complètement opposées, on peut ignorer la ligne, car si les deux pièces sont complètement opposées, on ne peut pas faire un alignement d’une caractéristique ou plus.

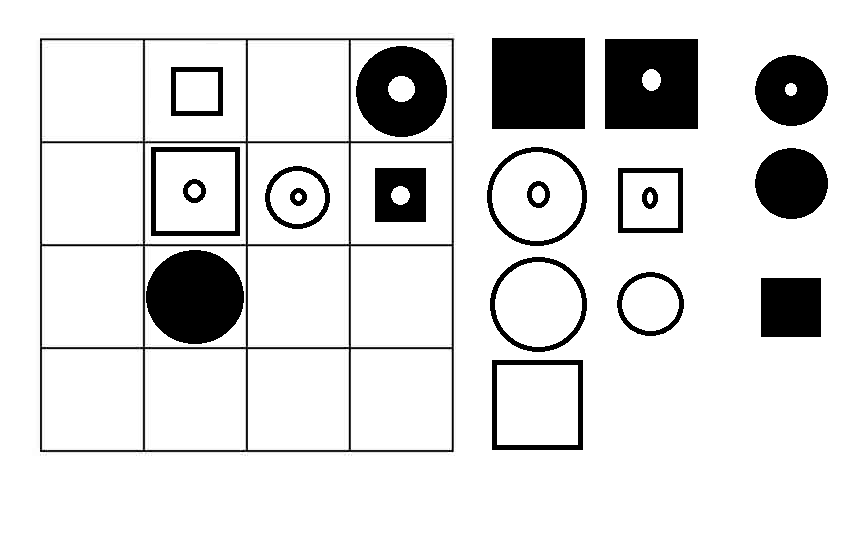


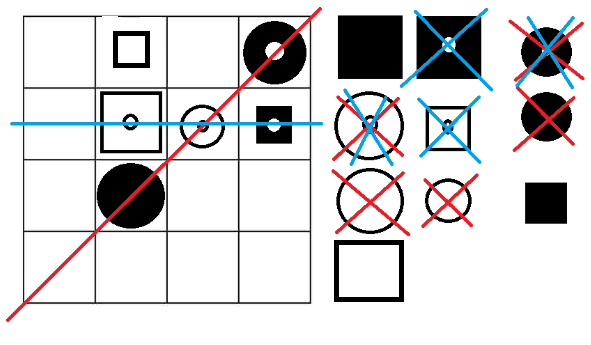
Troisièmement, si on a deux alignements de 3 pièces et que ces deux alignements sont d’une même caractéristique mais opposée, la partie est perdante pour la personne qui a placé la pièce qui permet le 2ème alignement. On peut d’abord prioritiser davantage de donnée une pièce qui va permettre cette situation à notre adversaire:



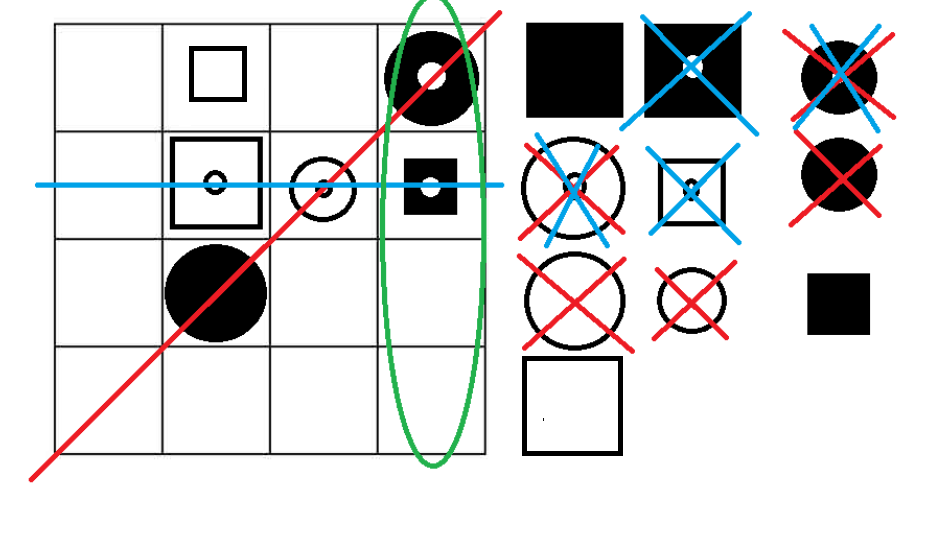
Comme on peut voir, la personne qui est la prochaine à jouer va gagner, car toutes les pièces sont des carrés ou des cercles, et il a un alignement de carrés et un autre de cercles.

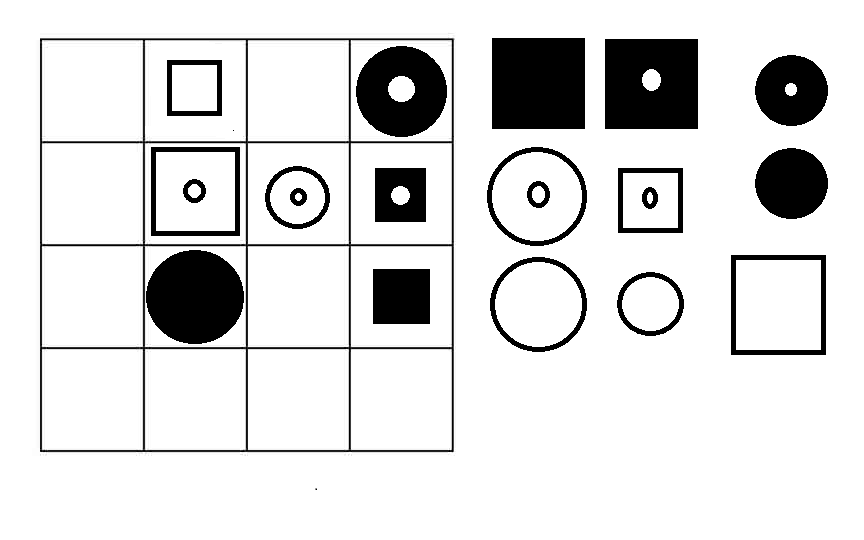
Quatrièmement, quand on joue un jeu de quarto, c’est important de ne pas donner une pièce qui sera gagnante, comme c’est nous qui donnons la pièce à jouer à notre adversaire. La meilleure stratégie est de jouer pour survivre. D’abord, quand on a 3 pièces d’une même caractéristique qui sont alignées, on veut être sûr qu’il reste un nombre impair de pièces de caractéristiques différentes quand nous devons choisir la pièce à donner. Comme ça, le joueur opposé va être forcé de nous donner une pièce gagnante s’il ne bloque pas la ligne. On peut appliquer le même procédé si on a 2 pièces d’une caractéristique similaire alignées. On peut choisir de placer stratégiquement une 3ème pièce s’il nous restera un nombre impair de pièces non-gagnantes, mais en même temps on doit s’assurer de ne pas donner un mouvement qui est gagnant. D’abord, si on a plusieurs alignements de 2 pièces et de 3 pièces qui ont une caractéristique similaire, on peut forcer notre adversaire à nous donner une pièce gagnante par élimination des pièces qui seront perdantes pour nous.

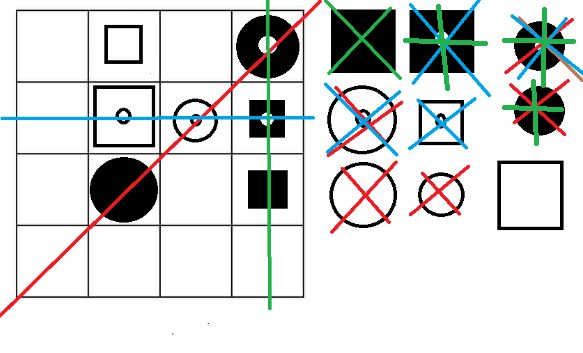


Dans cette situation, notre adversaire nous donne le petit carré noir plat à jouer. 

Par la stratégie citée en haut, on peut éliminer les pièces gagnantes. Il nous reste 3 pièces à jouer, mais en jouant une de ces 3 pièces notre adversaire sera en route de victoire puisqu’il a un nombre impair de pièces à piger, sauf si on bloque un des alignements de 3 pièces.

Cependant, en plaçant stratégiquement la pièce noire carrée, on peut éliminer une des pièces non-gagnantes en la rendant gagnante. Ainsi, en soustrayant une pièce de plus des pièces non-gagnantes, on obtient un nombre impair si on avait un nombre pair avant (on change la parité). On doit jouer la pièce qui est petite, plate, noire et carrée. On n’a pas d’alignements de 2 pièces petites seules, ni de 2 pièces plates seules, ni de 2 pièces carrées seules. Cependant, on a un alignement de 2 pièces noires tout seules. D’abord, en mettant la pièce noire sur cet alignement, on rend la pièce noire carrée grande plate qui reste une pièce gagnante.



Par élimination, il nous reste une pièce non-gagnante qu’on donne à notre adversaire (un nombre impair, ce qui est à notre avantage).

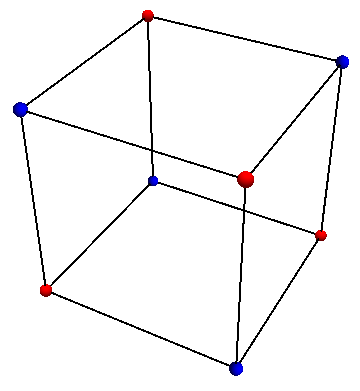
Bien sûr, notre adversaire veut bloquer un alignement pour qu’il ne puisse pas nous donner la pièce gagnante. On est forcé à lui donner le carré plat blanc grand, mais notre adversaire va essayer de bloquer d’une façon qu'il puisse obtenir un nombre impair de pièce non-gagnantes aussi. Par exemple, s’il bloque l’alignement bleu, il obtient un petit carré blanc troué seulement, un nombre impair de pièces non-gagnantes, et il va nous forcer à jouer celle-ci et bloquer un autre alignement, sinon il gagne. Cependant, en plaçant des pièces, il a aussi la possibilité de restreindre le nombre de pièces non-gagnantes. D’abord, on explore les possibilités d’une façon récursive, chaque fois le joueur essayant d’avoir un nombre impair de pièces non-gagnantes ultimement (parce qu’il a des séquences qui permettent d’obtenir un nombre plus petit de pièces non-gagnantes impair en jouant des mouvements qui donnent au début un nombre pair de possibilités). On vient juste de voir qu’on peut aussi éliminer grâce à ce procédé l’exploration des possibilités de pièce gagnantes, puisqu’elles… gagnent. Les tactiques courtes seront gérées par l’algorithme Minimax, mais la stratégie à longue terme va être gérée par les évaluations de position.

Pour le bonus qui est le jeu en 3D, j’utiliserai des notions d’algèbre linéaire (projection orthographique) et de trigonométrie (projection perspective, rotation et sélection de de cases).

MODE 3D :

Pour représenter les objets 3D, on va créer des propriétés qui contiennent les informations suivantes : les positions des coins des triangles ou carrés qui vont constituer les objets 3D et les normales de chaque triangle ou carré (c’est-à-dire le vecteur perpendiculaire). On va utiliser les coins pour tracer le contour de l’objet 3D et les normales pour déterminer quelle surface va être projetée sur l’écran.

Comme dans notre projet on pourrait seulement faire la rotation de notre objet, on va rendre la caméra statique et les coordonnées des coins des surfaces dynamiques. Comme ça, quand on va faire des changements dans notre espace trois-dimensionnel, les lignes qui unissent les points vont être plus facile à manipuler, car une ligne c’est essentiellement le trajet le plus court entre deux endroits.



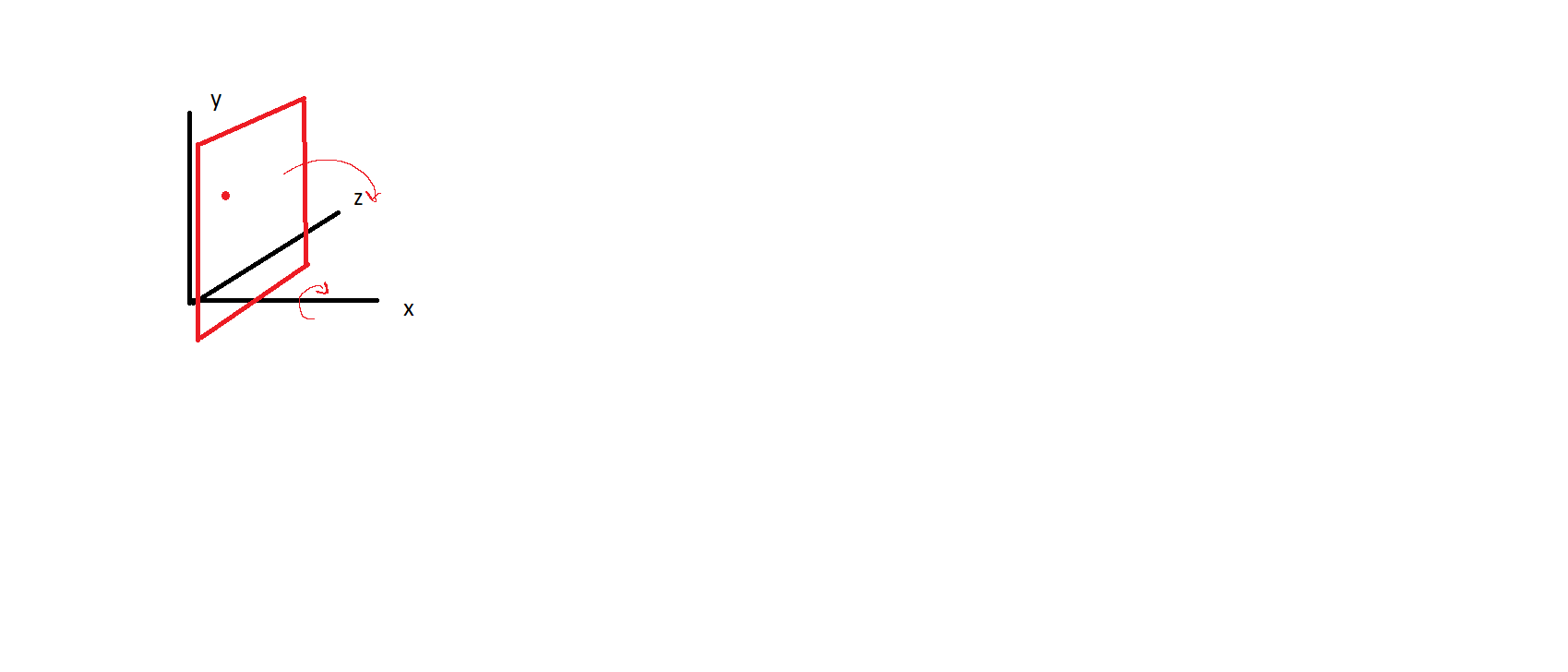
Source : <https://i.stack.imgur.com/UUFgY.png>

On veut que notre cube ou jeu face la rotation autour de lui-même, ainsi notre point de rotation sera (0,0,0). Ainsi, (0,0,0) est aussi le centre de notre plateau de jeu ou tous les coins sont à la même distance de celui-ci.

Point A : PA, Point centre : PC.

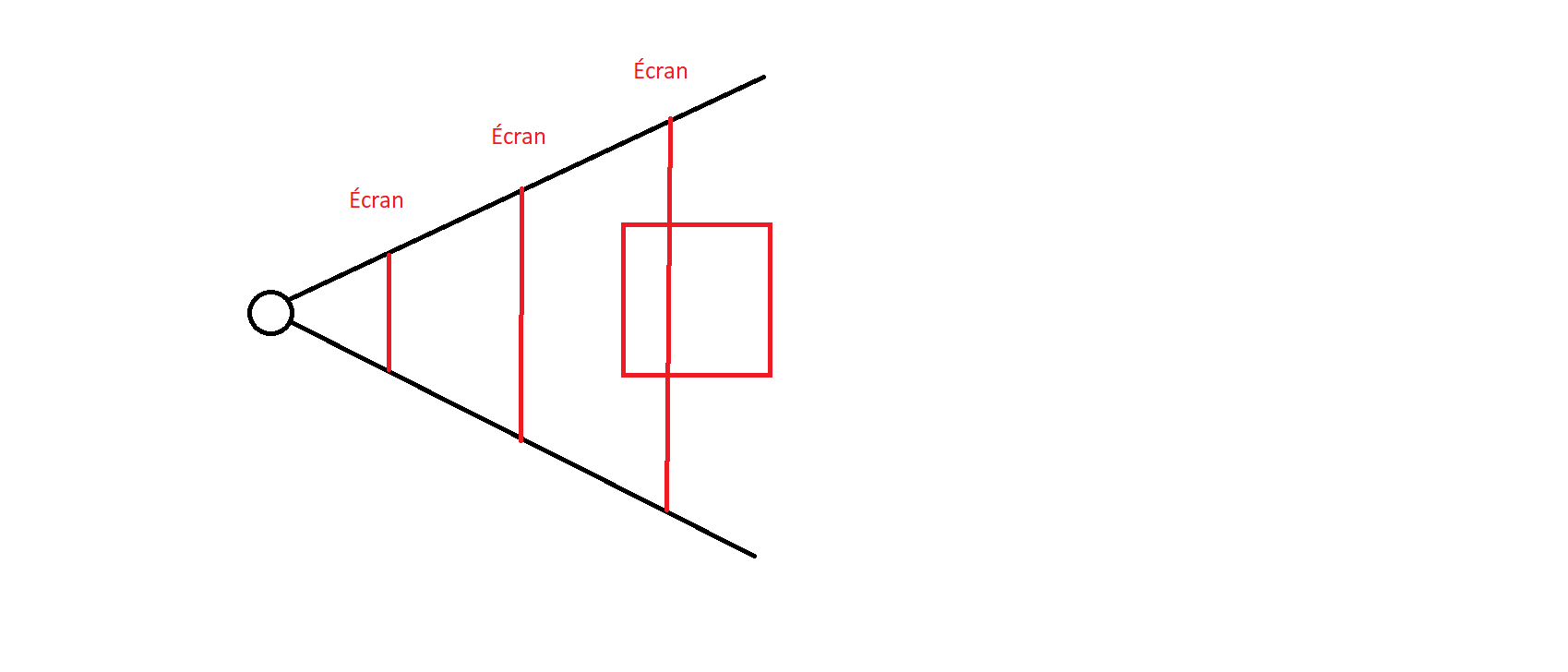
Sqrt((PCx-PAx)^2+(PCy-PAy)^2) = constante pour tous les coins.

D’abord, si on isole chaque dimension sur deux dimensions, on peut effectuer des rotations sur chacune des dimensions. Pour une effectuer une rotation sur un axe, on va créer un plan deux-dimensionnel des deux autres. Exemple :

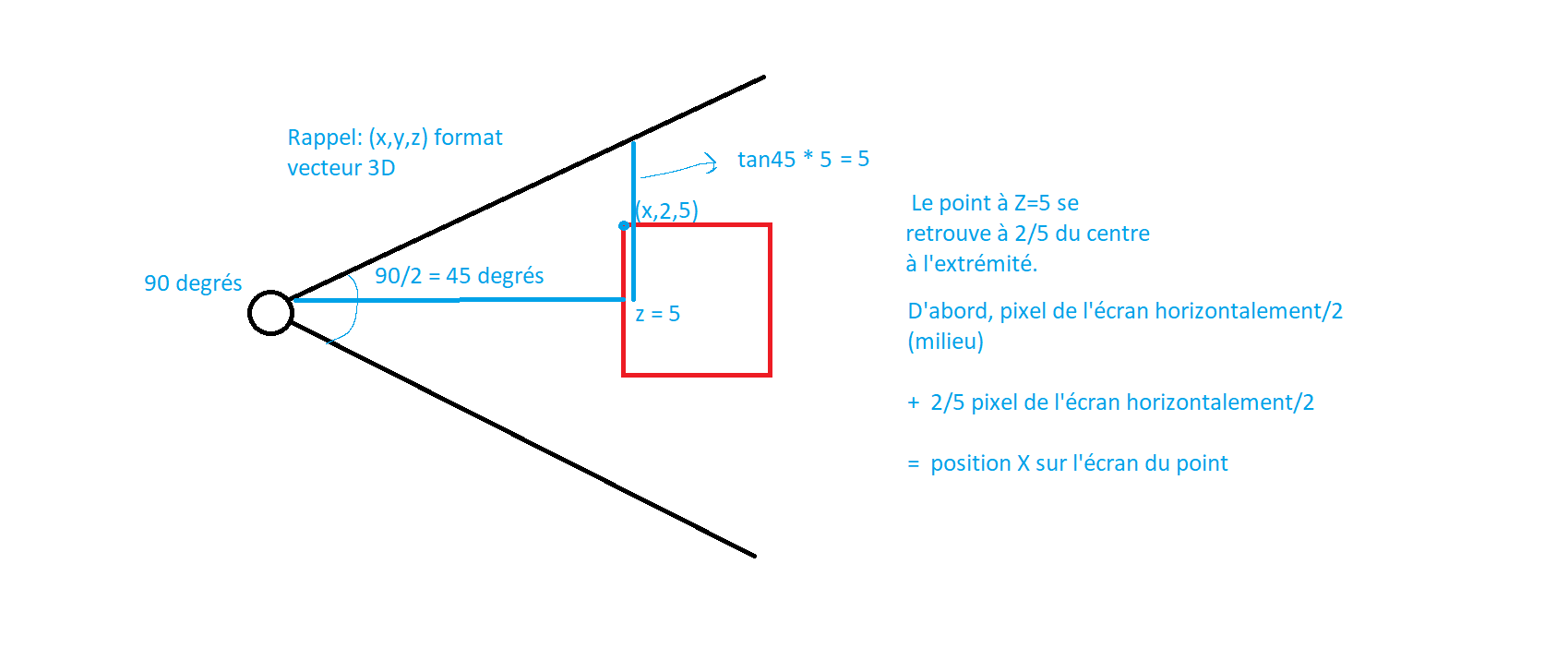


Si on veut faire une rotation sur l’axe des X, on considère le plan deux-dimensionnel Y-Z pour changer les coordonnées du point Y et Z.

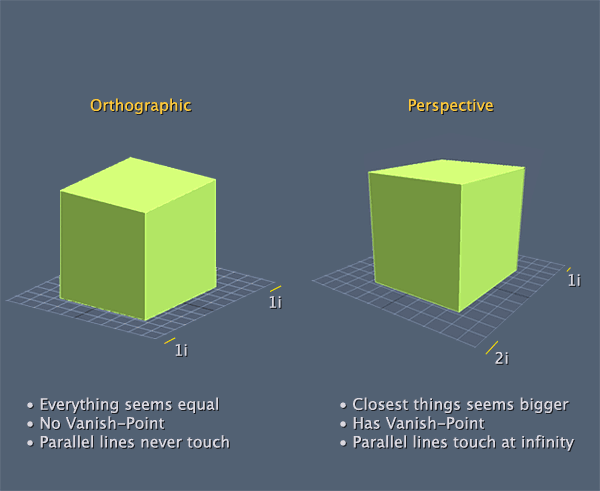
Maintenant, on réalise que la projection des objets qu’on fait n’est pas fidèle à l’œil humain, car on n’introduit pas encore un champ de vision. En effet, maintenant on verrait un cube comme un simple carré même si on se déplace. Intuitivement, on sait que si on se déplace dans une direction on va pouvoir voir l’autre côté du cube. Les objets qui sont plus loin vont se déplacer moins vite que les objets qui sont plus proche de nous.



On voit ici un champ de vision. Les extrémités du champs de vision représentent l’extrémité des choses qu’ont va voir sur l’écran, c’est pour cela que même si un cube a des surfaces isométriques, celle en-arrière va paraître plus petite. Pour bien positionner les coins sur l’écran, on va utiliser de la trigonométrie et faire des rapports :

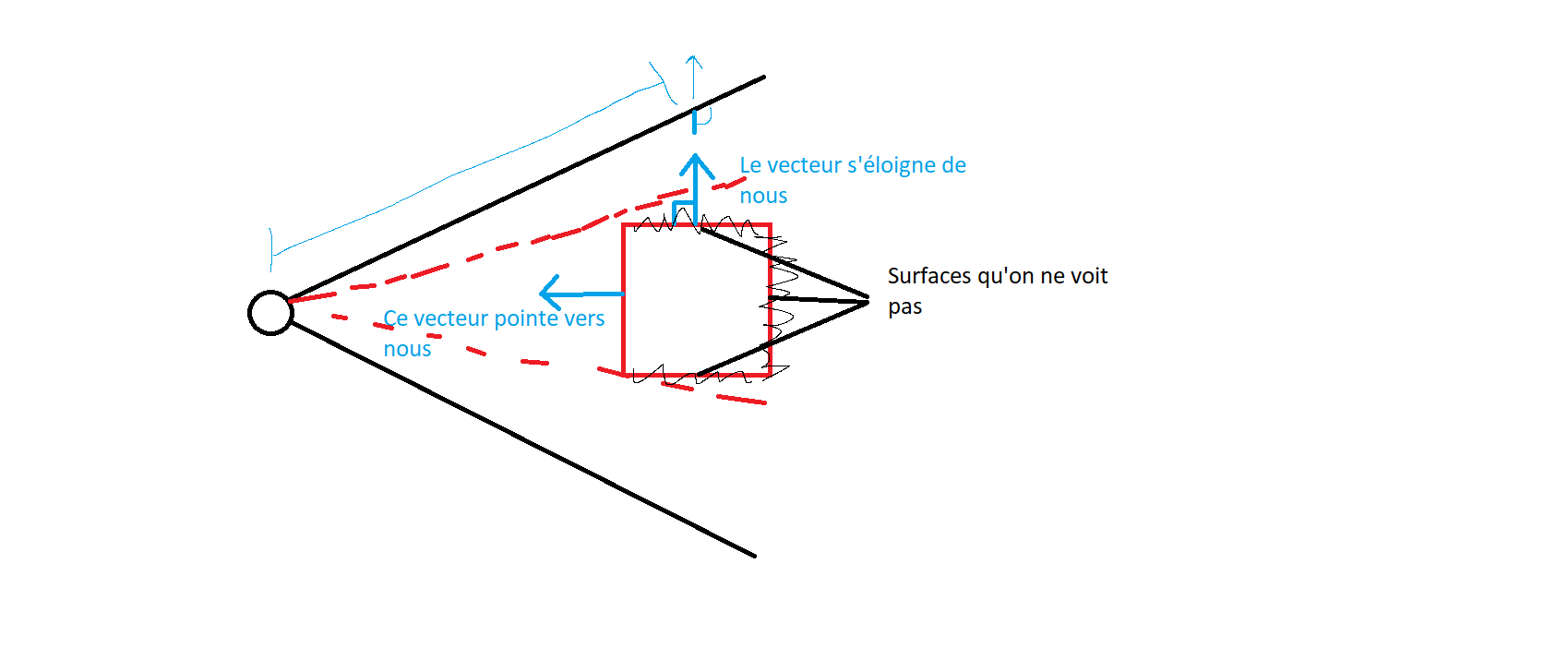


On va créer un triangle rectangle et déterminer la distance entre les deux lignes et le milieu à une certaine position Z (on utilise Tan). Cela étant, on prend la position X de notre point et on fait le rapport entre la position de l’extrémité de l’écran (distance entre la ligne de la caméra et le milieu) et la position X. Ceci va nous donner le rapport sur la ligne qui passe du milieu à l’extrémité de l’écran, comme l’extrémité de l’écran représente l’extrémité de notre champ de vision. On multiplie d’abord le rapport par la moitié de la largeur de notre écran et on additionne le milieu pour obtenir la position X sur l’écran de notre objet. On répète la même chose pour Y.



Maintenant, il nous reste à pouvoir cacher les lignes qu’on n’est pas supposé voir (le cube va paraître transparent). Pour faire cela, on doit trouver une façon efficace de dire à l’ordinateur quelle face ne doit pas apparaître et quelle face doit apparaître. Après beaucoup de réflexion, j’ai réalisé que si on fait un vecteur perpendiculaire à la surface d’un objet, on peut voir si ce vecteur se rapproche de nous (c’est-à-dire ne s’éloigne pas de nous) ou s’il s’éloigne de nous. S’il se rapproche de nous, on doit pouvoir voir la surface.

Exemple :



J’ai fait un peu de recherche sur YouTube pour voir s’il avait une façon efficace de trouver le vecteur perpendiculaire (la normale). J’ai trouvé une vidéo de 3blue1brown qui explique cela parfaitement :

<https://www.youtube.com/watch?v=eu6i7WJeinw>

La formule pour trouver le ‘’cross-product’’ de deux vecteurs est :

* Cx = Ay\*Bz – Az\*By
* Cy = Az\*Bx – Ax\*Bz
* Cz = Ax\*By – Ay\*Bx

Maintenant, on doit juste voir si le vecteur pointe vers nous. Pour faire, on va utiliser quelque chose qu’on a déjà appris : le produit vectoriel. En 3 dimensions, on doit seulement ajouter la multiplication des deux Z. On sait que si c’est perpendiculaire, le résultat est 0. Si ça s’éloigne, le résultat est plus grand que 0 comme le vecteur se rapproche à pointer vers la même direction et le même sens que le vecteur de l’extrémité du champ de vision et si c’est plus petit que 0, le vecteur pointe plus vers nous.

Comme on va avoir plusieurs pièces sur le plateau, on ne veut pas que les pièces se ‘’croisent’’. C’est-à-dire, on ne veut pas voir une pièce à travers une autre. Pour faire cela, je vais simplement dessiner les pièces en arrière en premier. Je n’ai pas trouvé une meilleure solution à cela. Aussi, j’ai réalisé que je peux aussi illuminer mes objets. Si la normale de ma surface pointe vers la source de lumière, la surface sera plus illuminée. C’est intuitif.

Aussi, j’ai changé mon plan de match : au début, je voulais créer mes propres objets sur Xojo. J’ai réalisé cependant que c’est du travail peu stimulant et très répétitif. D’abord, j’ai décidé de créer mes propres pièces de Quarto sur Blender, les exporter sous un format obj., convertir ce format obj. en txt. et lire l’information de cela pour créer mes objets 3D.

Ajustements : je vais trier mes triangles avec tri fusion, ordre de grandeur n \* log n.

Réduire le nombre de vertices pour l’illusion d’un cercle pour optimiser mon algorithme.

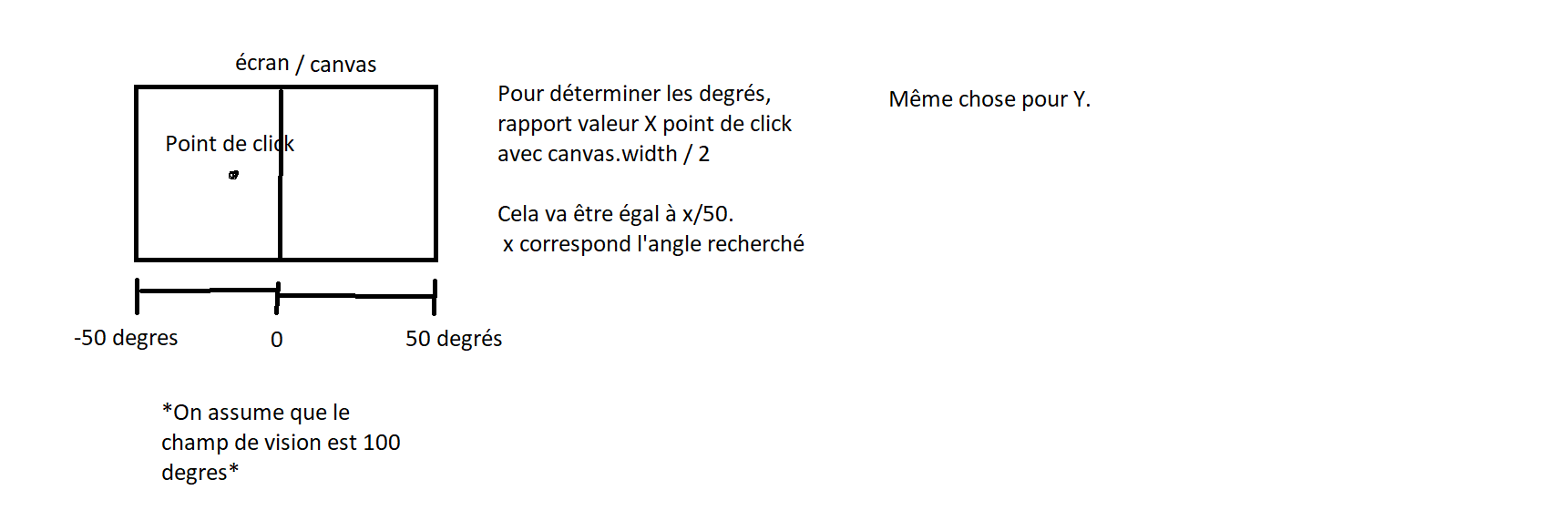
Calculer les normales de nouveau après une rotation, plus vite qu’applique la trigonométrie.

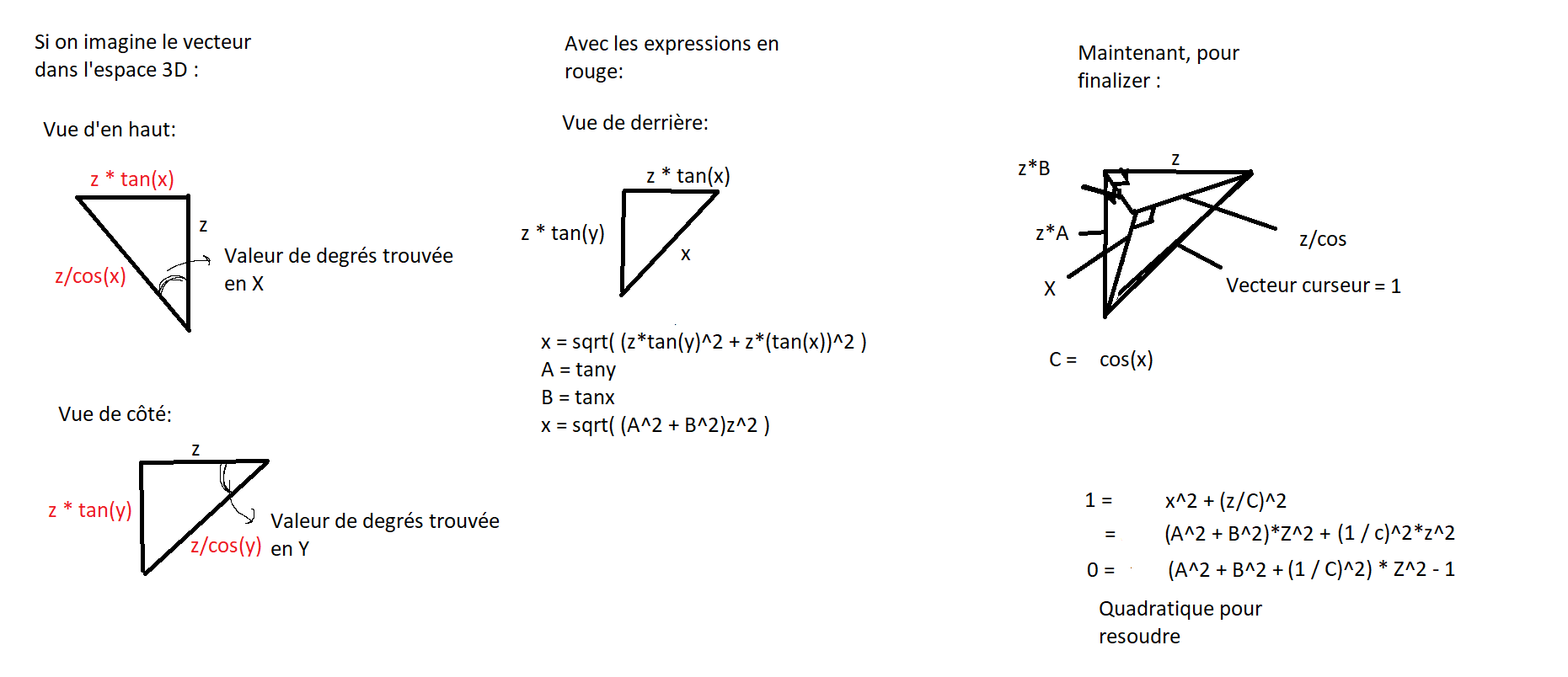
Chaque case sur le plateau va être un objet.

Remplacer ray casting pour sélectionner les objets en espace 3D par ma propre méthode :

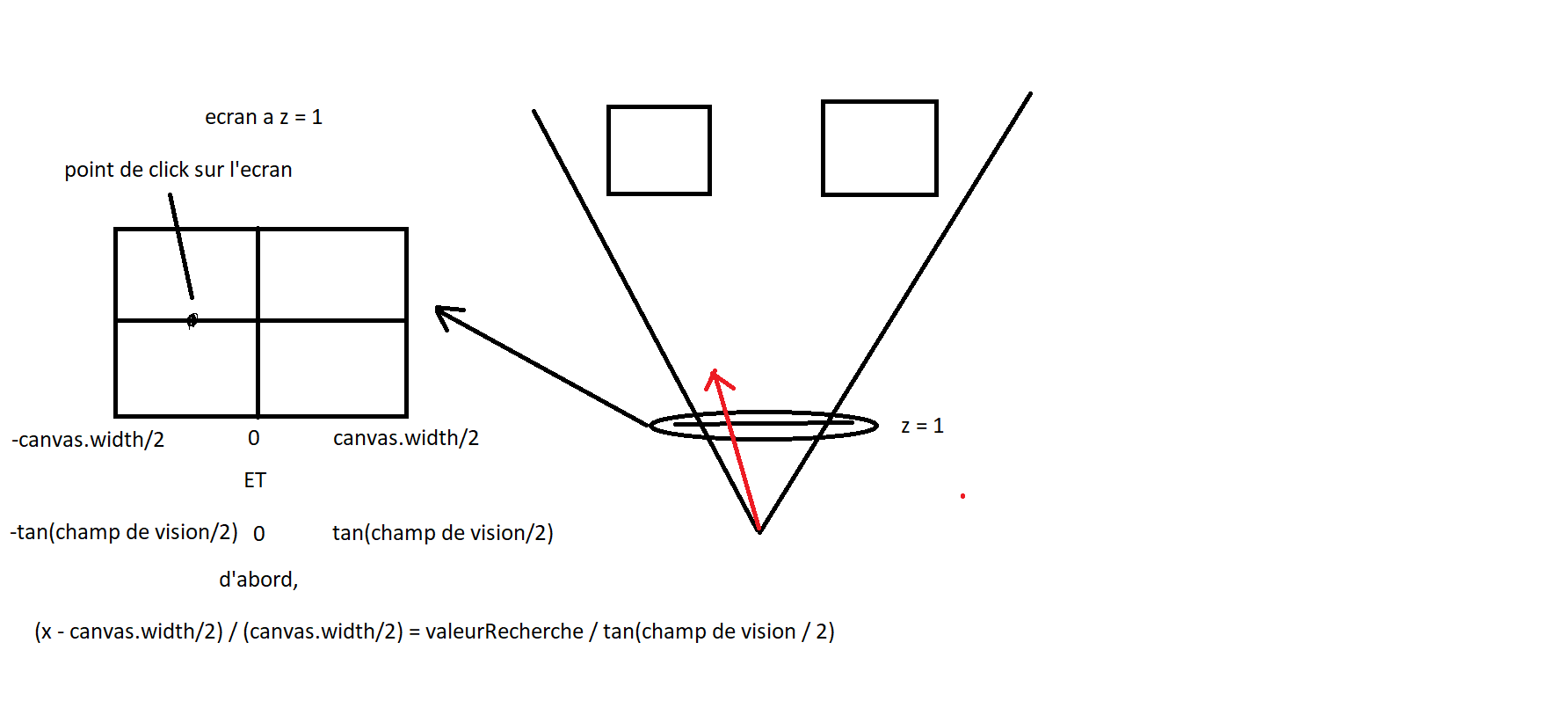
J’ai eu une superbe idée pour grandement optimiser la sélection de cases. Au lieu de faire du ray casting, je vais créer un vecteur avec une norme de 1 qui pointe dans la direction sur laquelle j’ai cliqué sur mon écran faire le produit vectoriel un par un avec des vecteurs qui relie notre position au centre de l’objet et donc la norme est réajusté à 1.

Pour déterminer dans quelle direction je click :





NOUVELLE METHODE : (raison : approximation de l’angle est mauvaise)

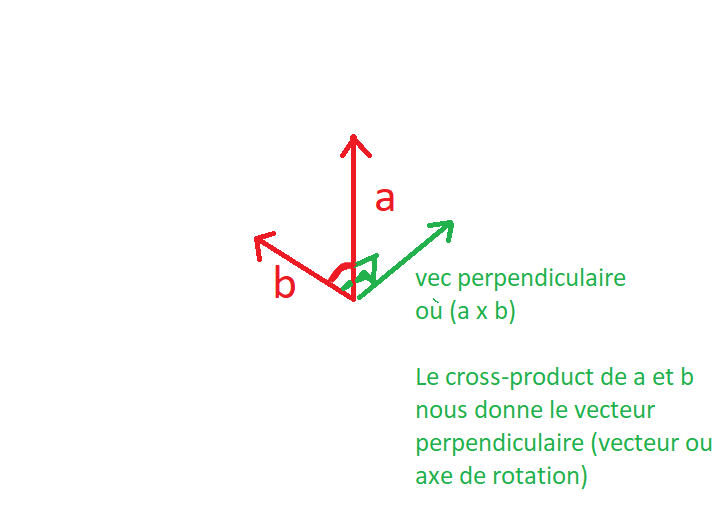
On va assumer que z = 1 déjà et on fait la proportion entre X et le bout de l’écran avec la valeur qu’on recherche et tan(champs de vision/2) :

Pour trouver les vecteurs qui pointent vers les objets dans notre espace 3D, on va simplement soustraire leur position de notre position. Pour ajuster la norme à 1, on va diviser les composantes du vecteur (X, Y et Z) par la norme trouvée à l’aide du théorème de Pythagore.

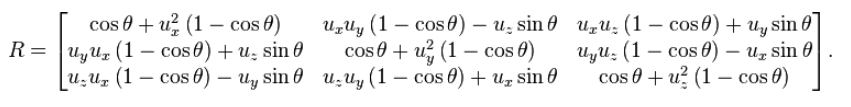
Maintenant, on va déterminer quel vecteur qui pointe vers les cubes est le plus similaire à notre vecteur qu’on a créé avec le click de la sourie. Pour faire cela, on va faire le produit vectoriel du vecteur de la sourie et de chacun des vecteurs qui pointent vers les cubes, Celui qui aura le plus grand résultat va être le plus similaire. (Intuitif, car des vecteurs perpendiculaires ont un produit vectoriel de 0, des vecteurs opposés ont un produit vectoriel négatif, d’abord vecteur qui ont le même sens et direction auront le plus grand produit vectoriel, assumant que dans chacun de ces cas les normes étaient les même).

Finalement, on doit trouver l’orientation de l’objet lors de son placement sur le plateau. Pour faire cela, on va prendre les deux vecteurs qui font un triangle sur la surface du plateau et on va prendre leur cross-product. On obtient ainsi un vecteur perpendiculaire à la surface du plateau.

Maintenant, on va créer un autre vecteur (a) de composant (0,1,0) qui représente l’orientation de nos objets lorsqu’ils sont importés par défaut. On vient juste de trouver le vecteur perpendiculaire au plateau ou la normale définit par a x b : (b). Si on peut créer un autre vecteur perpendiculaire à ces deux, on peut imaginer que les 2 vecteurs a et b sont situés sur un même plan et qu’avec le vecteur perpendiculaire à ces deux on peut effectuer une rotation d’angle X pour arriver du vecteur a au vecteur b (par exemple, on peut bouger l’axe des y et l’axe des X avec une rotation autour de l’axe des z, ou z est perpendiculaire à x et y.)



On peut facilement trouver l’angle entre a et b avec la loi des cosinus, puisque on peut trouver la longueur de a, de b et de ab puisqu’on connait les composantes. Ainsi, on peut appliquer la formule de rotation de Rodriguez avec l’angle trouver pour partir de a et aller à b. On fait passer chaque vertex par cette matrice de rotation.



u représente le vecteur autour duquel on effectue la rotation et thêta représente l’angle de rotation qu’on a trouvé. Une fois qu’on a effectuer la transformation, on peut change ajouter le vecteur du point milieu au vecteur de nos vertices et ajouter le vecteur de la normale du plateau pour que les surfaces des objets ne touchent pas au plateau (pour ne pas causer un comportement bizarroïde).