Dobble et Polys-Dobbles

Pour ceux qui vont s'ennuyer au début : accrochez vous la fin est pleine de belles maths et info j'ai une belle incompréhension des isométries du cube de dimension 4

Qu’est Dobble ?

Avant de lire ce qui suit allez jeter un œil à « Dobble » avec un moteur de recherche internet, vous y verrez des images de cartes à jouer (rondes) pleines de dessins variés

Vous avez des exemples de cartes de biDobble (page 7 ou 8) de cet article

Ce jeu est un ‘jeu familial’ idéal avec de nombreuses variantes de règles adapté à 2, 3 … 10 joueurs, Avec le fichiers joints vous n'avez plus qu'à les imprimer, il est encore plus simple d’ acheter « Dobble » un … mais quand on est informathicien on ne peut refréner le besoin de comprendre

Les diverses règles de ces jeux utilisent toutes le même fait combinatoire : deux (quelconques) des cartes ont toujours un (et un seul) dessin commun (ce 1 sera bien sur appelé à varier)

Le problème est : comment répartir un Stock de S images par Paquets de P d’entre elles sur C Cartes de telles sorte que l’unicité de dessin commun soit réalisée

Il y a un article de Wikipedia consacré à ce jeu : quand je l’ai lu la première fois les explications étaient remplacées par « pour comprendre il faudrait connaître espaces projectifs et corps finis », c’était mon cas et je n’ai pas compris

Depuis est apparue la référence :

↑ Plans projectifs, arithmétique modulaire et Dobble [archive], M. Deléglise, 2013.

Les cartes sont des droites d’un plan projectif fini, ce qui assure l’unique intersection

Des idées voisines sont développées vers la fin : dans le jeu de Trebble ce sont 3 cartes qui ont

unn dessin commun, Quabble … 4, dans d'autres variantes on aura 3,4 … choses en commun

C’est toujours possible, mais ...

Mes C cartes étant numérotées de 1 à C, je met comme dessins sur la carte i toutes les paires $\{i,j\}_{\{j \text{ différent de } i\}}$ ainsi les cartes p et q auront (exactement} {p,q} en commun

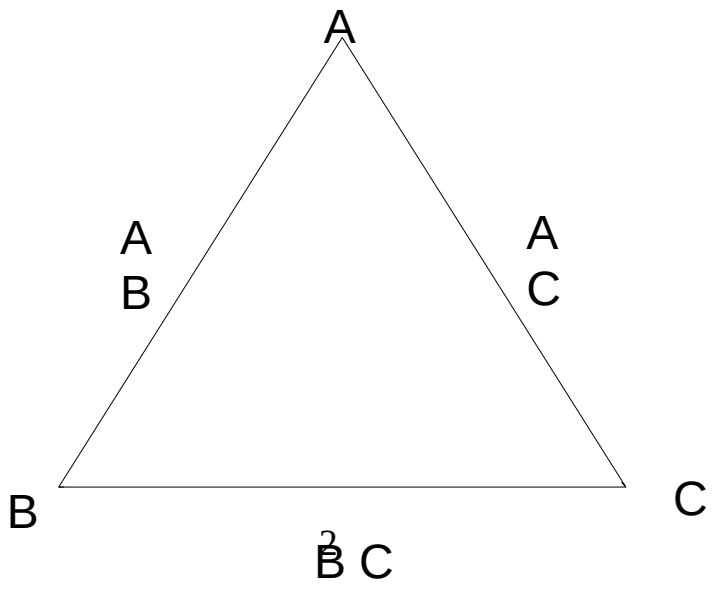
Un moyen de le ‘voir’ est : coupez la terre en tous sens (une mappemonde suffira) par des plans passant par son centre : cela vous donne des ‘grands cercles’, deux tels grands cercles auront toujours deux points communs (diamétralement opposés) ces grands cercles sont les cartes, les ‘doubles points’ les dessins

Si vous voulez faire cela avec un jeu de 52 cartes cela fera utiliser (52 . 51)/2= 1326 dessins, et chaque carte en aura 51 … tant qu’on y est au lieu des cartes rondes et de dessins on peut utiliser des pages de tableur entre lesquelles on aura à chercher LA coïncidence, cela n’a plus aucun intérêt ni aucun rapport avec un jeu

On va donc imposer une contrainte floue : on veut PEU de dessins au total et PEU de dessins par cartes

Pour commencer : des cas simples

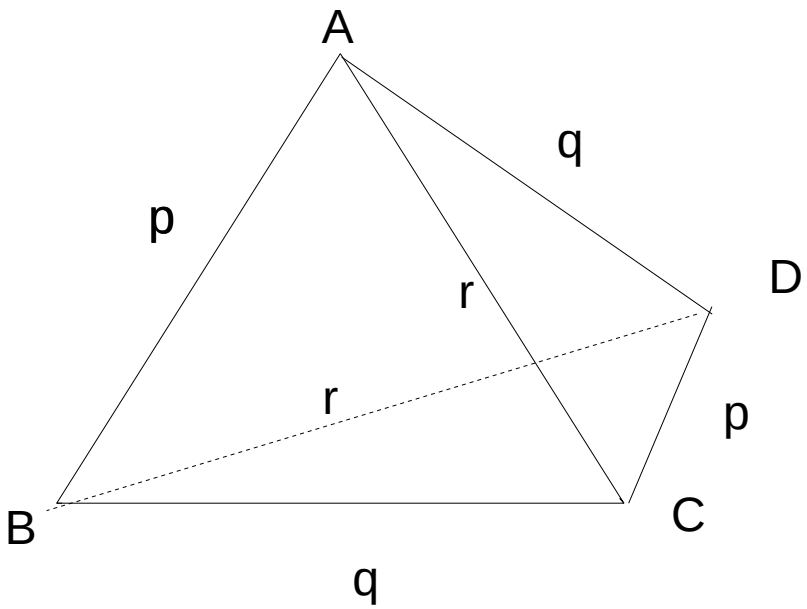
Coïncidence fait penser à co-incidence, incidence à intersection … je me suis retrouvé avec trois droites (des segments suffisent) :



Chacun des côtés du triangle sera une carte, deux cartes auront en commun … leur intersection ; évidemment on n'a que 3 cartes, le jeu est bien maigre

Avec un carré ? Non, il y a des côtés non-incidents

Avec un tétraèdre : là aussi il y a des arêtes non-incidentes, mais j'y ai vu le qualificatif « opposées » et je leur ai mis des étiquettes correspondantes



Je résume : on a 6 cartes (les arêtes du tétraèdre), chacune a 3 dessins (qui pour l'instant ne sont que des lettres) le tout fait :

A p B

A r C

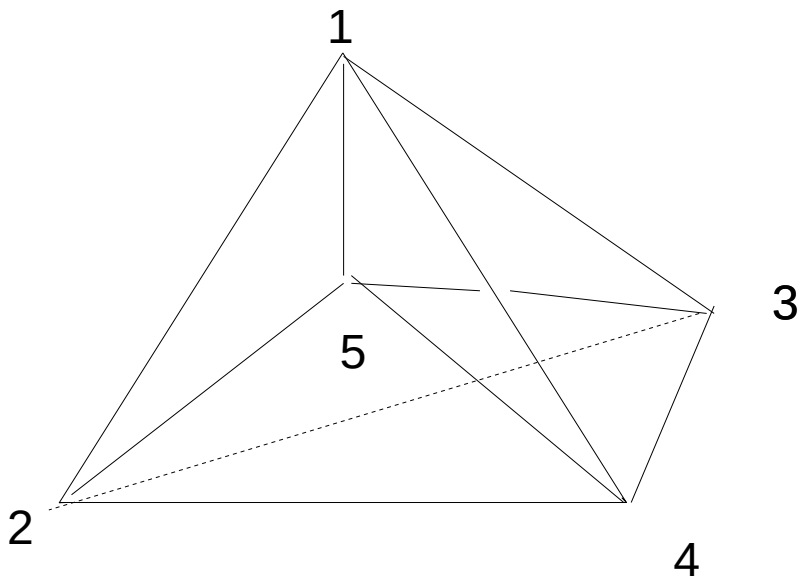
A q D

B q C

B r D

C p D

Pour continuer, on va traiter le ‘triangle équilatéral à 5 sommets’ (S_5)



On a cette fois 10 arêtes, l’arête 1-2 porte déjà les étiquettes 1 et 2 ce qui fait que (pour l’instant) elle n’a rien de commun avec 3-5

Je passe en version ‘tableur’ : avec 10 lignes et 10 colonnes, il me faut remplir le triangle strictement supérieur par les étiquettes voulues

biDobble pour le Simplexe à 5 sommets

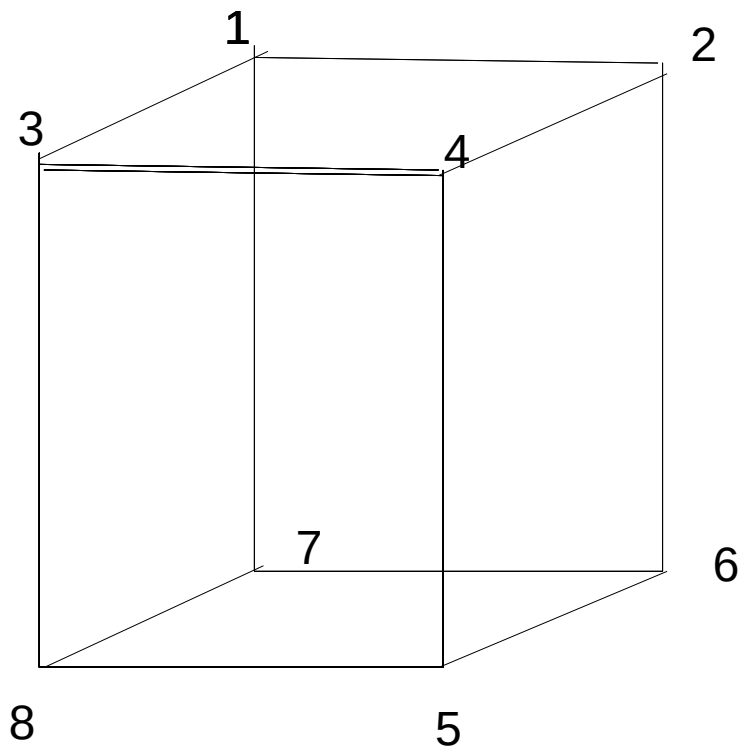
	1 2	1 3	1 4	1 5	2 3	2 4	2 5	3 4	3 5	4 5
1 2		1	1	1	2	2	2	a'	a"'	a"
1 3			1	1	3	b'	b"'	3	3	b"
1 4				1	c'	4	c"'	4	c"	4
1 5					d'	d"	5	d"'	5	5
2 3						2	2	3	3	e'
2 4							2	4	f'	4
2 5								g	5	5
3 4									3	4
3 5										5
4 5										

1 2	1 3	1 4	1 5	2 3	2 4	2 5	3 4	3 5	4 5
1 2 a' a" a"'	1 3 b' b" b"'	1 4 c' c" c"'	1 5 d' d" d"'	2 3 c' d' e'	2 4 b' d"' f'	2 5 b"' c"' g	3 4 a' d" g	3 5 a"' c" f'	4 5 a" b" e'

On est encore loin du but : je n'ai que 10 cartes, elles n'ont que 5 dessins chacune et (pour obtenir si peu) j'utilise déjà 17 dessins

Un vrai cas

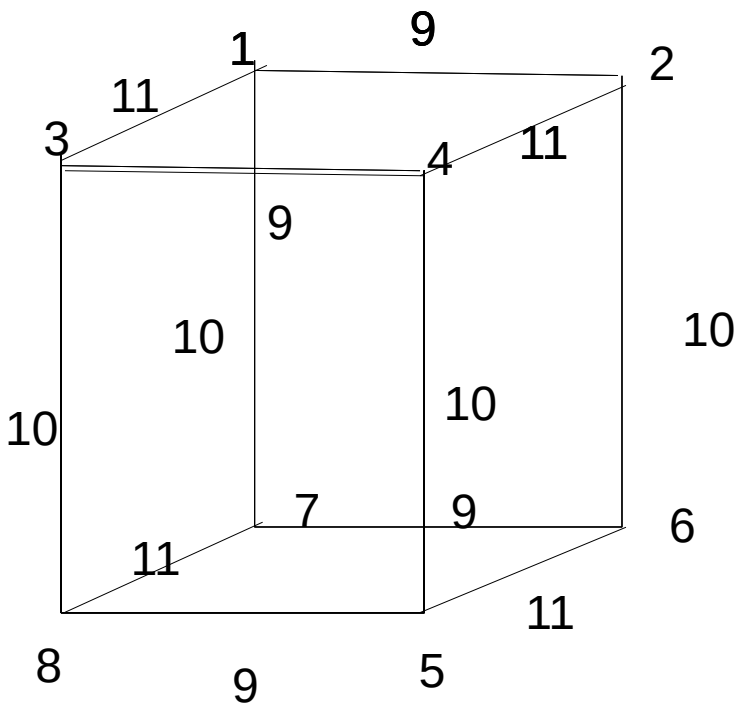
On passe à 8 sommets, mais 12 arêtes seulement : c'est le cube ordinaire de notre dimension 3



Comme dans les cas précédents chaque arête est déjà étiquetée par ses deux coins

Ensuite on regroupe les arêtes parallèles : on écrit 9 sur les quatre parallèles à Ox, 10 pour Oz et 11 pour Oy

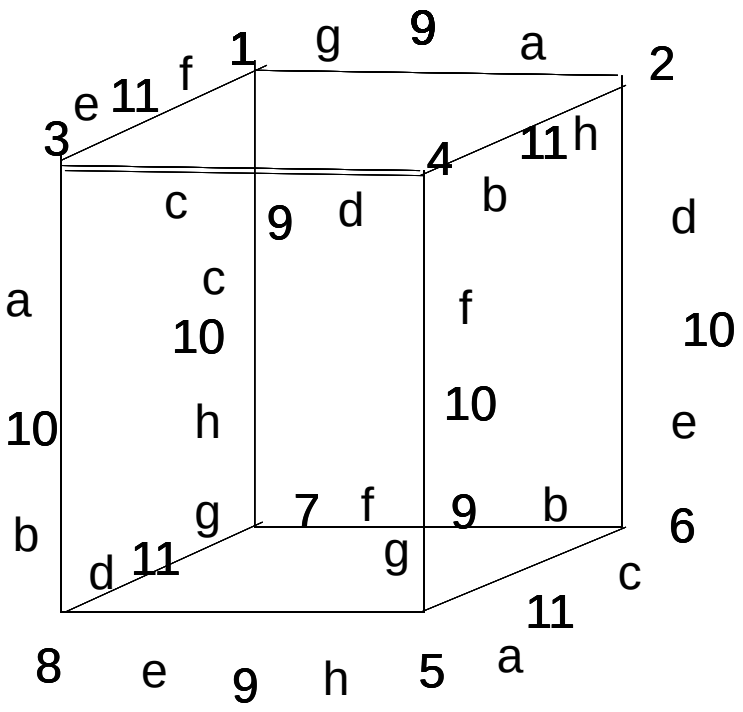
Comme on avait 12 arêtes, on a (12 . 11)/2 = 66 paires, on en a déjà vues 3 par coin (ça en fait 24) + 3 fois 4 parallèles (cela fait 3 fois 6 = 18 paires) ce qui fait qu'on en a vu 42 restent 24 … on va les regrouper, mais sans qu'on ait des cas déjà vus : donc pas deux parallèles, et jamais de coin commun : donc au plus 3, il reste donc



au moins 8 groupes (d'au plus 3) … sans coin commun et sans parallélisme

On verra plus tard pour de la théorie, ici on regarde (avec un crayon et une gomme) et j'ai ainsi trouvé les triplets comme le « a » : 3-8, 1-2, 5-6

« Comme vous voyez » sur le dessin qui suit … on n'y voit plus rien et il sera plus facile de vérifier sur une liste que sur ce dessin



Les voilà en mode liste :

1	g	9	a	2
1	f	11	e	3
1	c	10	h	7
2	h	11	b	4
2	d	10	e	6
3	c	9	d	4
3	a	10	b	8
4	f	10	9	5
5	a	11	c	6
5	h	9	e	8
6	b	9	f	9
7	g	11	d	8

On sent avoir atteint les limites de cette technique : sur le dessin comme sur la liste c'est si peu clair qu'on ne peut pas vérifier

N'ayant pas d'autres idées, j'ai recommencé avec l'octaèdre régulier … ça marche encore et on (je) n'y comprend toujours rien

Retour au jeu de Dobble : bi-Dobble

Avec les cas le plus évolué de ceux vus ci-dessus on a 12 cartes, 5 dessins par carte, au total 19 dessins. 12 c'est déjà peu mais 5 c'est vraiment trop peu

On va donc mettre 5 paires de dessins : par exemple

3 sera : 'un chien et un chat', mais pas forcement collés l'un à l'autre, pas à chaque fois les mêmes images ou photos

h sera : 'a et b', mais majuscule ou minuscule, petit ou gros, manuscrit ou typographique

De plus lors du jeu il faut dire les noms « génériques » comme animaux, lettres, couture, musique, écrire …

Dans l'exemple ci-dessus les deux **a** sautent aux yeux, mais les cartes étant rondes ce n'est pas toujours aussi apparent

Ou trouver un stock de tels dessins : sur pixabay.com il y en a un énorme stock. Défaut : elles ne sont pas au même format ni au même type, après les avoir téléchargées il y a du temps pour tout mettre dans le même style

Pour 'fabriquer' les cartes j'ai utilisé Gimp (on peut peut-être avec OpenOffice Draw .. mais je n'y suis pas arrivé)

Une autre variante : tri-Dobble

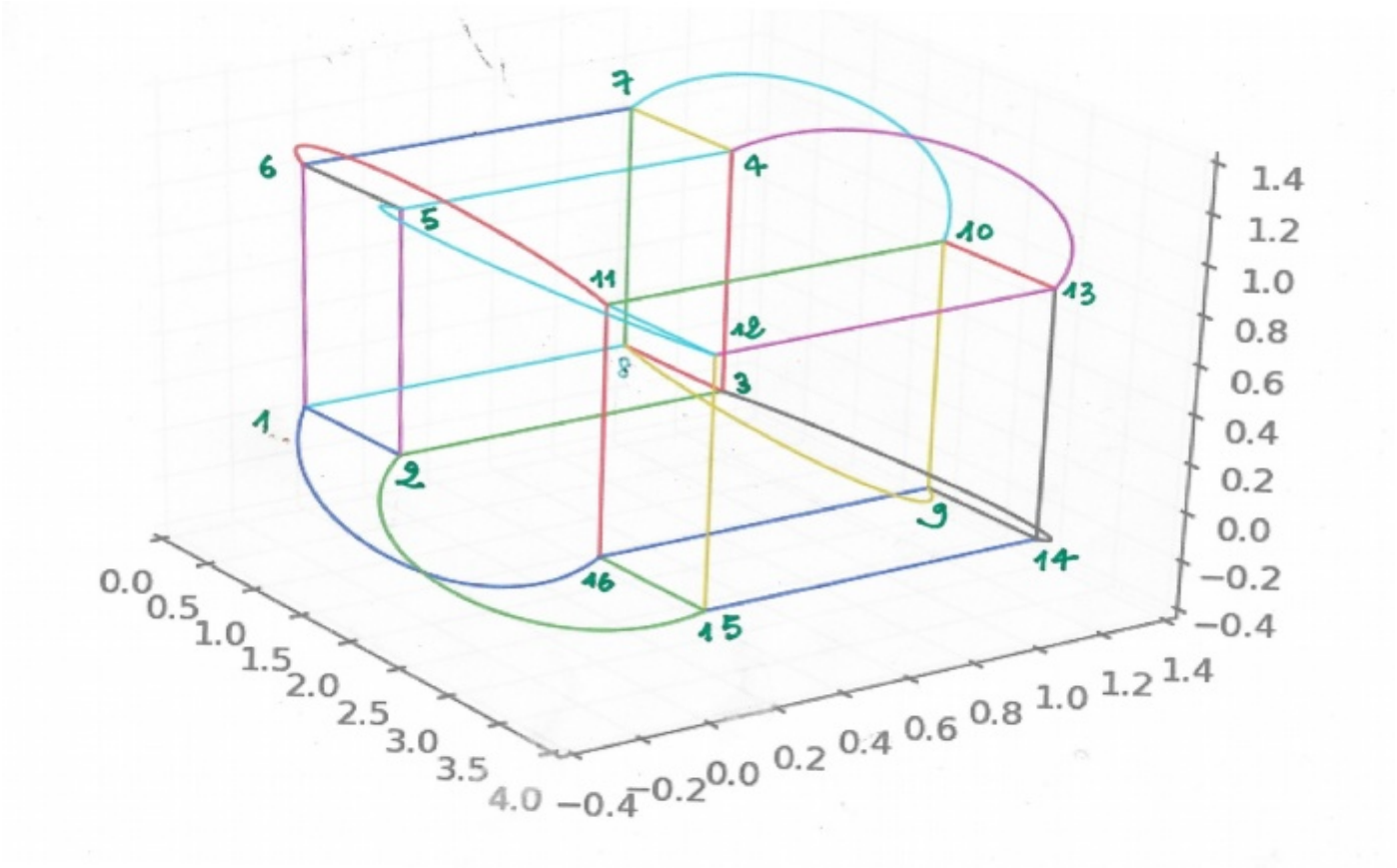
On retourne au tétraèdre : on avait trouvé 6 cartes, à 3 dessins. La nombre 6 reste très petit comme nombre de cartes, mais il y a un remède pour le nombre de dessins : si on dé-triple les dessins on en fera des groupes de 3 à identifier comme tels

➔	homme, femme, enfant	à identifier comme	famille
➔	chien, poisson, oiseau	…..	animaux
➔	+ , -, *	….	opérations

Pour éviter de ‘savoir par cœur ’ les contenus, il faut alors fabriquer un ‘grand nombre de groupes de 6 cartes’

Le cas du cube de dimension 4

Encore une fois utilisez internet pour voir des images de cubes de dimension 4 (à partir du nom tesseract) : la vision la plus courante utilise deux cubes emboîtes, j’ai essayé ainsi mais les croisements d’arêtes sont tels que l’on n’y voit rien très vite



J’ai donc dessiné (là je passe à l’imagerie Python) deux cubes côte-à-côte (le cube de gauche est pensé au niveau t=0, celui de droite t=1) en reliant leurs sommets ; la numérotation des sommets est un ‘code de Gray’ (dit aussi code de carillonneurs) : on ne change qu’une des coordonnées en passant d’un sommet au suivant

Bilan : on a 16 sommets, 32 arêtes, (32 . 31)/2 = 496 paires d’arêtes

On veut mettre un ensemble d'étiquettes sur chaque arête de telle sorte que toute paire d’arêtes ait UNNE étiquette commune

Les arêtes de direction Ox (idem Oy, Oz, Ot) forment ainsi un (donc quatre) paquet de 8 ce qui fait (8 . 7)/2 . 4 = 112 paires traitées

Les arêtes qui aboutissent à 1 (idem 2...16) sont quatre donc 6 paires : cela traite 6 . 16 = 96 nouvelles paires

Il reste 496 – 112 – 96 = 288 paires à traiter

Les paquets à venir n’auront qu’au plus 4 arêtes (on n’en veut pas deux parallèles) parmi lesquelles on ne veut pas de sommet commun : je vais les nommer QSPSBC =

QuadrupletSansParallelismeSansBoutCommun.

Idéalement on voudrait huit tels QSPSBC pour recouvrir les 32 arêtes

Une telle famille de huit QSPSBC fournira 8 . 6 = 48 paires et 48 divise 288 !! ça vaut le coup d'essayer. Ces OctoQSPSBC sont nommés OQSPSBC

Comme c'est ce que j'avais fait en dimension 3 j'ai essayé 'à la main' muni de copies du dessin colorié ci-dessus … et je n'ai pas trouvé un seul OQSPBSC : c'est trop compliqué, on n'y voit rien, on se trompe tout le temps → on va donc programmer tout cela (c'est le moment d'aller voir du code Python)

(allez voir K4 version 11,py ou plutôt K4 version 11,pdf "pour des raisons de sécurité" wordpress refuse les .py)

_ _ _ _ _ _

Retour de la programmation : ça a marché j'ai un OQSPSBC (celui noté RP(0, [14, 0, 0, 0, 0, 0, 0, 0]) dans les pages de programme) :

[1 , 2]	[6 , 7]	[3 , 4]	[5 , 12]
[3 , 8]	[4 , 5]	[1 , 6]	[2 , 15]
[4 , 7]	[1 , 8]	[2 , 5]	[3 , 14]
[5 , 6]	[2 , 3]	[7 , 8]	[1 , 16]
[9 , 14]	[10 , 11]	[12 , 15]	[4 , 13]
[10 , 13]	[14 , 15]	[11 , 16]	[8 , 9]
[11 , 12]	[9 , 16]	[13 , 14]	[7 , 10]
[15 , 16]	[12 , 13]	[9 , 10]	[6 , 11]

Je viens de vérifier avec le dessin-en-couleurs-numéroté : c'est bien un OQSPSBC. Ci-dessous ce OQSPSBC sera OQ_0

Je suis retourné voir la dimension 3 : les huit TSPSBC se déduisent du premier (nomme 'a') par des symétries !

Alors au lieu de chercher d'autres OQSPSBC il suffit de chercher les images de celui déjà trouvé par les isométries du cube

Les isométries de K4 ?

Vision géométrique :

Si S1...S16 sont les sommets je sais pouvoir trouver 16 isométries du cube envoyant S1 sur S1 … S16 (par exemple en enchaînant des symétries du cube). En les utilisant à l'envers je peux ramener S1...S16 en S1

Il me reste à étudier les isométries de K4 telles que S1 → S1. Ces isométries vont forcément expédier les voisins de S1 (les sommets à distance 1 de S1) sur des voisins de S1. Ces voisins sont 4 (on est en dimension 4), ils sont 2 à 2 équidistants (de $\sqrt{2}$) ils constituent donc un T3 (tétraèdre régulier de la dimension 3) qui a 4 ! = 24 isométries

J'ai donc 16 fois 24 (372) isométries de K4

Vision matricielle :

Je change le centre et le côté du cube : mon centre est O le côté est 2, mes coins sont [±1, ±1 ±1 ±1], mon isométrie transforme les [1, 0, 0, 0] en des [0 , 0 , -1, 0] (ce sont les isobarycentres de quatre sommets contigus) et les matrices sont des « matrices de substitution) (unn 1 par ligne ou colonne) au signes près on en retrouve 24 . 16 = 372

(si vous y tenez : aller voir la fin de K4 version 11,pdf)

– – – – – –

Ce qui reste à faire ? : prendre l'image de OQ_0 par ces 372 applications, éliminer celles ayant une arête commune avec OQ_0, enfin en chercher 5 n'ayant pas de paire en conflit (= en commun) avec les paires de OQ_0

Toutes réflexions faites les 372 ne vont pas servir : dès que l'image d'un axe sera ± lui même on est sûr que l'arête du quadruplet sera elle même, si donc cela se produit deux fois la case du tableau triangulaire à remplir (voir page 4) sera déjà occupée

On retire ces isométries … il en reste 272

On calcule alors les 272 images de OQ_0 et on va chercher à les 'juxtaposer' comme on avait fait avec le cube de dimension 3. Je nomme OQ_1 le premier venu

On compare OQ_0 et OQ_1 : c'est bon ces OQSPQBC sont compatibles

Mais Q2 n’est pas compatible avec OQ_0, ni OQ_1 avec Q3 non plus … finalement aucun des 270 restant ne va

Finalement il y a deux paquets de 136 OQSPSBC : chaque q de Paquet1 est compatible avec chaque q’ de Paquet2 mais deux de Paquet1 ou de Paquet2 sont incompatibles. J’imagine qu’il y a une raison théorique (‘parité’? Orientation ?)

J'ai cherché ,,, et je n'ai pas trouvé

où en est-on ?

On avait 496 paires d’arête, 112 étiquetées par parallélisme, 96 par coin commun, sur les 288 restantes on a OQ_0 et OQ_1 qui fournissent 48 paires chacun : on en est à 304.

Les regroupements d’arêtes sont toujours à chercher parmi les QSPQBC (mais plus 8 d’un coup) en vérifiant qu’il n’y a pas de collision : les paires d’arêtes que l’on va enregistrer sont ‘neuves’ et ces nouveaux enregistrements ne mettent pas d’étiquettes communes sur nos carte (= arêtes du cube)

On trouve 10 QSPSBC : on en est à 364

et maintenant ?

On va encore chercher parmi les QSPSBC … mais on sait qu’il va y avoir un conflit : un au moins des enregistrements va créer un conflit : deux cartes auront deux symboles communs

Il faut alors dés-enregistrer le symbole d’une des cartes

On programme cela : on en trouve 10, ce qui ajoute (non pas 60 cases mais) 30 : chaque désenregistement coûte 3 : on est passé à 394

on continue alors en essayant d’enregistrer des QSPSBC (de perspectives au plus 4) et éventuellement on désenregistre en cas de conflit … on arrive à 415

on continue alors en essayant d’enregistrer des QSPSBC (de perspectives au plus 3) et éventuellement on désenregistre en cas de conflit … on arrive à 424

on continue alors en essayant d’enregistrer des QSPSBC (de perspectives au plus 2) : il n’y en a pas

on continue alors en enregistrant les cases restées vides jusqu’à 496

le bilan

On a utilisé 130 symboles, répartis sur 32 cartes, chacune de ces cartes a un nombre de symboles compris entre 11 et 14, deux cartes ont unn dessin commun

Il reste à fabriquer ces cartes … là encore de la programmation doit pouvoir aider (?) on peut peut-être dire à GIMP « prend chacun des dessins des cartes et tourne le d'un angle aléatoire »

La fabrication des cartes

Première chose à avoir : 130 images. Pour cela pixabay est parfait, au lieu de prendre « toutes les images » on se limite à « images vectorielles », presque toutes sont déjà sur fond transparent ce qui gagnera du temps

Les images issues de pixabay sont des .png ou des .jpg, on les ouvre avec GIMP, puis

1) on les met à le taille (commande GIMP : Maj T) en remplaçant leur plus grande dimension par 64 puis on les déplace (commande GIMP : M) pour qu'elles ne soient plus dans le coin du rectangle de fond transparent

2) mes images étaient numérotées : j'ai tourné la 28-ième (par exemple) de 28 fois 30° (module 360°) (commande GIMP : Maj R)

Je me suis alors fait des « ébauches » de cartes avec 11, 12, 13, 14 points prêts pour y placer le images, ces ébauches ont une bordure ronde rouge d'épaisseur 2mm (de diamètre 19 cm)

Avant d'attaquer la confection on demande à Python une liste (de k de 0 à 31) des

[k, _ ,nombre d'images de la carte, _ ,liste des numéros des images à y placer]

Il est alors presque rapide de faire les cartes … mais j'ai découvert qu'il fallait s'auto-programmer-en-papier en écrivant la liste des choses à faire sur une feuille. A titre d'exemple je vous donne ma liste d'instructions

- ➔ ouvrir « carré 21x21 vide.xcf » (nomme 21 ci-dessous)
- ➔ ouvrir le fichier d'image voulu (Ex : carte 16.xcf), cliquer sur le cadre
- ➔ CTRL C
- ➔ cliquer sur le cadre de 21
- ➔ CTRL V
- ➔ Maj T
- ➔ taper 748 puis cliquer sur Echelle (j'avais mon GIMP qui voulait rester en mode pixel)
- ➔ M (pour Move cela permet de bouger l'image à la souris)

➔ glisser l'image au centre de la fenêtre

➔ Maj Ctrl E (pour Exporter) puis entrer un nom comme « carte 16 19.jpg » puis Entrer puis Exporter

➔ cliquer dans le cadre de 21 puis « suppr » (la fenêtre 21 est re-vide)

➔ fermer le fichier d'image voulu

Vous le devinez on peut sûrement faire plus malin, créer des macros (?), écrire un script (en bash ? en Python?)

Avec ce qui précède les images.jpg doivent s'imprimer dans un cercle de diamètre 19 cm au centre de votre feuille A4

Une autre variante : Trebble

Cette fois ci la chose ne commun sera commune à TROIS cartes, pour faire cela au lieu de penser 'deux droites ont unn point commun' on passe à 'trois plans ont unn point commun'

C'était déjà un peu faux avec les droites ou segments, ça va être pire avec les plans, mais c'est quand même l'idée.

Pour éviter les parallélismes : je passe en projectif, pour assurer la finitude : il suffit de prendre un corps fini

Pour avoir des triplets de plans il me faut un $P_3(K)$ ... quel K ? le calcul du cardinal m'a vite amené à $F_3 = Z/3Z$

Ci dessous $E = P_3(F_3)$, comment numériser cela ? On utilise juste la définition :

$E = (F_3)^4$ */colinéation, parmi les quadruplets non nuls on distingue par nombre de zéros à la fin et le dernier terme non nul est pris égal à 1, on a donc :

les [a,b,c,1] les [u,v,1,0] les [m,1,0,0] et [1,0,0,0]

Avec card(F_3)=3 cela fait 27+9+3+1=40 points

Les plans de E sont [les sous-espaces de dimension 3 de $(F_3)^4$]* /colinéation chacun d'entre eux à une équation ax+by+cz+dt =0 pour un (a,b,c,d) dans E, c'est un orthogonal au sens dualité (et pas euclidien)

La programmation

```
import numpy as np                    # sert uniquement à np.vdot le produit scalaire
q=3

K=range(q)                                    # K =K1 = [0,1,2]
K2=[[x,y] for x in K for y in K]              # K2 = les couples
K3=[[x,y,z] for x in K for y in K for z in K] # K3 =  les triplets

""" on produit tous les points de l'espace projectif EP """
P1=[x+[1] for x in K3]                # P1 = pas à l'infini
P2=[x+[1,0] for x in K2]              # le + concatène les listes
P3=[[x]+[1,0,0] for x in K]
P4=[[1,0,0,0]]
EP=P1+P2+P3+P4

""" EP = [[0, 0, 0, 1], [0, 0, 1, 1], [0, 0, 2, 1], [0, 1, 0, 1], [0, 1, 1, 1],
          [0, 1, 2, 1], [0, 2, 0, 1], [0, 2, 1, 1], [0, 2, 2, 1], [1, 0, 0, 1],
          [1, 0, 1, 1], [1, 0, 2, 1], [1, 1, 0, 1], [1, 1, 1, 1], [1, 1, 2, 1],
          [1, 2, 0, 1], [1, 2, 1, 1], [1, 2, 2, 1], [2, 0, 0, 1], [2, 0, 1, 1],
          [2, 0, 2, 1], [2, 1, 0, 1], [2, 1, 1, 1], [2, 1, 2, 1], [2, 2, 0, 1],
          [2, 2, 1, 1], [2, 2, 2, 1], [0, 0, 1, 0], [0, 1, 1, 0], [0, 2, 1, 0],
          [1, 0, 1, 0], [1, 1, 1, 0], [1, 2, 1, 0], [2, 0, 1, 0], [2, 1, 1, 0],
          [2, 2, 1, 0], [0, 1, 0, 0], [1, 1, 0, 0], [2, 1, 0, 0], [1, 0, 0, 0]] """

def evalue(U,V):                      #le produit scalaire est modulo 3
   return (np.vdot(U,V) % q)

def pointsDe(p):     # on a les rangs dans EP des points du plan P
   return [EP.index(x) for x in EP if evalue(p,x)==0]

cartes=[pointsDe(plan) for plan in EP]

"""[[27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
```

[2, 5, 8, 11, 14, 17, 20, 23, 26, 36, 37, 38, 39] …… ‘””’

```
""" des fonctions pour vérifier l’intersection de trois plans """
def intersection(P,Q,R):
    return [x for x in P if (x in Q and x in R)]

""" PC est le Paquet de Cartes à traiter"""
def verifie(PC,a,b,c):
    return len(intersection(PC[a],PC[b],PC[c]))==1
```

et le résultat est … MAUVAIS ! 520 fois l’intersection n’est pas de cardinal 1, j’avais oublié les ‘faisceaux de plans’ qui contiennent un même droite

J’ai bien essayé de recommencer en retirant un puis deux puis 3 plans 520 passe à 482 puis 440 puis 408 … mais le ‘taux de mauvais triplets’ ne varie pas

Comme en cas de ‘mauvais triplet’ on a quand même une intersection non vide et que 520/(40.39.38) < 0.9 %, j’ai décidé que c’était bon quand même. « It’s not a bug it’s a feature »

Au passage quand j’ai obtenu ces intersections non ponctuelles, j’ai regardé comment est faite une droite de cet espace, en voici une :

A= [2 1 1 1]

B= [0 2 1 0]

C = [2 0 2 1]

D = [2 2 0 1]

À quoi voit-on que c’est une droite ? C = A+B et D = A-B, on peut aussi partir de B et C : A et D en sont différence et somme (tout cela est modulo 3 et dans le plan avant colinéation dont est issue notre droite)

Vision ‘géométrique’ : les sommets d’un tétraèdre, les somme et différence des sommets d’une arête forment l’arête opposée. C’est cohérent : le plan vectoriel est de cardinal 9, on retire 0 : 8, d’où 4 droites vectorielles organisables comme dit ci-dessus en somme-différence

Il n’y a plus qu’à fabriquer les cartes : cette fois ci elles seront triangulaires en l’honneur de q=3

_ _ _ _ _ _

Exemples de règles du jeu possibles :

_ avec au moins 2 joueurs

chacun part avec une carte, au centre le paquet des cartes restantes ; Quand un joueur a repéré un trio (lui+le pot+un autre joueur), il le nomme, prend la carte du pot central et la met sur (ou sous) sa pile

_ avec au moins 3 joueurs

après s'être partagé les cartes les joueurs sont en rond avec leur tas de cartes montrant celle du dessus visible ; chacun joue contre-avec ses deux voisins : quand il a repéré un trio il prend la carte de chacun de ses voisins et les met sur (ou sous) sa pile

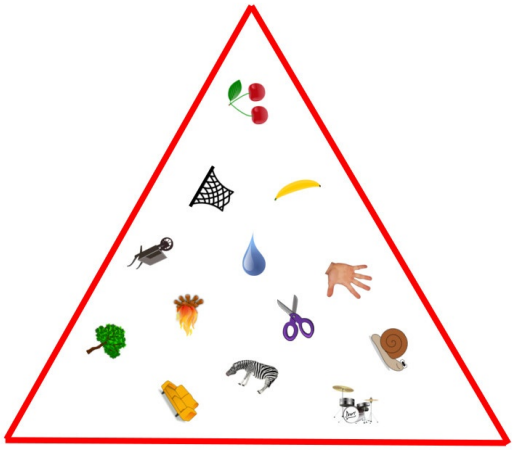
Variante : une fois le trio repéré il donne sa carte au voisin qu'il choisit, le but est alors de se débarrasser de toutes ses cartes

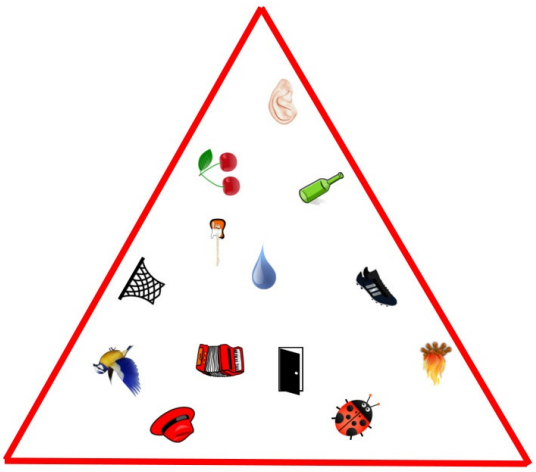
_ avec au moins 4 joueurs

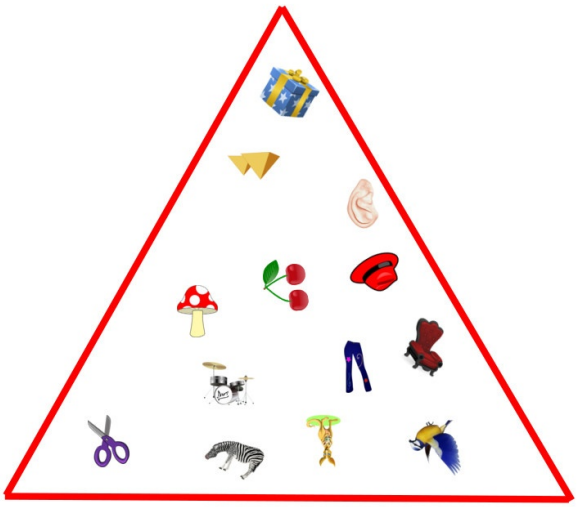
après s'être partagé les cartes les jours sont en rond avec leur tas de cartes montrant celle du dessus visible ; chacun joue contre TOUS les autres (mais pas lui) : dès qu'il a repéré un trio il le nomme et prend les trois cartes

_ _ _ _ _ _

Dans le cas des 3 cartes qui suivent …. l'image commune est : les cerises. Elles sont vues ici 'dans le même sens' mais quand on joue les cartes sont tournées 'pas forcément comme cela'







Et une dernière : Quabble

Cette fois ci la chose en commun sera commune à QUATRE cartes, pour faire cela au lieu de penser ‘deux droites ont unn point commun’ on passe à ‘quatre sous-espaces de dimension trois ont unn point commun’

C’était déjà un peu faux avec les droites ou segments, ça va être pire avec les plans, mais c’est quand même l’idée.

Pour éviter les parallélismes : je passe en projectif, pour assurer la finitude : il suffit de prendre un corps fini

Pour avoir des quadruplets de 3-plans il me faut un $P_4(K)$ ... quel K ? le calcul du cardinal m’a vite amené à $F_2 = Z/2Z$

Ci dessous $E = P_4(F_2)$, comment numériser cela ? On utilise juste la définition :

$E = (F_2)^5$ */colinéation, parmi les quintuplets non nuls on distingue par nombre de zéros à la fin et le dernier terme non nul est pris égal à 1, on a donc :

les [a,b,c,d,1] les [u,v,w,1,0] les [m,n,1,0,0] les [x,1,0,0,0] et [1,0,0,0,0]

Cela fait 31 points

Les 3-plans de E sont [les sous-espaces de dimension 4 de $(F_2)^5$ */colinéation chacun d’entre eux à une équation a.x+b.y+c.z+d.t +e.u=0 pour un (a,b,c,d,e) dans E : c’est un orthogonal au sens dualité (et pas euclidien)

La programmation

Note : hors commentaires, affichage de résultats, vérifications, ce programme fait 15 lignes et je compte les en tête des définitions de fonctions

```
import numpy as np        """ servira juste à m’éviter de réécrire le produit scalaire """
q=2

""" on produit tous les 31 points de l’espace projectif, c’est un peu idiot : comme q=2 la colinéation c’est … rien, on a juste les points non nuls de K^5 """

K1=[x for x in range(q)]
```

```
K2=[[x,y] for x in K1 for y in K1]
K3=[[x,y,z] for x in K1 for y in K1 for z in K1]
K4=[[x,y,z,t] for x in K1 for y in K1 for z in K1 for t in K1]

P1=[x+[1] for x in K4]
P2=[x+[1,0] for x in K3]
P3=[x+[1,0,0] for x in K2]
P4=[[x]+[1,0,0,0] for x in K1]
P5=[[1,0,0,0,0]]
P=P1+P2+P3+P4+P5

""" P = [[0, 0, 0, 0, 1], [0, 0, 0, 1, 1], [0, 0, 1, 0, 1], [0, 0, 1, 1, 1],
        [0, 1, 0, 0, 1], [0, 1, 0, 1, 1], [0, 1, 1, 0, 1], [0, 1, 1, 1, 1],
        [1, 0, 0, 0, 1], [1, 0, 0, 1, 1], [1, 0, 1, 0, 1], [1, 0, 1, 1, 1],
        [1, 1, 0, 0, 1], [1, 1, 0, 1, 1], [1, 1, 1, 0, 1], [1, 1, 1, 1, 1],
        [0, 0, 0, 1, 0], [0, 0, 1, 1, 0], [0, 1, 0, 1, 0], [0, 1, 1, 1, 0],
        [1, 0, 0, 1, 0], [1, 0, 1, 1, 0], [1, 1, 0, 1, 0], [1, 1, 1, 1, 0],
        [0, 0, 1, 0, 0], [0, 1, 1, 0, 0], [1, 0, 1, 0, 0], [1, 1, 1, 0, 0],
        [0, 1, 0, 0, 0], [1, 1, 0, 0, 0], [1, 0, 0, 0, 0]]  """

def evalue(P,V):
    return (np.vdot(P,V) % q)

def pointsDe(Plan):
    return [P.index(x) for x in P if evalue(Plan,x)==0]

cartes=[pointsDe(plan) for plan in P]

"""[[16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
 [1, 3, 5, 7, 9, 11, 13, 15, 24, 25, 26, 27, 28, 29, 30],
 [2, 3, 6, 7, 10, 11, 14, 15, 16, 18, 20, 22, 28, 29, 30],
 [1, 2, 5, 6, 9, 10, 13, 14, 17, 19, 21, 23, 28, 29, 30],
 [4, 5, 6, 7, 12, 13, 14, 15, 16, 17, 20, 21, 24, 26, 30],
 [1, 3, 4, 6, 9, 11, 12, 14, 18, 19, 22, 23, 24, 26, 30],
```

```
 [2, 3, 4, 5, 10, 11, 12, 13, 16, 19, 20, 23, 25, 27, 30],
 [1, 2, 4, 7, 9, 10, 12, 15, 17, 18, 21, 22, 25, 27, 30],
 [8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 24, 25, 28],
 [1, 3, 5, 7, 8, 10, 12, 14, 20, 21, 22, 23, 24, 25, 28],
 [2, 3, 6, 7, 8, 9, 12, 13, 16, 18, 21, 23, 26, 27, 28],
 [1, 2, 5, 6, 8, 11, 12, 15, 17, 19, 20, 22, 26, 27, 28],
 [4, 5, 6, 7, 8, 9, 10, 11, 16, 17, 22, 23, 24, 27, 29],
 [1, 3, 4, 6, 8, 10, 13, 15, 18, 19, 20, 21, 24, 27, 29],
 [2, 3, 4, 5, 8, 9, 14, 15, 16, 19, 21, 22, 25, 26, 29],
 [1, 2, 4, 7, 8, 11, 13, 14, 17, 18, 20, 23, 25, 26, 29],
 [0, 2, 4, 6, 8, 10, 12, 14, 24, 25, 26, 27, 28, 29, 30],
 [0, 3, 4, 7, 8, 11, 12, 15, 17, 19, 21, 23, 28, 29, 30],
 [0, 2, 5, 7, 8, 10, 13, 15, 18, 19, 22, 23, 24, 26, 30],
 [0, 3, 5, 6, 8, 11, 13, 14, 17, 18, 21, 22, 25, 27, 30],
 [0, 2, 4, 6, 9, 11, 13, 15, 20, 21, 22, 23, 24, 25, 28],
 [0, 3, 4, 7, 9, 10, 13, 14, 17, 19, 20, 22, 26, 27, 28],
 [0, 2, 5, 7, 9, 11, 12, 14, 18, 19, 20, 21, 24, 27, 29],
 [0, 3, 5, 6, 9, 10, 12, 15, 17, 18, 20, 23, 25, 26, 29],
 [0, 1, 4, 5, 8, 9, 12, 13, 16, 18, 20, 22, 28, 29, 30],
 [0, 1, 6, 7, 8, 9, 14, 15, 16, 19, 20, 23, 25, 27, 30],
 [0, 1, 4, 5, 10, 11, 14, 15, 16, 18, 21, 23, 26, 27, 28],
 [0, 1, 6, 7, 10, 11, 12, 13, 16, 19, 21, 22, 25, 26, 29],
 [0, 1, 2, 3, 8, 9, 10, 11, 16, 17, 20, 21, 24, 26, 30],
 [0, 1, 2, 3, 12, 13, 14, 15, 16, 17, 22, 23, 24, 27, 29],
 [0, 1, 2, 3, 4, 5, 6, 7, 16, 17, 18, 19, 24, 25, 28]]     """

""" des fonctions pour vérifier  """
def intersection(P,Q,R,S):
    return [x for x in P if (x in Q and x in R and x in S)]

def verifie(a,b,c,d):
    return len(intersection(cartes[a],cartes[b],cartes[c],cartes[d]))==1

ListeRate=[[]]
```

```
rate = 0


for a in range(31):
    for b in range(a+1,31):
        for c in range(b+1,31):
            for d in range(c+1,31):
                if not(verifie(a,b,c,d)):
                    rate+=1
                    ListeRate=ListeRate+[[a,b,c,d]]
```

""" ZUT !!! 5426 fois les 4-plans forment un faisceau, l'intersection est un 2-plan, cela fait 17% des quadruplets

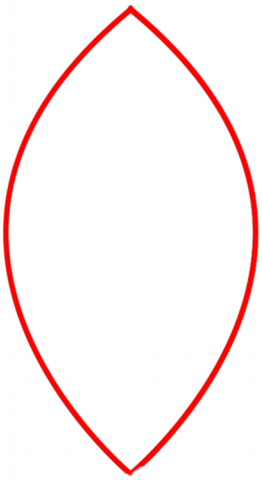
Quand ont a un 2-plan en commun cela fait TROIS quadruplets, on va dire que c'est fait exprès : « It's not a bug it's a feature »

Exemple : les cartes 0 3 20 26 ont en commun 21 23 28

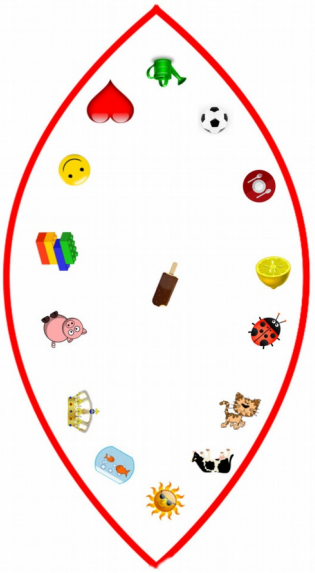
J'avais 'bêtement' eu peur d'avoir des 4-plans d'intersection un 3-plan … merci Grassmann, la dimension de l'espace ambiant n'est que 5 """

Les cartes

Quand j'avais q=3, j'ai dessiné des cartes à 3 côtés, maintenant q=2 et le polygone régulier à deux côtés laisse peu de place, d'où mes deux Bézier





Ci-dessus les cartes 3 6 20 21

Exemples de règles du jeu possibles :

_ avec au moins 1 joueur (?)

là le mieux est de chercher un autre joueur … si ce n'est vraiment pas possible :

vous faites quatre tas de cartes : un pour vous et trois 'talons', il faut trouver les dessin commun aux quatre tas. Avantage : comme vous êtes seul il est plus facile de tricher sans être vu

_ avec au moins 2 joueur (?)

deux cartes fixes (ou non selon votre choix) sont visibles, chaque joueur a sa pile (après bagarre car le nombre de cartes est impair), le premier qui a trouvé quel dessin était en 4 exemplaires, le nomme et, prend la carte de son adversaire

_ avec au moins 3 joueurs

trois joueur + un une pile … celui qui trouve prend les quatre, à la fin c'est la pile qui perd ! Alors chaque joueur lui en redonne un peu du dessous de sa pile

_ avec au moins 4 joueurs

…….

_ avec au moins 5 joueurs

dans l'exemple ci-dessus : c'est le soleil qui est en commun

Encore une : Diffle

Problème très voisin : fabriquer des cartes qui au lieu d'avoir un élément commun aient un élément différent

Solution très simple : des cartes avec un dessin chacune tous différents … intérêt zéro

Dans le style des jeux précédents on peut fabriquer 10 cartes ayant chacune 9 des 10 dessins d'un ensemble de base, deux cartes distinctes auront 8 dessins commun plus un autre