

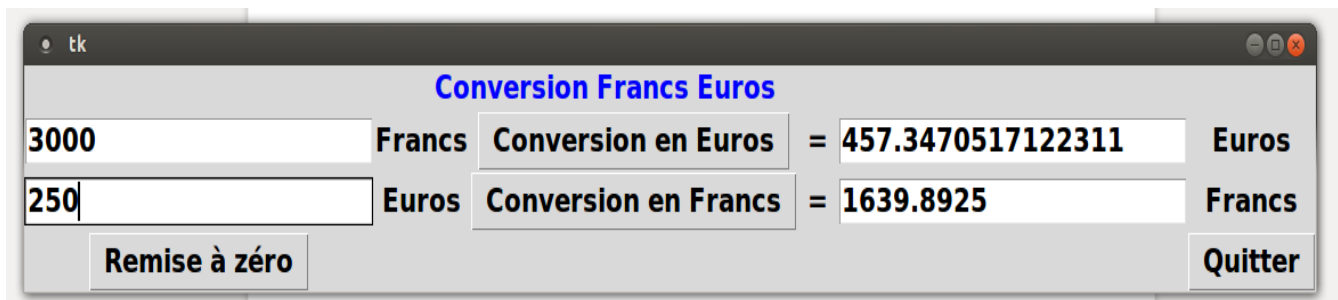
tk

Conversion Francs Euros

Francs **Conversion en Euros** = Euros

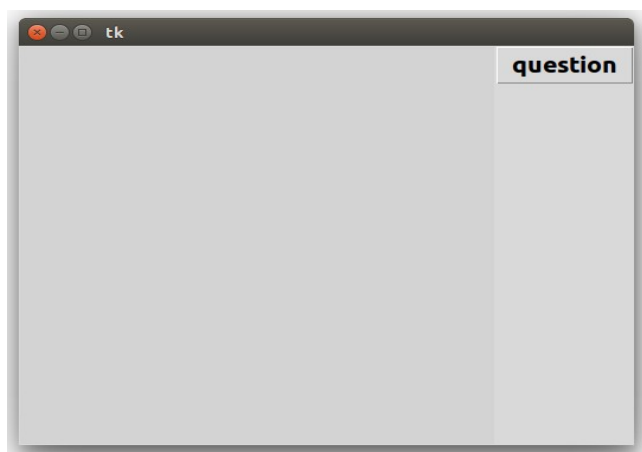
Euros **Conversion en Francs** = Francs

Remise à zéro **Quitter**



Exercice 2 - Jeu « associer un mot à une image »

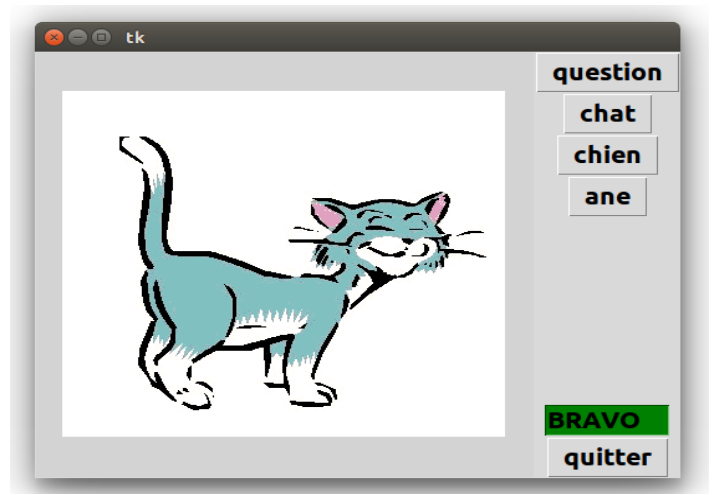
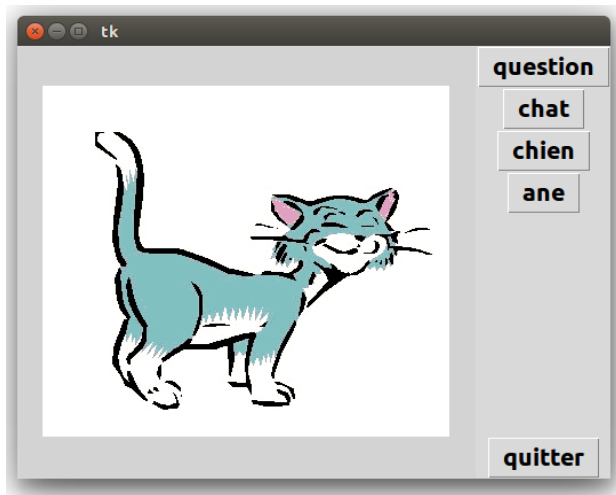
Il s'agit d'afficher une image et trois possibilités de réponse; l'utilisateur clique sur l'un des boutons réponse pour choisir le mot correspondant à l'image. Dans un « vrai » exercice il faudrait enchaîner plusieurs questions ce que nous ne ferons pas ici; nous nous contenterons aujourd'hui, pour mettre en place les widgets, d'une seule question.....



Au lancement du jeu apparaîtra une fenêtre avec un canvas et un bouton « question ».

Si on clique sur ce bouton, apparaîtra une image et les trois boutons réponses (ainsi qu'un bouton quitter)

Si on clique sur l'une des réponses, il apparaîtra une zone texte dans laquelle sera écrit soit FAUX soit BRAVO (et cette zone sera verte si c'est juste, rouge sinon).



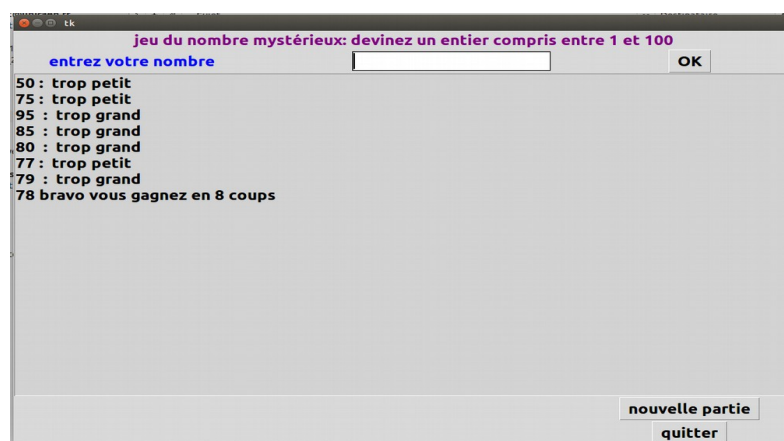
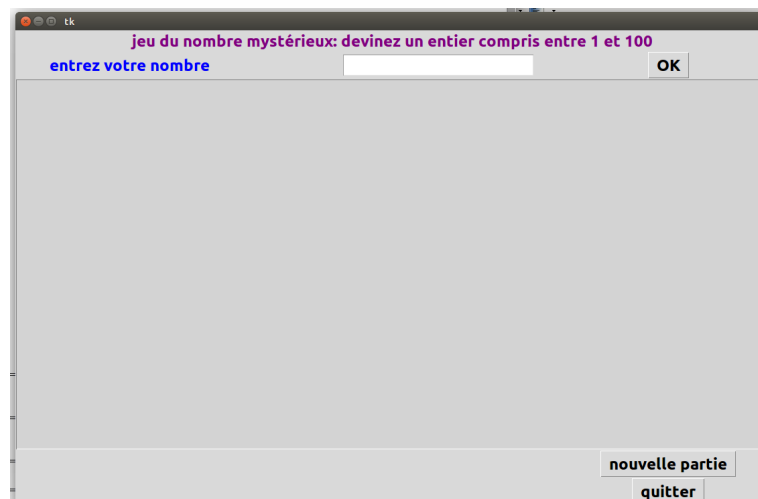
Exercice 3 - Un jeu : le jeu du nombre mystérieux

Une application à réaliser de façon simple puis plus sophistiquée.....

On veut écrire un petit logiciel de jeu: l'ordinateur choisit aléatoirement un nombre entre 1 et 100 et l'utilisateur doit deviner ce nombre.

Il s'agit ici de faire une version graphique en utilisant Tkinter.

Une présentation possible du jeu:

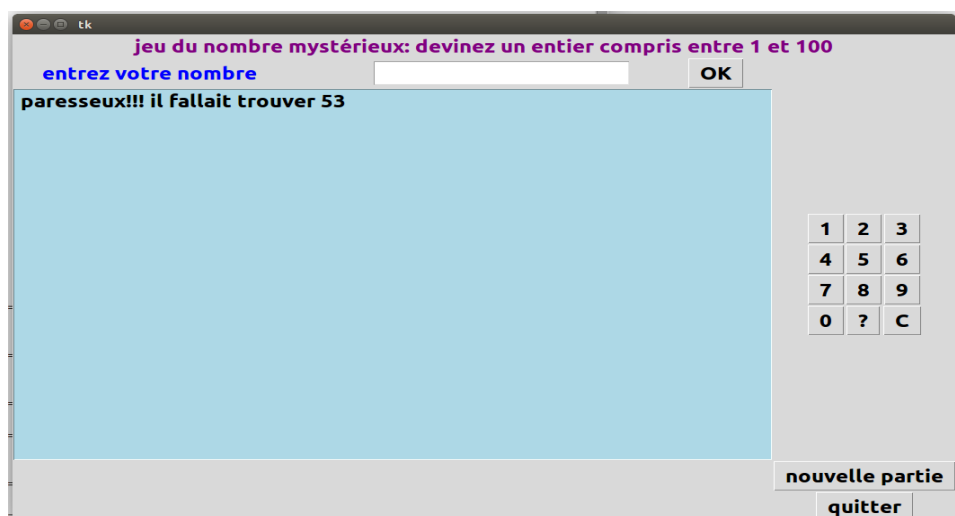


On commencera par une version simple qui permette de jouer .

On ne mettra pas en place de tests dans un premier temps.

On ajoutera ensuite des améliorations (dans l'ordre que vous voulez)

- mettre en place un pavé numérique pour la saisie des chiffres avec les touches C pour effacer la saisie en cours et ? pour demander la réponse (voir capture d'écran ci-dessous)
- mettre en place des tests correspondants à votre façon de saisir (saisie effective de chiffres, ne pas appuyer sur OK avant d'avoir saisi quelque chose.....)
- désactiver ou faire disparaître certains boutons quand l'utilisateur ne doit pas s'en servir (comme « nouvelle partie » au cours du jeu)
- mettre un smiley à l'écran quand on gagne
- avoir un nombre de coups max à jouer et perdre si on n'a pas trouvé avant
- compter des points sur plusieurs parties (en rajoutant des affichages)...
- vous pouvez aussi imaginer d'autres fonctionnalités...



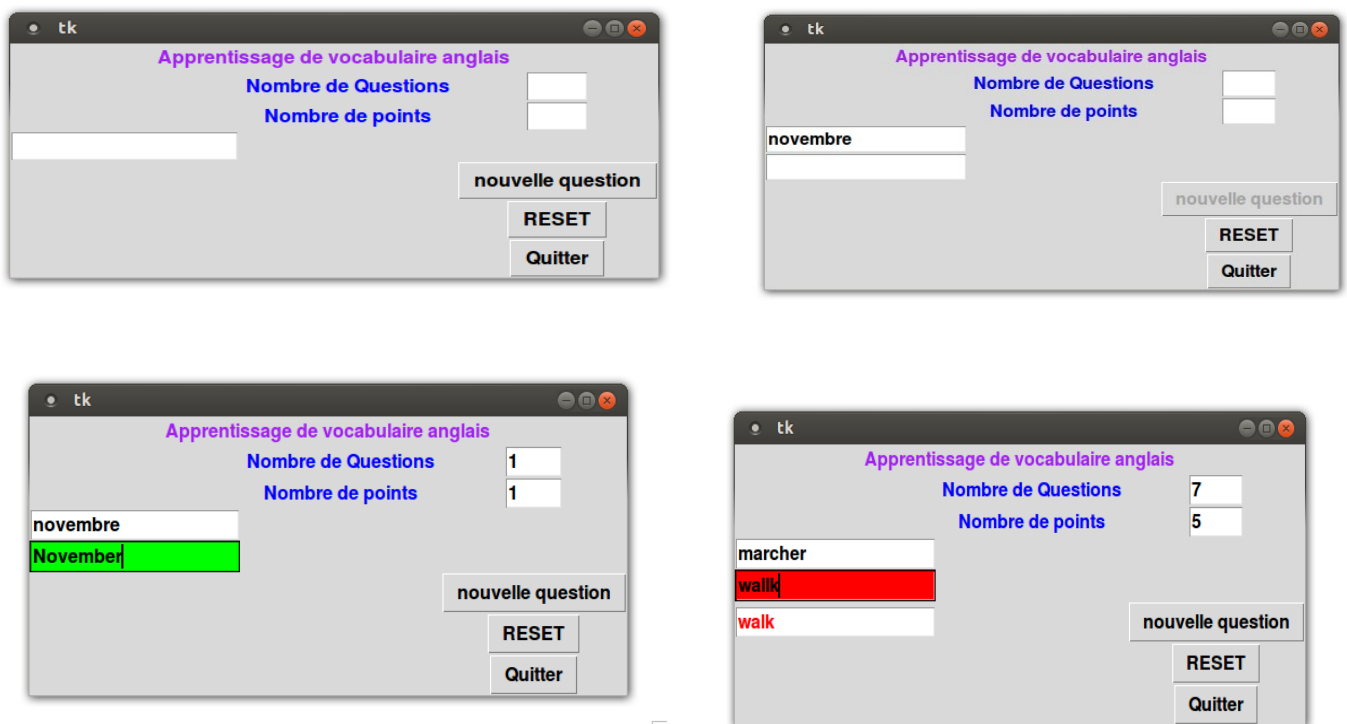
On pourra aussi changer de mode de saisie (entrée dans la zone de saisie)

Exercice 4 - Apprentissage de vocabulaire anglais et Tkinter

Il s'agit de réaliser un logiciel d'apprentissage de vocabulaire anglais. Pour cela, l'ordinateur dispose d'un dictionnaire de mots français/anglais. L'ordinateur donne à l'utilisateur un mot en français pris au hasard, celui-ci devra saisir la traduction anglaise de ce mot. L'ordinateur vérifiera la réponse et comptera les points et les questions.

Pour commencer l'utilisateur doit cliquer sur le bouton nouvelle question, qui incrémente le nombre de question et met en place une zone de saisie pour la réponse. L'ordinateur attendra le "return" de validation, après la saisie du mot en anglais, pour déclencher la vérification et le comptage du nombre de points.

L'interface se présente sous la forme suivante :



Remarque: on trouvera un dictionnaire pour les tests dans le fichier dicoanglais.py sur les pages moodle usuelles.

Remarques :

→ Faire « désactiver » le bouton "nouvelle question" tant que la réponse n'a pas été saisie ni validée.

→ si la réponse est juste, la réponse apparaît sur fond vert, sinon elle apparaît sur fond rouge et la correction s'affiche dessous.

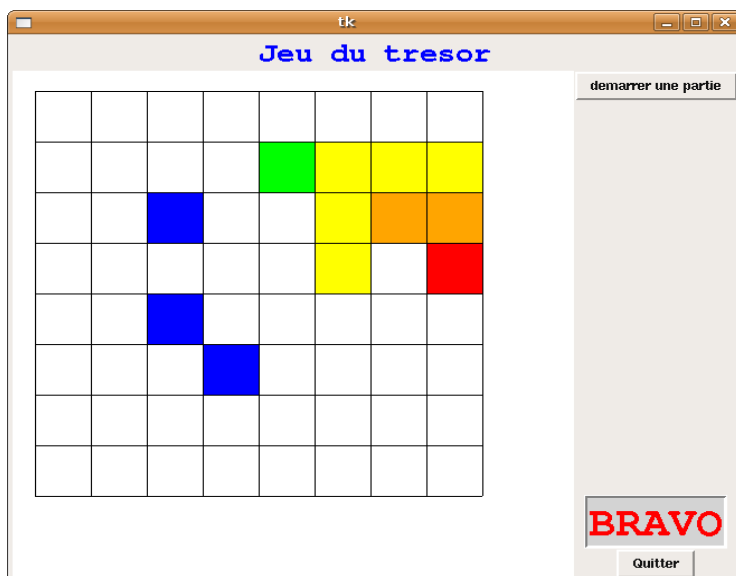
Exercice 5 - Jeu du Trésor :

Un trésor est caché aléatoirement dans une grille; pour le retrouver le joueur clique sur une case. La case se colorie en rouge si on a trouvé le trésor, en orange si on est très près (les 8 cases autour), en jaune si on est un peu plus loin (si on est à une distance de 2), en vert à une distance de 3 et en bleu sinon. Si on a trouvé le trésor, on doit faire s'afficher BRAVO, sinon l'utilisateur continue à cliquer tant qu'il n'a pas gagné (on prévoit quand même un bouton « quitter »).

On doit ensuite pouvoir faire une nouvelle partie (la grille est redessinée, un nouveau trésor est caché).

Remarques:

- rien ne doit s'afficher si on clique en dehors de la grille.
- si on a trouvé le trésor et si on continue à cliquer il ne doit rien se passer.
- la zone où s'affiche BRAVO ne doit être à l'écran que lorsqu'on a gagné (elle disparaît quand on fait une nouvelle partie).



Remarque : on n'a pas besoin de garder une trace de la grille et des coups déjà joués. On dessinera les cases de couleurs au fur et à mesure

Exercice 6 - Jeu Pierre-Feuille-Ciseau

Pierre-feuille-ciseaux est un jeu d'enfants: les deux joueurs choisissent simultanément un des trois coups en le symbolisant de la main.

De façon générale, la *pierre* bat les *ciseaux* (en les émoussant), les *ciseaux* battent la *feuille* (en la coupant), la *feuille* bat la *pierre* (en l'enveloppant).

Il s'agit de réaliser une petite interface pour jouer à "pierre/feuille/ciseau": on va réaliser un jeu qui permettra à un joueur de jouer contre l'ordinateur (qui choisira au hasard).

On réalisera l'interface suivante:



Pour les scores, on indiquera uniquement celui du joueur (pas celui de l'ordinateur); le joueur marquera un point s'il bat l'ordinateur, perd un point si c'est l'ordinateur qui gagne; aucun point n'est marqué si les deux ont choisi la même chose.

Pour jouer, le joueur clique sur l'un des boutons pierre feuille ou ciseau. A ce moment là, l'ordinateur choisit aléatoirement pierre feuille ou ciseau. Les deux images s'affichent dans les canvas et le score est mis à jour. Un nouveau click sur l'un des boutons pierre, feuille, ciseau efface les images et on recommence.

Le bouton "nouvelle partie" remet le score à Zéro, efface les canvas et permet de recommencer.

On pourra dans un premier temps ne pas mettre les images.

On suggère de:

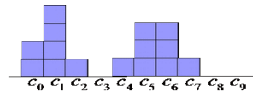
1. mettre en place les widgets
2. mettre en place le bouton nouvelle partie
3. mettre en place les boutons de jeu pierre, feuille, ciseau: pour cela on écrira une fonction jeu()

Exercice 7 - Simulation d'un tas de sable

Un tas de sable de longueur k sera représenté par une liste d'entiers $c = [c_0, \dots, c_{k-1}]$, où

c_i est le nombre de grains de la colonne i . On dit que le tas de table c contient une falaise (c_i, c_{i+1}) à la position i si $c_i \geq c_{i+1} + 2$. On représentera graphiquement un tas de sable par un diagramme où c_i qui est le nombre de grains de la i -ième colonne comporte c_i case(s) verticale(s).

Exemple: on considère le tas de sable $c = [2, 4, 1, 0, 1, 3, 3, 1, 0, 0]$, de longueur 10. Il contient deux falaises, (c_1, c_2) et (c_6, c_7) , aux positions 1 et 6. Son diagramme est :



On s'intéresse à l'évolution du tas de sable au cours du temps. Cette évolution est définie par la règle suivante : s'il y a une falaise à la position i , alors un grain peut tomber de la colonne i à la colonne c_{i+1} . Quand il y a plusieurs falaises à la fois, on choisit d'appliquer la règle à n'importe laquelle des falaises mais à exactement une falaise à chaque étape.

Exemple: Le tas de sable $c = [2, 4, 1, 0, 1, 3, 3, 1, 0, 0]$ peut devenir à l'étape suivante ou bien le tas $c = [2, 3, 2, 0, 1, 3, 3, 1, 0, 0]$ ou bien le tas $c = [2, 4, 1, 0, 1, 3, 2, 2, 0, 0]$.



Le processus évolue jusqu'à ce qu'on arrive à un état stable, c'est-à-dire un tas n'ayant plus aucune falaise, dont ne pouvant pas se modifier.

Exemple: Le tas $c = [2, 4, 1, 0, 1, 3, 3, 1, 0, 0]$ peut arriver au bout de 5 étapes à un état stable.



Travail demandé:

a. Partie traitement

Dans un premier temps, on ne s'occupe pas de la représentation graphique du tas de sable par un diagramme. Un tas de sable sera simple représenté par une liste d'entiers

$$c = [c_0, \dots, c_{k-1}]$$

1) Ecrire la fonction **initTas(k)** qui, étant donné un entier k , renvoie une liste de longueur k dont tous les éléments sont des entiers nuls à l'exception du premier qui vaut k .

Ex : `initTas(3)` renvoie `[3,0,0]`.

2) Ecrire la fonction **falaise(tas)** qui, étant donné une liste d'entiers qui représente un tas de sable, retourne la liste des positions où il y a une falaise.

Exemple : `falaise([2,4,1,0,1,3,3,1,0,0])` retourne `[1,6]`

3) Ecrire la fonction **etape(tas)** qui, prenant en entrée une liste d'entiers représentant un tas, retourne False si le tas n'a pas de falaise et sinon calcule le tas à l'étape suivante (c'est-à-dire le nouveau tas obtenu après la chute d'un grain d'une des falaises) et retourne True (on pourra prendre la première falaise ou la dernière ou une au hasard). Le tas a donc éventuellement été modifié par l'appel de cette fonction.

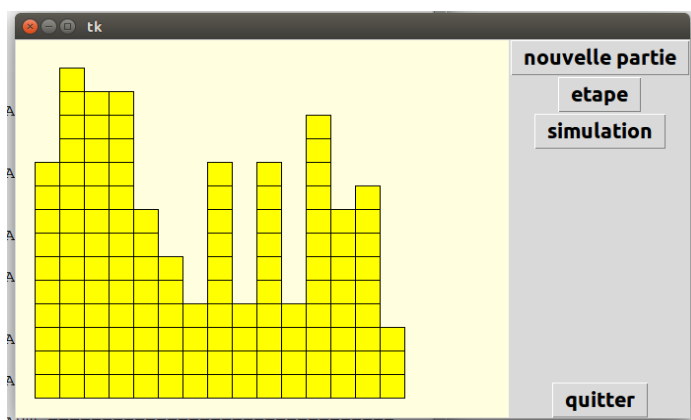
b. Partie interface:

La fenêtre de l'application se présente de la façon suivante:

→ Un canevas sur lequel est dessiné le tas de sable.

→ Un bouton « etape » sur lequel l'utilisateur peut cliquer pour simuler une étape de calcul, c'est-à-dire pour obtenir l'affichage du nouveau tas de sable suite à la chute d'un grain.

→ Un bouton « simulation » qui lance le programme jusqu'à arriver à un état stable du tas. Toutes les itérations s'exécutent alors de manière automatique. On peut contrôler la vitesse de l'animation en utilisant la fonction **sleep(seconds)** du module **time** et garantir la mise à jour du canevas avec la méthode **update()** du canevas.



1) Mettre en place les différents widgets

2) Ecrire la fonction **dessinegrain(i,j)** qui dessine le j-ième de la colonne i.

3) Ecrire la fonction **dessinecol(i,ci)** qui dessine la colonne i contenant ci grains.

4) Ecrire la fonction **dessine(tas)** qui dessine le tas de sable tas.

5) Réaliser le programme complet (pour cette première version, on utilisera initTas pour le tas de départ)

c. Amélioration :

Mettre en place un bouton « nouvelle partie » : à chaque clic sur ce bouton, le dessin précédent s'efface, un nouveau tas est généré aléatoirement (on fabriquera une liste aléatoire) et s'affiche.

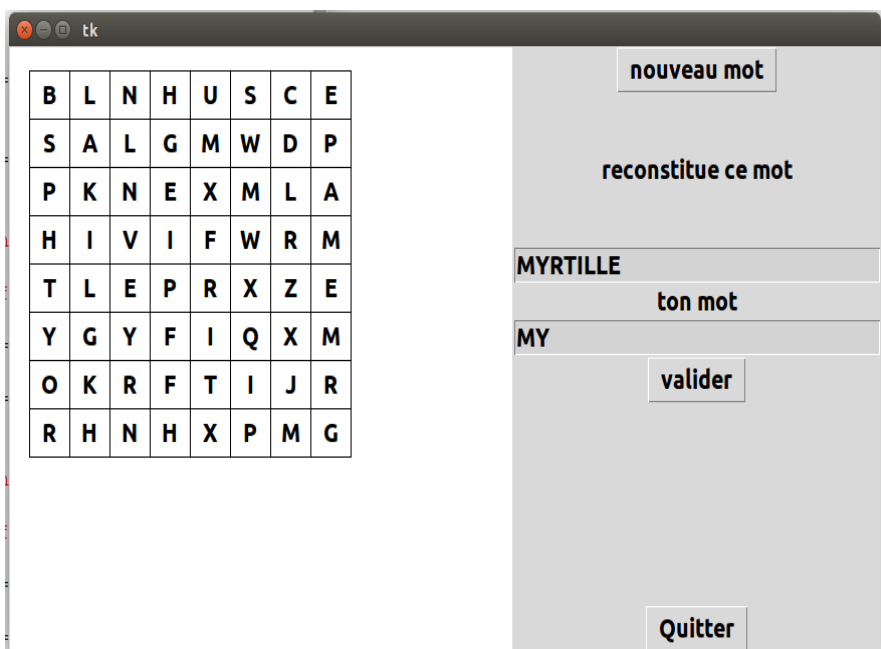
Exercice 8 - Petit jeu de lettres avec Tkinter

Il s'agit de réaliser une interface Tkinter pour programmer un petit exercice sur des lettres destiné à des écoliers: on va afficher une grille de lettres et un mot à reconstituer. L'enfant doit cliquer sur les lettres de la grille pour ré-écrire le mot proposé. Au fur et à mesure des clicks, les lettres doivent s'afficher; l'utilisateur clique ensuite sur valider quand il pense avoir fini.

Si la réponse est le mot cherché il doit s'afficher bravo en vert ou faux en rouge.

Le jeu commencera lorsque l'enfant clique sur jouer: à ce moment là, la grille de lettres s'affiche et l'enfant peut commencer à cliquer.

Chaque nouveau click sur jouer doit faire s'afficher une nouvelle grille et un nouveau mot.



Partie 1:

Mise en place des widgets permettant de faire cette interface (sans dessin de la grille, juste les widgets).

Partie 2:

On suppose avoir en variable globale le mot="ABRICOT" et une grille de lettres grillemot contenant les lettres de ce mot. On représentera la grille de lettres par une liste de lignes, chaque ligne étant une liste de lettres.

mot = "ABRICOT"

grillemot=[['Y', 'G', 'N', 'T', 'R', 'A', 'K', 'Q'], ['E', 'V', 'K', 'A', 'F', 'A', 'L', 'Q'], ['B', 'I', 'S', 'U',

'E', 'R', 'X', 'E'], ['R', 'Z', 'E', 'O', 'A', 'B', 'Z', 'O'], ['P', 'K', 'L', 'W', 'X', 'L', 'Y', 'Y'], ['H', 'W', 'L', 'V', 'T', 'O', 'W', 'G'], ['B', 'I', 'O', 'R', 'Z', 'X', 'Q', 'C'], ['K', 'L', 'Y', 'J', 'B', 'T', 'N', 'O']]

remarque: on trouvera sur la page web usuelle le fichier donnéesTP8 qui contient une liste de mots et une grille pour ABRICOT (ainsi que la liste alpha -voir plus loin-).

a. Ecrire la fonction de commande du bouton jouer qui affiche la grille de lettres.

Pour écrire dans le canvas can on utilisera une instruction du style:

`can.create_text(x,y,text="A",font=("helvetica",15,"bold"))`

qui écrit la lettre "A" au point de coordonnées (x,y)

b. Ecrire la fonction `ecrire(event)` qui, lorsque l'élève clique sur une lettre de la grille, affiche cette lettre dans la zone de saisie.

c. Ecrire la fonction qui est la commande du bouton valider

d. Mettre au point le jeu pour ce mot et cette grille.



Partie 3:

On va maintenant créer une grille aléatoire mais contenant forcément le mot à trouver.

Ecrire la fonction `cree_grille(mot)` qui étant donnée un mot maximum huit lettres va créer une grille 8x8 contenant les lettres de ce mot complétées par des lettres choisies aléatoirement dans la liste `alpha=['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']`

On pourra commencer par créer une liste de 64 lettres dont celles du mot en complétant par des lettres prises aléatoirement puis en mélangeant cette liste avant de la mettre dans une grille.

On peut maintenant enchaîner les jeux à partir d'une liste de mots. Chaque appel de jouer doit produire un nouvel affichage....

Partie 4:

Améliorer le jeu..... (compter les points, tests, empêcher l'utilisateur de faire n'importe quoi, laisser l'utilisateur corriger une erreur.....)

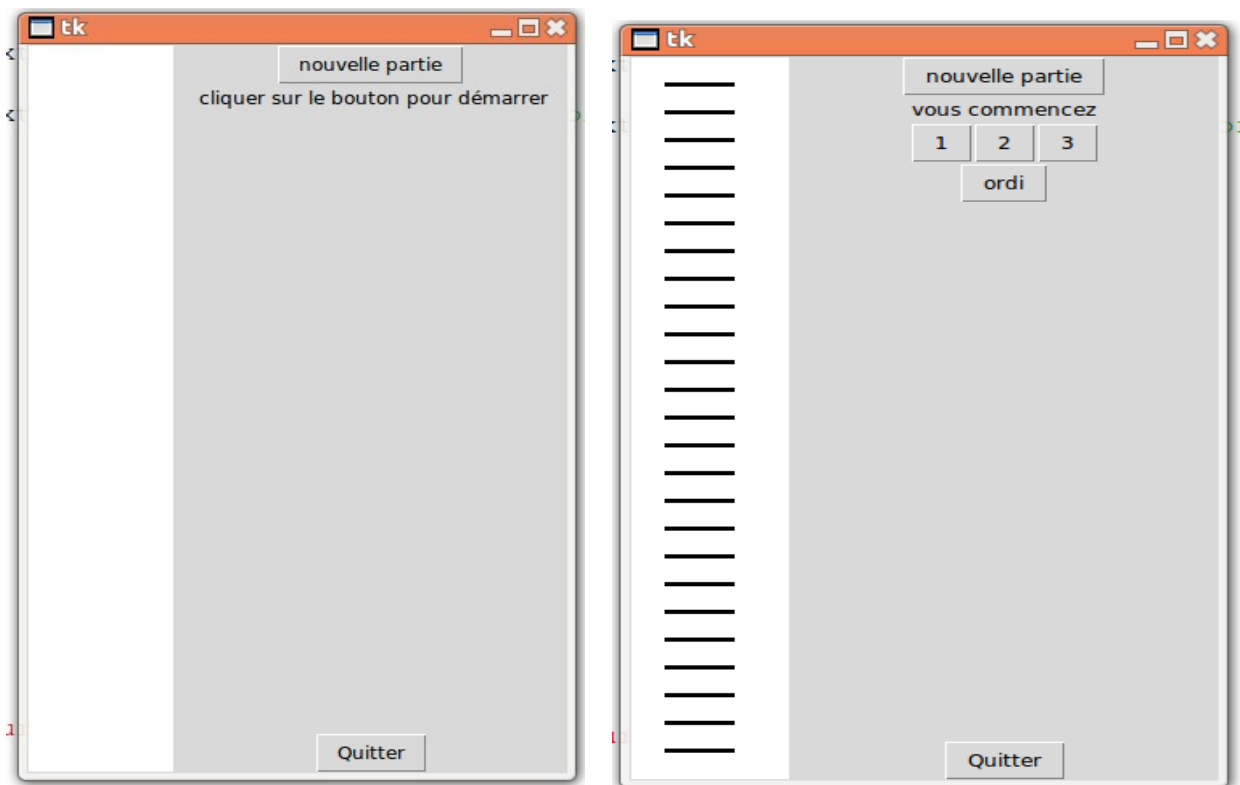
Exercice 9 - Jeu des allumettes

Il s'agit de réaliser une interface Tkinter pour jouer au jeu des allumettes. Ce jeu très connu et dont il existe de nombreuses variantes se jouera de la manière suivante: on dispose d'un certain nombre d'allumettes (ce sera 25 par défaut) .

Chacun son tour, un des deux joueurs prend une deux ou trois allumettes selon son choix. Celui qui est obligé de prendre la dernière allumette perd la partie.

Il est demandé de faire un programme avec Tkinter pour jouer à ce jeu: un joueur joue contre l'ordinateur (qui pourra jouer de manière aléatoire ou avec stratégie).

On aura donc l'interface suivante pour démarrer la partie:



- Le joueur qui commence la partie doit être tiré au sort.
- le marquage 1,2,3 des boutons indique le nombre d'allumettes à retirer
- le bouton ordi déclenchera le coup de l'ordinateur.

A la fin de la partie, l'ordinateur annonce le gagnant et permet de recommencer (avec le bouton nouvelle partie)

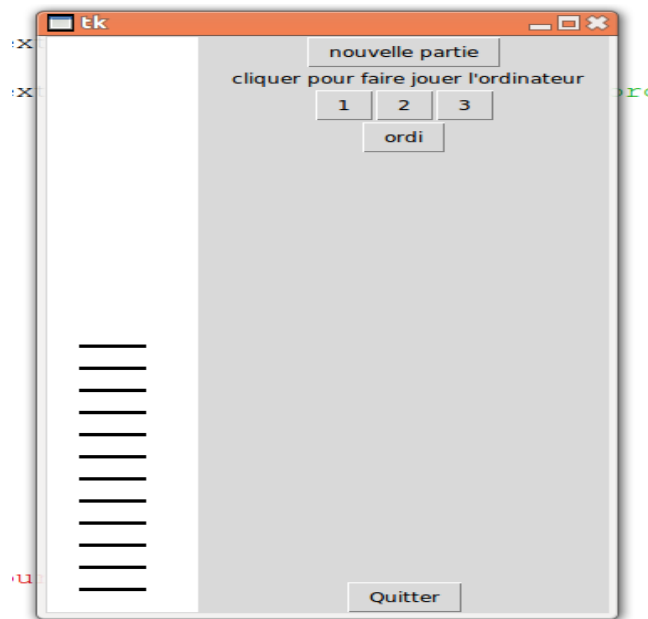
Remarques:

→ plutôt que d'effacer les allumettes, on les redessinera à chaque fois (avec le nombre restant d'allumettes qui sera en variable globale)

→ dès le départ on testera de ne pas tenter de retirer plus d'allumettes que ce qui reste

→ au départ on fera jouer l'ordinateur au hasard. Ensuite on pourra mettre au point la stratégie gagnante: pour gagner, il faut laisser sur le tas un nombre d'allumettes congru à 1 modulo 4.

→ on supposera que le joueur joue correctement (il ne prend pas la place de l'ordi, il ne joue pas plusieurs fois de suite)



Conseils:

1. mettre en place les différents widgets et les variables nécessaires (joueur , nombre d'allumettes , nombre d'allumettes restantes)
2. définir les fonctions de boutons (elles ne font rien au départ)
3. positionner les widgets qui seront à l'écran au départ et mettre le positionnement des autres dans les fonctions de boutons
4. écrire la fonction qui dessine les allumettes (fonction qui n'est pas une commande de bouton)
5. écrire la fonction jouer(k) qui fait jouer k allumettes (que ce soit le joueur 1 ou le joueur 2 qui la déclenche). Cette fonction teste qu'il reste au moins k allumettes, si oui elle retire k allumettes et redessine les nouvelles allumettes puis elle teste s'il reste alors des allumettes: si oui elle indique à qui c'est de jouer sinon elle annonce qui a gagné (c'est donc la fonction qui gère les variables globales du nombre d'allumettes et du joueur en cours)
6. compléter les fonctions de boutons des boutons lancer, 1, 2, 3 et ordi
7. mettre en place une première version du jeu
8. jouer et constater les défauts: on peut tricher; mettre en place l'activation/désactivation des boutons pour que le joueur ne puisse jouer que lorsque c'est son tour

On pourra ensuite mettre en place une stratégie pour l'ordinateur, proposer de compter des points en enchaînant des parties, changer le nombre d'allumettes, faire un mode deux joueurs humains.....

On pourra aussi modifier la déco.....

Exercice 10 – Déplacements dans une grille

Partie A : mode console

Il s'agit de faire en mode console un petit « jeu » : l'utilisateur doit programmer des déplacements dans une grille pour aller chercher des trésors en évitant des obstacles..... L'idée n'est pas de se déplacer au fur et à mesure mais de prévoir tous les déplacements à l'avance.

On travaillera avec une grille carrée de taille n , une grille sera une liste de listes.

Question1 : Ecrire la fonction `creeGrille(n,val)` qui renvoie une grille carrée de taille n où toutes les « cases » contiennent `val`.

Question2 : Ecrire la fonction `affiche(grille)` qui affiche en mode console la grille. Il ne doit plus rester de `[]` dans l'affichage (voir exemple ci-dessous).

Question3 : On veut placer aléatoirement un certain nombre de trésors et un certains nombre de fantômes.

a. Ecrire une fonction qui renvoie la liste de toutes les cases.

b. Ecrire la fonction `aleagrille(n,p)` qui crée une grille carrée de taille n puis choisit p cases et mets dans ces cases au hasard soit 1 soit 0 et renvoie la grille ainsi obtenue

Ecrire la fonction

```
>>> aleagrille(8,10)
[[['_','_','_','_','0','_','_','1','_'],['_','_','_','1','_','_','_','_'],['_','_','_','_','_','_','_','_'],['_','_','_','_','_','_','_','1'],['_','_','_','_','_','_','_','_'],['_','_','_','_','_','_','_','_'],['_','_','_','_','_','_','_','_'],['_','_','_','_','_','_','_','_']]
>>> affiche(aleagrille(8,10))
__1____
____0___
_____1__
__0_0___
10__0_0_
_____0___
_____
```

Question 4 : On indiquera un déplacement avec l'un des caractères 'H' (haut) 'B' (bas) 'D' (droite) ou 'G' (gauche).

Ecrire la fonction `deplacement(grille, dep, i, j)` qui effectue le déplacement `dep` à partir de la case `i,j` : la fonction doit vérifier que le déplacement ne fait pas sortir de la grille et elle doit modifier la grille (en changeant * qui représente la position du joueur de case). A la fin la fonction renvoie la position du joueur (nouvelle ou inchangée).

Dans un premier temps on ne se préoccupe pas des contenus des cases.

Question 5 : On va maintenant gérer une suite de déplacements cette suite étant sous forme d'une string « HGGDD » par exemple.

Ecrire la fonction `suitedeplacements(grille, i,j,ch)` qui effectue la suite de déplacements `ch` à partir de la case `(i,j)` en utilisant la fonction précédente . A la fin, la position doit être renvoyée.

Question 6 : faire une fonction pour une partie :

l'ordinateur doit générer une grille l'afficher et demander à l'utilisateur sa suite de déplacement, le joueur partant de la case (0,0). Ensuite ces déplacements doivent être effectués un par un par la machine avec un affichage à chaque fois de la grille.

Question 7 : On va maintenant modifier les fonctions qui précèdent pour gérer des points. Au départ l'utilisateur a 0 point.

S'il passe sur un trésor (cases 0) il gagne 5 points (une seule fois le trésor une fois pris disparaît), s'il passe sur un fantôme (cases 1, idem le fantôme « disparaît ») il perd 10 points et s'il « tente » de sortir de la grille il perd 2 points.

Pour gérer cela on fera renvoyer à la fonction déplacement le tuple (i,j,p) où p est le nombre de points correspondant à ce déplacement et le tuple (i,j,total) à la fonction suite déplacements où total est le nombre de points correspondant à la suite de déplacements.

On pourra rajouter un affichage des gains/pertes au fur et à mesure.

À la fin de la séquence de déplacements, l'utilisateur a gagné s'il est revenu à la case (0,0) avec un nombre de points positifs qu'on affichera.

Dans un premier temps on pourra faire les tests avec une grille de taille 8 avec 10 objets.

Question 8 : améliorations :

- faire choisir la taille et le nombre d'objets à l'utilisateur
- tester la suite de déplacements donnée par l'utilisateur avant de lancer le déplacement pour voir si la saisie est correcte
- regarder s'il reste des trésors à la fin du parcours et l'indiquer à l'utilisateur
- etc....

exemple de début de partie :

```
* _ _ _ _ _ 1 _
_ 1 _ _ _ _ 0 _
_ _ _ 0 _ _ _ _
0 _ _ _ _ _ _ _
_ _ _ 1 _ _ _ _
_ _ _ _ 0 _ 1
_ _ _ _ 0 0 _
_ _ _ _ _ _ _ _
```

votre suite de déplacements :

BBBDDDDHDDBBBBBBDHHHHHHHHHHHHHGGGGGGGGGG

```
_ _ _ _ _ 1 _
* 1 _ _ _ _ 0 _
_ _ _ 0 _ _ _ _
0 _ _ _ _ _ _ _
_ _ _ 1 _ _ _ _
_ _ _ _ 0 _ 1
_ _ _ _ 0 0 _
_ _ _ _ _ _ _ _
```

etc..... et on termine avec

bravo vous êtes sortis avec 4 points

Partie B : mode Tkinter

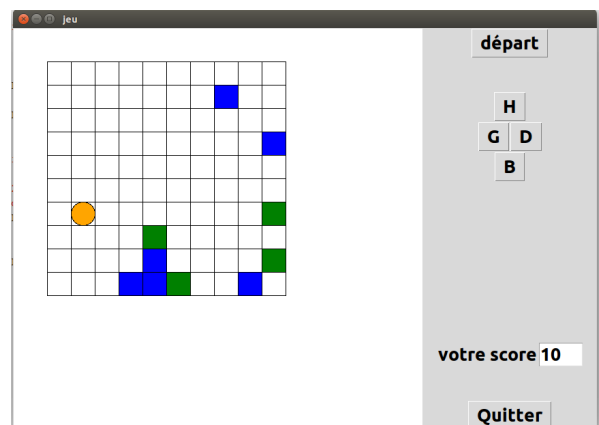
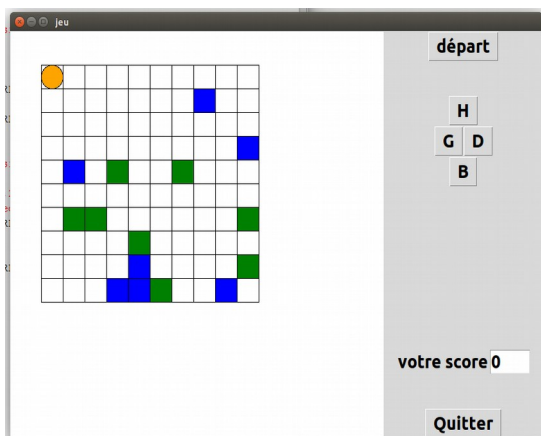
Il s'agit de faire, avec une interface graphique, une autre version du petit « jeu » programmé ci-dessus : ici l'utilisateur doit toujours se déplacer dans une grille mais d'une part, les déplacements se feront avec des clics sur des boutons de direction et d'autre part les déplacements se feront au fur et à mesure ; il y a toujours des fantômes et des trésors....

Bien entendu vous allez ré-utiliser un certain nombre de fonctions de la première partie, notamment tout ce qui concerne la gestion des grilles .

L'interface se présente de la manière suivante au départ :



Un clic sur le bouton de départ fait apparaître une grille avec la position de départ (cercle orange) les fantômes (cases bleues) et les trésors (cases vertes) ainsi que les boutons de déplacement et le score.



Pendant le jeu, le score est mis à jour au fur et à mesure

remarques :

- sur la capture la grille est de taille 10 avec 15 objets mais cela doit être paramétrable
- les cases sont de taille 35 mais cela doit être modifiable (mettre en variable globale)

Question 1 Mettre en place les widgets nécessaires.

Question 2 Écrire la fonction `dessineGrille(m)` où `m` est une matrice carrée comme celle de la partie 1A. Les cases contenant des trésors seront en verts, celles contenant des

fantôme seront en bleues et celle contenant la position du joueur contiendra un rond orange (case où il y a '*' dans la grille).

Question 3 Écrire les fonctions qui sont les commandes des boutons de déplacement. On utilisera la fonction `deplacement(m,d,i,j)` de la partie 1A. A chaque étape la grille est redessinée avec les nouvelles valeurs. On pourra avoir une variable globale contenant la matrice en cours et une pour la position du joueur.

Question 4 Écrire la fonction qui lance une partie.

Question 5 Mettre en place la gestion des scores (il n'y a pas forcément besoin d'une variable globale).

Une version simple de ce jeu doit alors être en place.

Remarque : pour l'instant il n'y a pas de fin de jeu prévue..... à part le bouton quitter.

Partie C : les extensions

Choisissez votre énoncé.....

→ Énoncé A

1. fin de jeu :

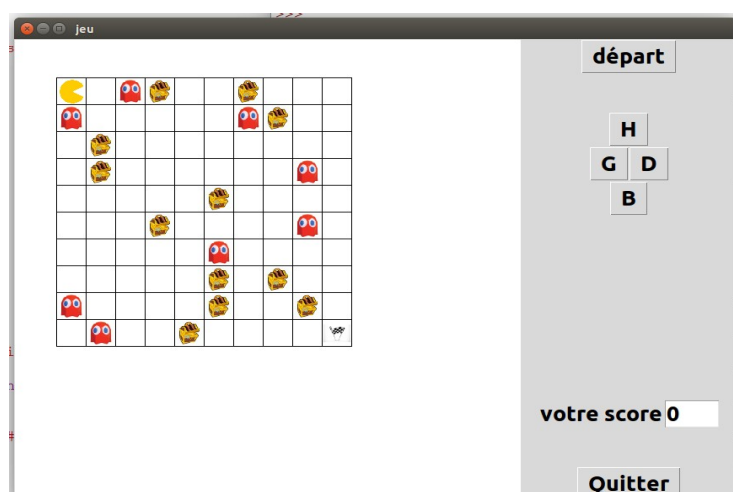
On commencera par la mise en place d'une fin de jeu : dans votre cas, le jeu s'arrête dès que le joueur passe dans la case en bas à droite et uniquement dans ce cas. Dès que le joueur arrive dans la case, le jeu s'interrompt : le joueur ne doit plus pouvoir faire de déplacement tant qu'il n'a pas cliqué sur le bouton « départ » ; de plus pendant le jeu, il ne doit pas pouvoir cliquer sur départ tant qu'il n'est pas arrivé à la case en bas à droite.

Mettre en place cette fin de jeu (vous avez le choix de la méthode : faire disparaître les boutons ou les rendre inactif ou les bloquer ou modifier les fonctions de jeu ou....)

On fera un affichage indiquant au joueur que la partie est finie et qu'il a gagné ou perdu selon le cas où son total de points est positif ou négatif.

2. dessin:

On veut faire une application plus « jolie » en remplaçant les carrés ou ronds de couleur par des images comme sur la capture ci-dessous. Vous trouverez 4 images (taille 35) dans le dossier imagesTP sous moodle. Faites les modifications nécessaires.



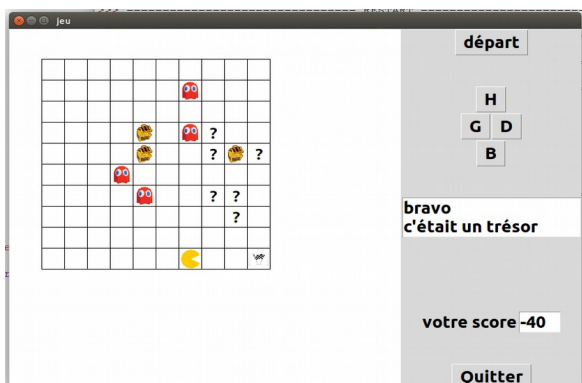
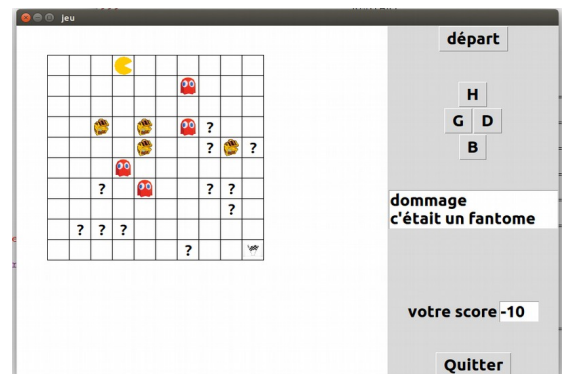
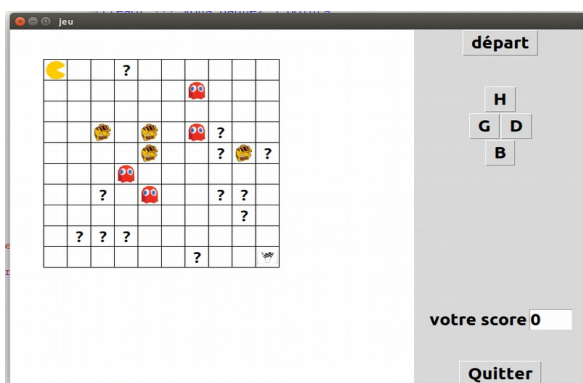
3. cases surprises

On rajoute des cases où il y a des « surprises » : dans une case surprise, il y a soit un trésor soit un fantôme mais l'utilisateur ne sait pas quoi!

Pour l'affichage, on écrira un ? dans la case (on pourra utiliser `can.create_text`). Dans la grille qui sert pour le jeu, on mettra des 2 pour les trésors cachés et des 3 pour les fantômes cachés. En tout il doit y avoir p cases occupées par 0, 1, 2 ou 3.

Modifier la fonction `aleagrille` et la fonction d'affichage pour gérer ces nouvelles cases.

Quand on passe sur une cases surprise, le contenu est dévoilé (il faut faire un affichage) et les points sont gérés.



Modifier les fonctions de déplacements pour gérer ces nouvelles cases.

Quand on arrive sur la case d'arrivée les cases surprises sont dévoilées (avec un tour cyan pour repérer les surprises) et un bilan des trésors restants est affiché (voir capture ci-dessous)

Question bonus :

On rajoute une case spéciale qui n'est pas apparente et qui ne contient ni trésor ni fantôme ni surprise ni départ ou arrivée. Dans la grille de jeu on mettra un symbole par exemple '\$' (mais à l'affichage la case sera vide).

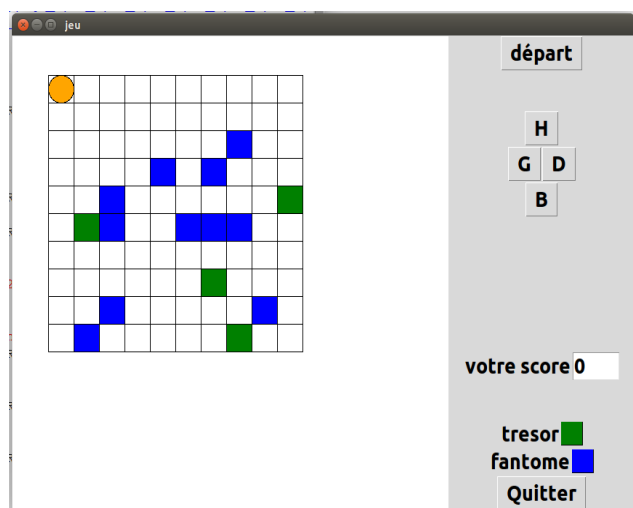
Si on l'utilisateur passe sur cette case une annonce est faite à l'utilisateur et tous les trésors et fantômes sont inversés (une case fantôme devient une case trésor et réciproquement).

Faites les modifications nécessaires.

→ Enoncé B

1. Affichage

On va afficher deux petits carrés de couleur pour savoir ce qui est trésor et ce qui est fantôme. On pourra faire comme dans la capture d'écran ci-dessous.



2. Fin du jeu

On va rajouter une fin de jeu. Dans votre cas le jeu est fini quand il n'y a plus de trésors.

- Ecrire une fonction `encoreTresors(m)` qui teste s'il reste des trésors dans la grille.
- Modifier le jeu pour mettre en place la fin du jeu.

Dès que le joueur a atteint tous les trésors le jeu s'interrompt : le joueur ne doit plus pouvoir faire de déplacement tant qu'il n'a pas cliqué sur le bouton « départ » ; de plus pendant le jeu, il ne doit pas pouvoir cliquer sur départ tant qu'il n'a pas atteint tous les trésors.

Mettre en place cette fin de jeu (vous avez le choix de la méthode : faire disparaître les boutons ou les rendre inactif ou les bloquer ou modifier les fonctions de jeu ou....)

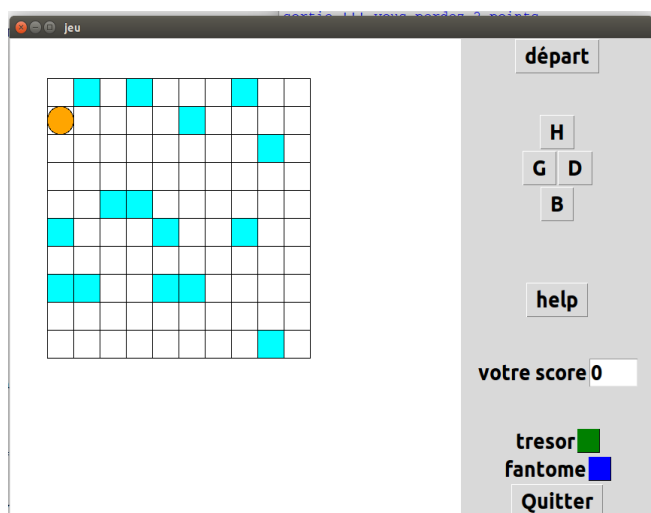
- On rajoutera un affichage indiquant au joueur que la partie est finie et qu'il a gagné ou perdu selon le cas où son total de points est positif ou négatif. Vous pouvez faire l'affichage dans le canvas avec `can.create_text` (si `can` est le nom du canvas) ou dans une zone de la partie droite de la fenêtre.



3. Jeu de mémoire

On va faire une nouvelle version du jeu. Quand on clique sur le bouton départ l'affichage est comme avant. Mais dès qu'on clique sur un des boutons de déplacement, les trésors et fantômes sont tous représentés à l'écran par des cases colorées en cyan (voir capture), l'utilisateur doit se mémoriser où étaient les trésors..... Mettre en place cette nouvelle façon de jouer.

On rajoutera ensuite un bouton help. Si on clique dessus les trésors/fantômes reprennent leurs couleurs.... Jusqu'au prochain click sur un bouton de déplacement. On pourra avoir un booléen qui prendra la valeur False quand on met tout en cyan et True quand on met les bonnes couleurs.



On pourra rajouter une pénalité : chaque click sur help enlève 15 points.

→ Enoncé C

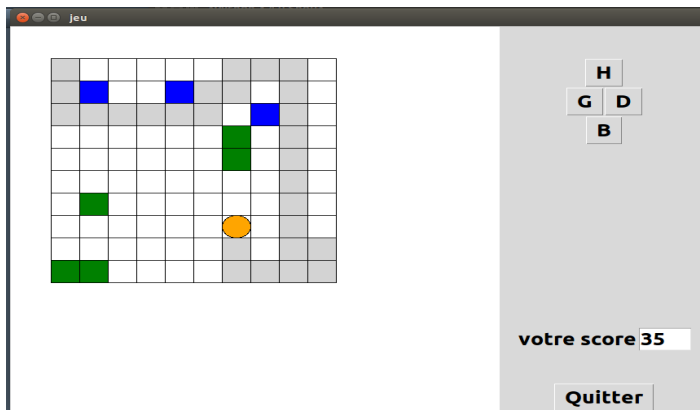
1. Cases déjà visitées:

On rajoute une règle : il est interdit de passer deux fois dans la même case. On va donc « marquer » les cases déjà visitées.

Au fur et à mesure des déplacements, on mettra un symbole particulier (par exemple 'v')

dans la grille de jeu et à l'écran les cases déjà visitées seront grisées (voir capture).

Chaque tentative de revenir dans une case grise devra faire perdre un point à l'utilisateur (et celui-ci ne bougera pas).



2. Possibilité de recommencer avec les mêmes fantômes/trésors

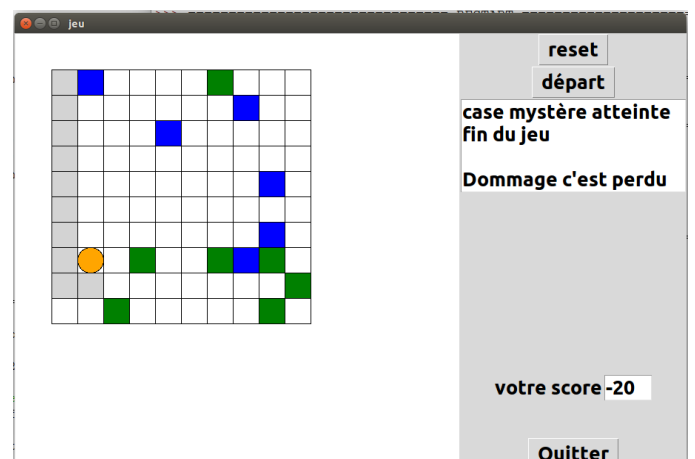
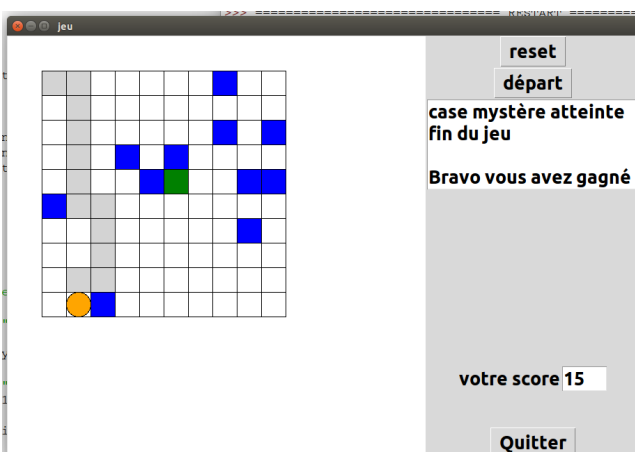
Avec cette nouvelle règle, il faut imaginer des meilleurs chemins. On va donc proposer à l'utilisateur de recommencer la partie en cours avec les mêmes trésors et fantômes. Il faudra donc garder en mémoire la grille de départ (on pourra utiliser la fonction `deepcopy` du module `copy`). Il faudra aussi rajouter un bouton spécial « reset » qui remettra le score à 0 et permettra de jouer avec les mêmes trésors/fantômes que la partie en cours.

3. Fin de jeu :

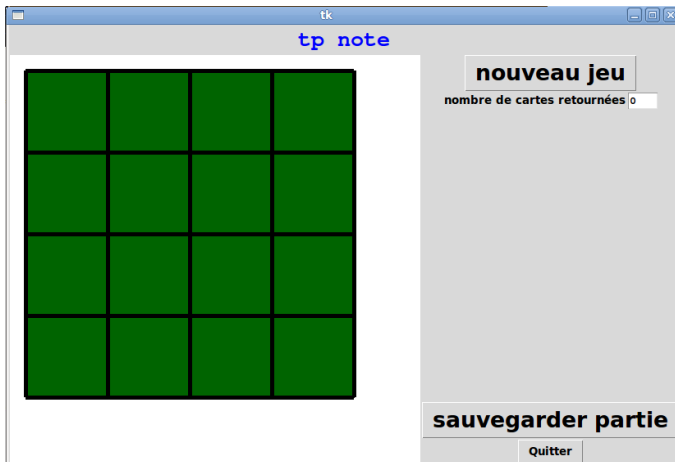
On va maintenant mettre en place une fin de jeu : dans votre cas, le jeu s'arrête dès que le joueur revient dans une case « mystère ». Cette case est choisie aléatoirement en début de partie (elle ne peut pas contenir trésor ou fantômes) mais reste « cachée ». Dès que le joueur arrive dans la case, on le prévient et le jeu s'interrompt : le joueur ne doit plus pouvoir faire de déplacement tant qu'il n'a pas cliqué sur le bouton « départ » ; de plus pendant le jeu, il ne doit pas pouvoir cliquer sur départ tant qu'il n'est pas revenu cette case mystère.

Mettre en place cette fin de jeu (vous avez le choix de la méthode : faire disparaître les boutons ou les rendre inactif ou les bloquer ou modifier les fonctions de jeu ou...)

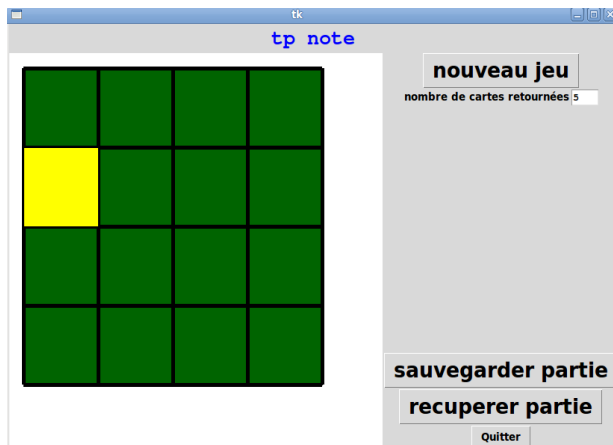
On fera un affichage indiquant au joueur que la partie est finie et qu'il a gagné ou perdu selon le cas où son total de points est positif ou négatif.



Un clic sur le bouton « nouveau jeu » fait apparaître une grille avec des cases qui sont toutes vertes foncées.



Un clic sur l'une des cases, permet de connaître la couleur cachée de cette case. Un deuxième clic (ce deuxième clic lui peut être fait n'importe où dans le canvas de la zone de la grille) fera revenir à la couleur vert foncé d'origine.... Le nombre de cases retournées doit s'afficher au fur et à mesure.



On doit avoir une possibilité de sauvegarder une partie (c'est à dire le nombre de clics déjà faits et la grille des couleurs) puis de récupérer cette partie (ou de faire un nouveau jeu) mais on ne stockera qu'une seule partie en sauvegarde et toujours dans le même fichier. Après une sauvegarde la grille s'efface et on revient à l'écran du départ.

Les boutons auront le rôle suivant:

bouton nouveau jeu: ré-initialise la matrice de référence aléatoirement ainsi que le nombre de clics et dessine la grille dans la zone de dessin

bouton sauvegarder partie : sauvegarde de la matrice et du nombre de clics dans un fichier.

On ne pourra sauvegarder qu'une seule partie à la fois et elle sera stockée toujours dans le fichier qu'on nommera partie.

bouton recuperer une partie : permet de récupérer une partie et de mettre à jour la matrice référence et le nombre de clics.

bouton quitter : comme son nom l'indique

remarque : on peut sauvegarder une partie qu'une case soit retournée ou pas. Par contre quand on récupère une partie alors à l'affichage toutes les cases sont retournées côté vert foncé.

Autres actions : il faut gérer les clics : un clic sur une des cases de la grille fait augmenter le nombre de clics de 1 et fait l'affichage à l'emplacement du clic de la couleur correspondante. Le clic suivant (qui peut être fait n'importe où dans le canvas) fait ré-apparaître le vert foncé d'origine. On pourra avoir une variable globale booléenne pour savoir si un clic a déjà eu lieu.....

Variables

On conseille d'utiliser les variables suivantes :

- nb : nombre de cases de la grille supposée carrée (grille 4 sur 4 par défaut ici)
- c: taille des cases de la grille
- x0, y0 : coordonnées du point en haut à gauche de la grille
- reference : la grille des couleurs est représentée par cette matrice (qui correspond à la grille dessinée à l'écran) . C'est donc une liste de listes (chaque sous-liste représentant une ligne). Cette variable est ré-initialisée quand on lance un nouveau jeu ou quand on récupère une partie.
- liste_couleurs : c'est la liste de toutes les couleurs utilisées pour la grille. Chaque couleur apparaissant exactement deux fois dans la matrice reference.
- Un booléen pour savoir où on en est dans les clics

Fonctions à écrire :

- écrire la fonction cree_mat(n) qui renvoie une matrice carrée (liste de listes) de taille n avec initialement toutes les valeurs à 0

```
>>> cree_mat(4)
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

- écrire la fonction aleatoire() qui renvoie une matrice où les couleurs de liste_couleur auront toutes été mises deux fois aléatoirement dans la matrice (on suppose que la taille de liste_couleur convient ; par défaut on aura 8 couleurs pour une matrice 4*4)
exemple :

```
>>> liste_couleurs=['blue','red','green','grey', 'yellow', 'cyan', 'pink', 'orange']
>>> aleatoire()
[['blue', 'blue', 'red', 'grey'], ['yellow', 'yellow', 'cyan', 'pink'], ['red', 'pink', 'orange', 'orange'], ['green', 'green', 'cyan', 'grey']]
```

- écrire une fonction qui va aider à faire la sauvegarde : on va sauvegarder 2 éléments : le nombre de clics (nombre total) et la matrice reference en cours. La fonction ecriture(mat,n,f) doit écrire n et la matrice mat dans le fichier f (n est écrit sur la première ligne et ensuite une ligne du fichier texte correspondra à une ligne de la matrice)
- écrire une fonction pour récupérer une matrice : la fonction lire(f) doit aller lire le

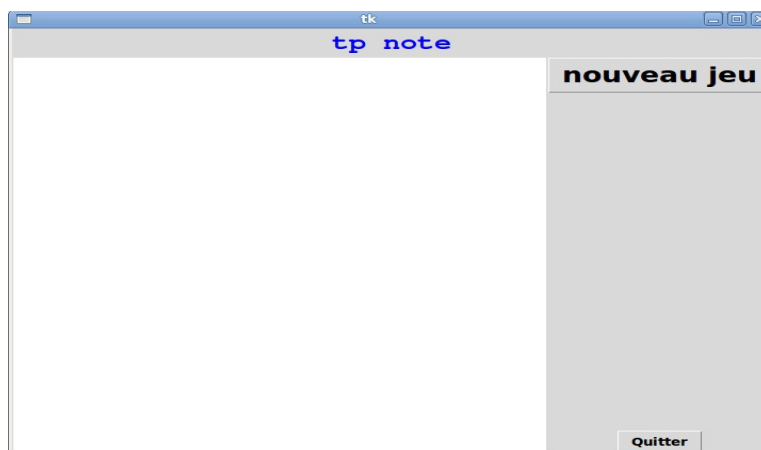
nombre de clics et la matrice écrite dans ce fichier *f* et renvoyer un tuple contenant cet entier et la matrice (liste de listes). On suppose que le fichier *f* est bien écrit.

- dessin de la grille : écrire la fonction `dessine_grille (x0,y0,c,nb)` qui dessine la grille avec toutes les cases en vert: cette grille est de *nb* cases sur *nb* cases, chaque case carrée étant de taille *c* et la grille étant dessinée à partir du point *x0,y0* (point en haut à gauche de la grille).
- la gestion des clics dans la grille : il faut écrire une fonction qui réagit à chaque clic.

Remarque: On considèrera que l'utilisateur se sert correctement du programme

Améliorations possibles

On part de l'interface réalisée précédemment pour faire un jeu de mémoire : il s'agira de retrouver dans la grille (qui n'est pas modifiée pendant une partie) des paires de cartes identiques. On part de l'interface suivante:



Un clic sur le bouton nouveau jeu provoque l'affichage de la fenêtre de la manière suivante :

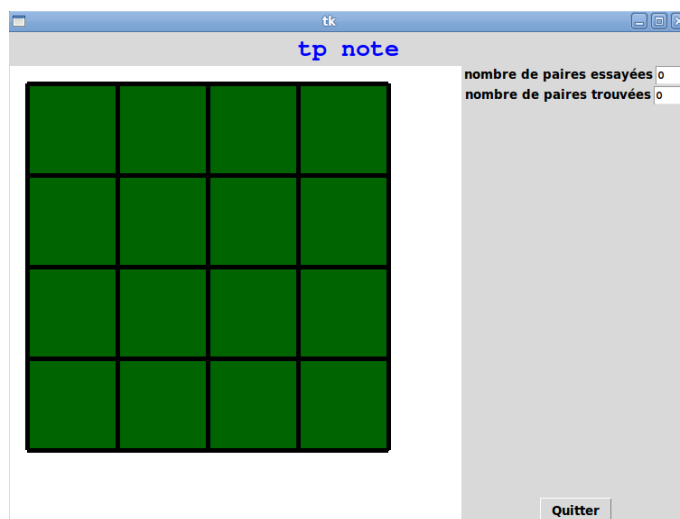


figure 1

Ensuite l'utilisateur clique successivement sur deux cases et fait se retourner deux

cartes:

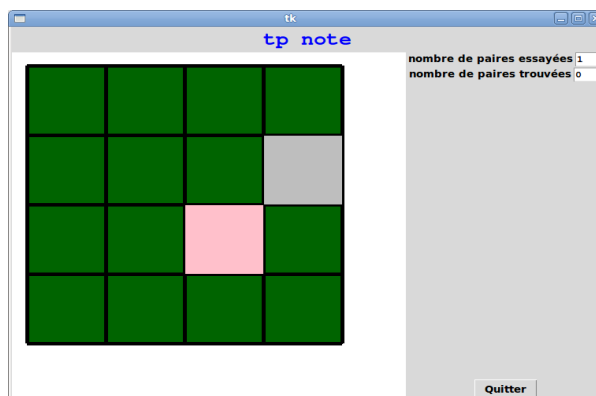


figure 2

Si on retourne deux cartes de même couleur on aura formé une paire. On gèrera à la fois le nombre de paires de cartes retournées et le nombre de paires trouvées. Les clics sont modifiés par rapport à la première partie du TP noté : on fait 2 clics successifs et c'est le 3ème clic qui fait se retourner les 2 cases.

Si on trouve une paire de cartes de la même couleur : alors un message BRAVO s'affiche sous la grille et le nombre de paires retrouvées est mise à jour.

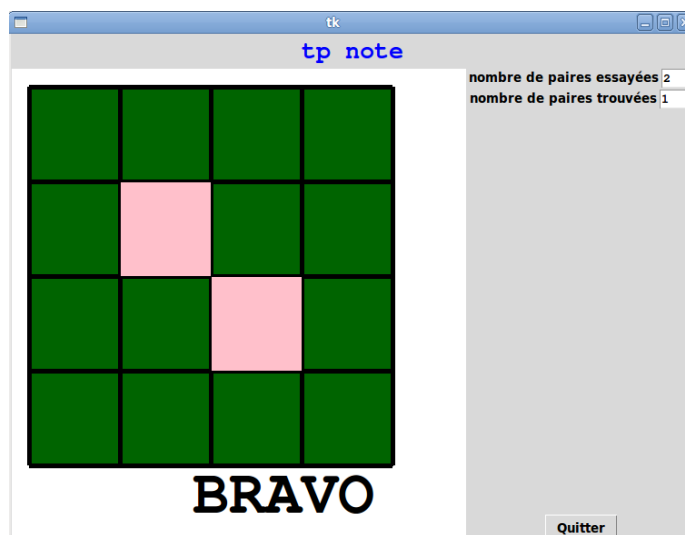


Figure 3

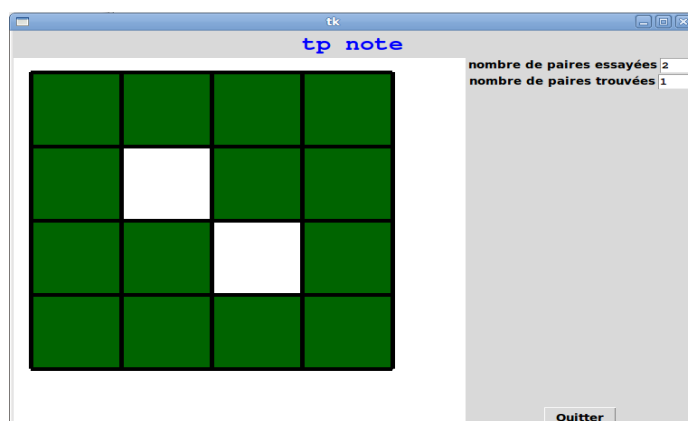


figure 4
Le clic suivant fait disparaître

l'inscription « bravo » Les deux cases « utilisées » pour former la paire vont rester blanches pour tout le reste de la partie.

Le jeu se poursuit alors, jusqu'à ce qu'on ait trouvé toutes les paires de cartes à trouver.

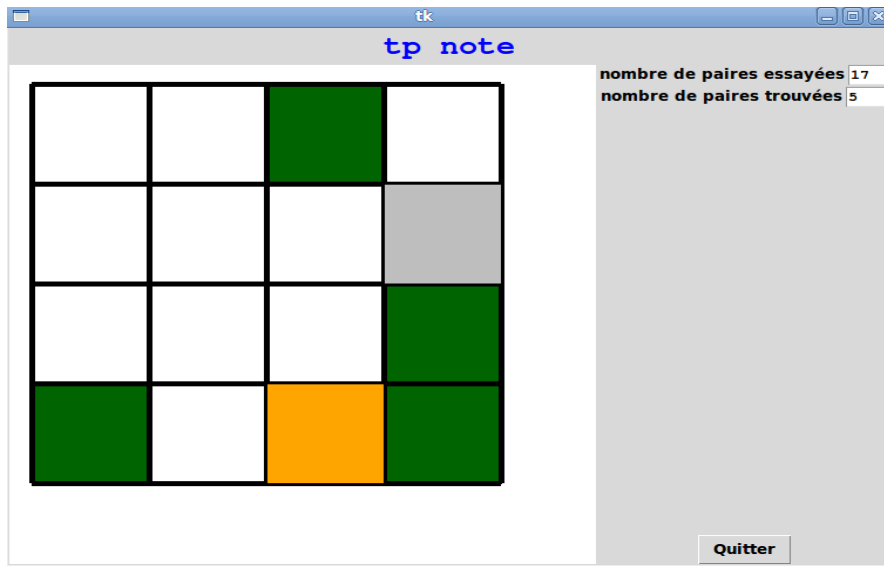


figure 5

La dernière paire retrouvée marche comme les autres sauf que le clic suivant affiche la fenêtre de la figure 7

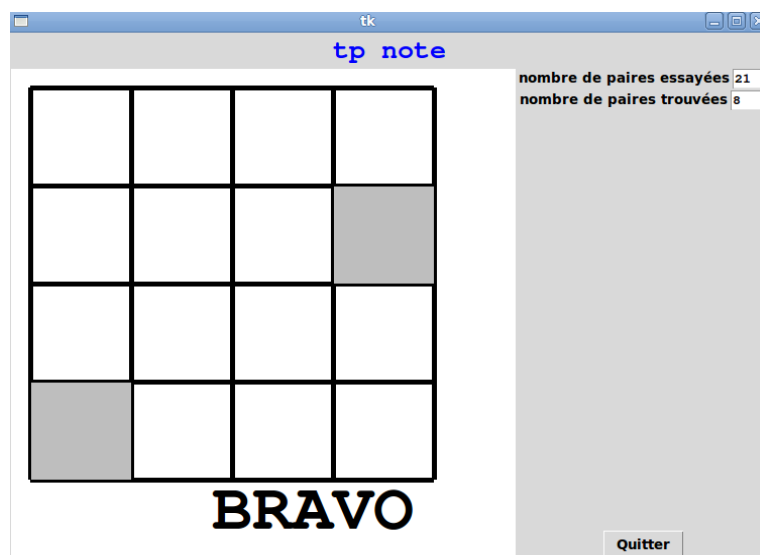


figure 6

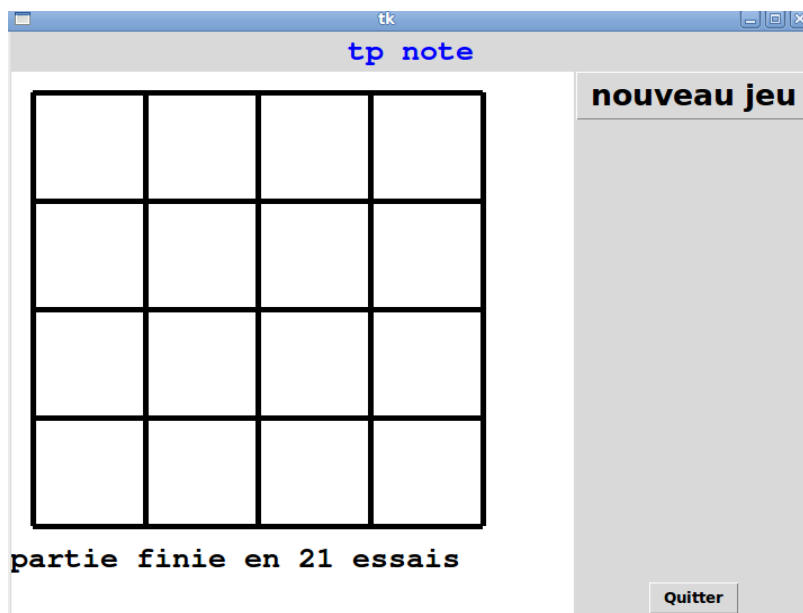


figure 7

La fin de partie est annoncée avec le score final, les nombre sde pars trouvées et essayées sont effacés et le bouton nouveau jeu ré-apparaît.

Un clic sur ce bouton refait apparaître la fenêtre de la figure1 mais avec de nouvelles données : on doit jouer avec une autre matrice la fois suivante.

Fonctions à écrire / plan de travail

A. Première partie:

Il s'agit maintenant de réaliser l'interface expliquée ci-dessus et correspondant aux dessins.

On suggère de procéder dans l'ordre suivant

→ modifier la fonction de clic pour :

cliquer deux fois pour afficher 2 cartes (c'est le 3ème clic qui fait se retourner les deux cartes)

juger si les cartes retournées sont identiques

lancer une fonction bravo() si nécessaire

→ écrire la fonction bravo()

qui déclenche l'affiche du message bravo, met à jour le nombre de paires retrouvées

→ modifier l'affichage et éventuellement la matrice de référence pour que les cartes utilisées de la matrice restent en blanc par la suite

→ il ne reste plus qu'à enchaîner tout ceci pour mettre en place l'interface demandée

→ bien gérer la fin de partie

→ ensuite il faut modifier éventuellement la commande du bouton nouveau jeu pour que tout soit bien modifié pour une partie suivante.

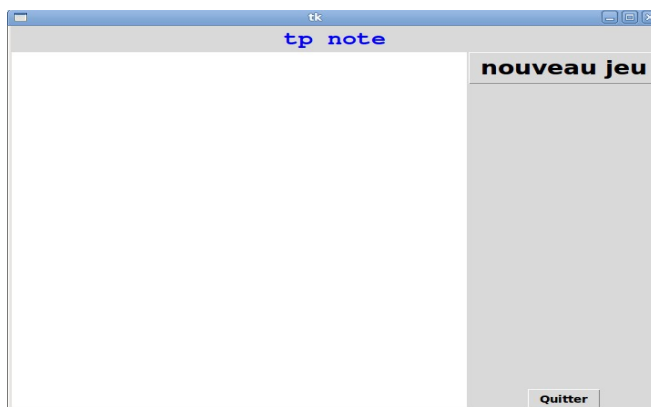
B. Améliorations possibles

Quand ceci est en place, on peut essayer d'améliorer votre programme avec par exemple les possibilités suivantes :

- permettre à l'utilisateur de choisir le nombre de cartes à découvrir c'est à dire la taille du memory (qui aura 9,16, 25 ou «36 cases)
- mettre des images à la place des couleurs
- utiliser les fonctions de sauvegarde et de récupération (que l'on devra légèrement modifier) pour pouvoir arrêter une partie en cours et la reprendre dans l'état où on l'a laissé plus tard.

→ Autre jeu

On part de l'interface suivante :



Un clic sur le bouton nouveau jeu provoque l'affichage de la fenêtre de la manière suivante :

(le bouton nouveau jeu a disparu)

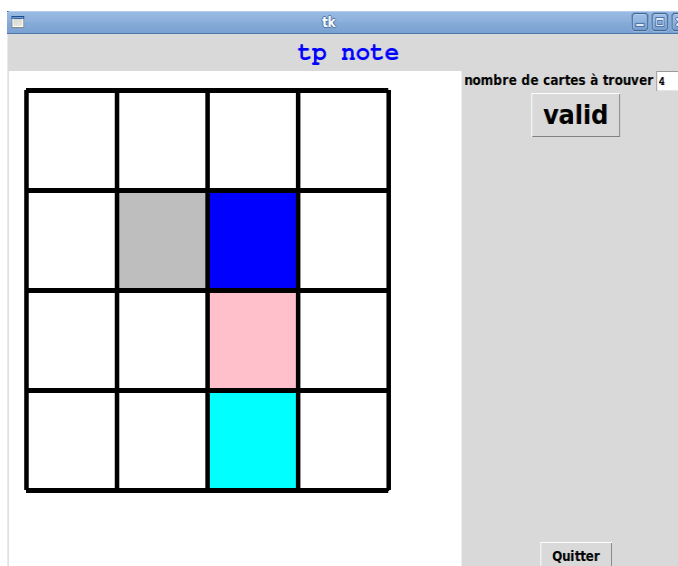


figure 1

Un bouton valid est apparu. Quand l'utilisateur pense avoir mémorisé la situation, il clique dessus. La grille verte s'affiche alors (et le bouton valid disparaît)

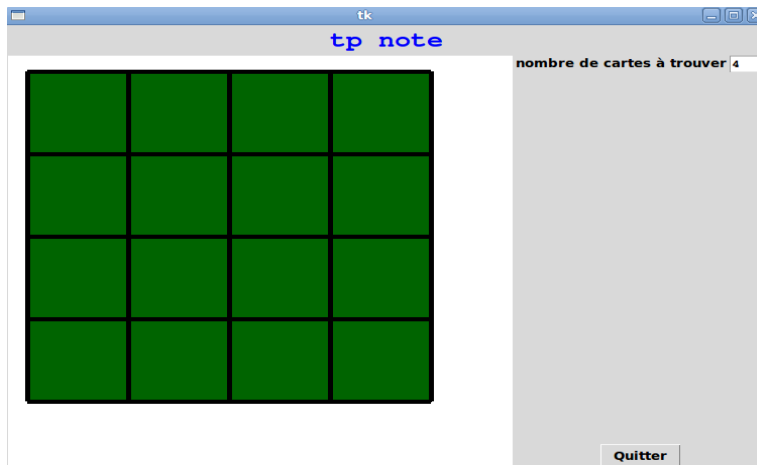
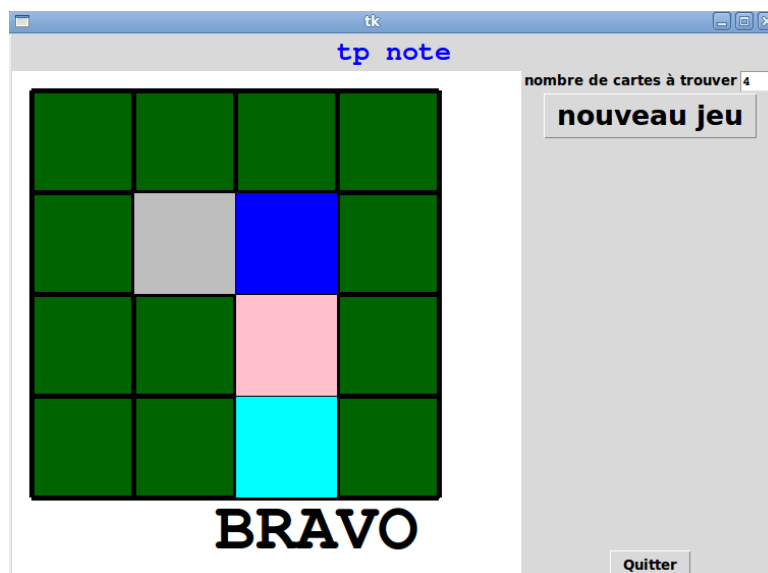


Figure 2

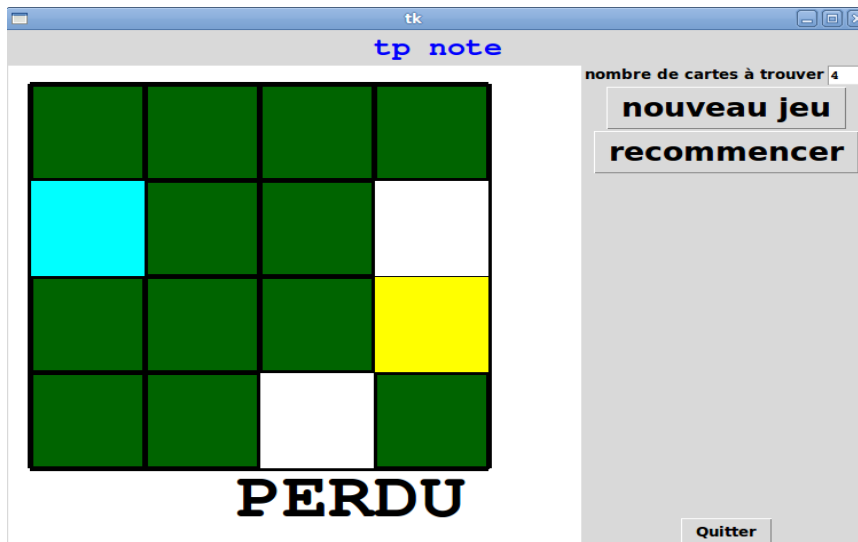
L'utilisateur clique sur les cases qu'il pense être les bonnes. Les cartes se retournent au fur et à mesure.

Au bout du nombre de clics correspondant au nombre de cases à trouver (ce sera 4 par défaut et c'est affiché en haut de la fenêtre), si les bonnes cases ont été trouvées, un message « BRAVO » doit s'afficher et le bouton nouveau jeu ré-apparaît pour pouvoir faire une nouvelle partie.



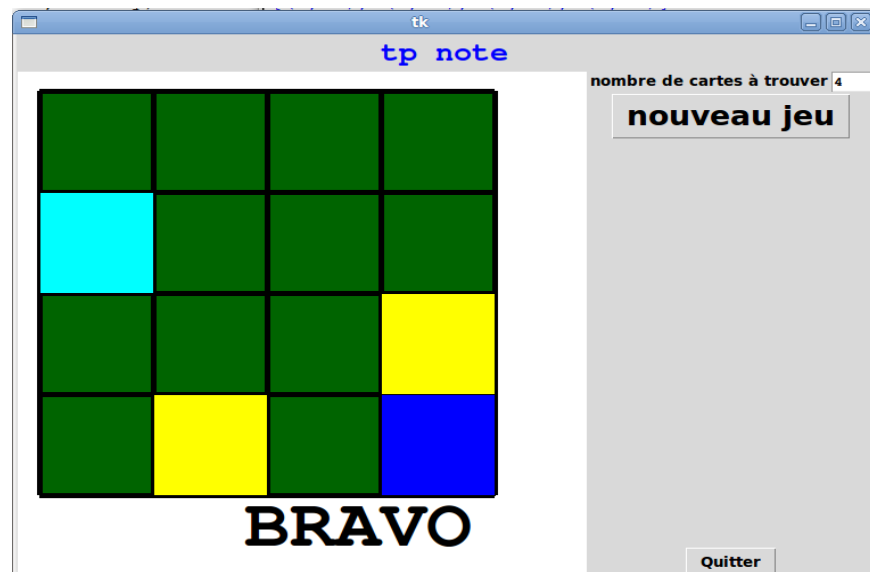
Si on relance un nouveau jeu : les cases à trouver doivent être différentes.

Si par contre l'utilisateur s'est trompé alors un message « PERDU » s'affiche. Et l'utilisateur a le choix entre le bouton « recommencer » qui lui permet de jouer avec la même grille ou le bouton « nouveau jeu » qui le fera jouer avec une autre grille. Dans les deux cas, le jeu se déroule comme auparavant.



Si on a cliqué sur recommencer on doit avoir les mêmes cases qu'auparavant.

Et là on gagne !!!



Fonctions à écrire / plan de travail

A. Première partie:

Il s'agit maintenant de réaliser l'interface expliquée ci-dessus et correspondant aux dessins.

On suggère de procéder dans l'ordre suivant :

→ écrire la fonction `matrice(n,p)` qui renvoie une matrice carrée de taille `n` contenant `p` valeurs de la liste de couleurs (valeurs quelconques y compris en double) et « white » dans toutes les autres cases.

→ mettre en place l'affichage d'une grille quelconque (matrice obtenue avec la fonction ci-dessus)

→ mettre en place le bouton valid : cette commande dessine la grille verte

fait disparaître le bouton valid

→ modifier la fonction de clic pour :

cliquer successivement sur 4 cartes qui seront retournées en même temps le 4ème clic déclenchant aussi la suite c'est à dire:

-juger la réponse

-faire l'affichage

-mettre le ou les boutons (nouveau jeu , recommencer)

→ il ne reste plus qu'à enchaîner tout ceci pour mettre en place l'interface demandée

→ ensuite il faut modifier éventuellement la commande du bouton nouveau jeu pour que tout soit bien modifié pour une partie suivante.

B. Améliorations possibles

Quand ceci est en place, on peut essayer d'améliorer votre programme avec par exemple les possibilités suivantes :

→ permettre à l'utilisateur de choisir le nombre de cartes à découvrir

→ afficher le nombre de cases justes et permettre de recommencer sans que l'affichage ne soit refait

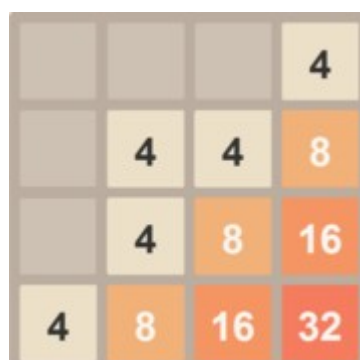
→ mettre en place un système de score global ou de record

Exercice 12 – jeu 2048

Le petit jeu à programmer est inspiré du jeu de 2048 (version jeu en ligne)

2048 est un jeu vidéo de puzzle conçu par le développeur italien Gabriele Cirulli en mars 2014. Le but du jeu est de faire glisser des tuiles sur une grille, pour combiner les tuiles de mêmes valeurs et créer ainsi une tuile portant le nombre 2048.

2048 se joue sur une grille de 4×4 cases, avec des tuiles de couleurs et de valeurs variées (mais toujours des puissances de deux) qui peuvent être déplacées quand le joueur appuie sur les touches fléchées de son clavier.



			4
	4	4	8
	4	8	16
4	8	16	32

Le gameplay du jeu repose sur l'utilisation des touches fléchées du clavier d'ordinateur pour déplacer les tuiles vers la gauche, la droite, le haut ou le bas. Lors d'un mouvement, l'ensemble des tuiles du plateau sont déplacés dans la même direction jusqu'à rencontrer les bords du plateau ou une autre tuile sur leur chemin. Si deux tuiles, ayant le même nombre, entrent en collision durant le mouvement, elles fusionnent en une nouvelle tuile de valeur double (par exemple deux tuiles de valeur « 2 » donnent une tuile de valeur « 4 »). À chaque mouvement, une tuile portant un 2 ou un 4 apparaît dans une case vide de manière aléatoire.

Le jeu, simple au début, se complexifie de plus en plus, du fait du manque de place pour faire bouger les tuiles, et des erreurs de manipulation possibles, pouvant entraîner un blocage des tuiles et donc la fin du jeu à plus ou moins long terme, selon l'habileté du joueur.

Pourtant, et bien que très chronophage, 2048 possède la particularité de ne jamais rendre l'échec frustrant, au contraire : le fait de recommencer pour tenter une nouvelle stratégie fait partie du plaisir.

La partie est gagnée lorsqu'une tuile portant la valeur « 2048 » apparaît sur la grille, d'où le nom du jeu. Quand le joueur n'a plus de mouvements légaux (plus d'espaces vides ou de tuiles adjacentes avec la même valeur), le jeu se termine (source wikipedia)

Nous allons faire un jeu qui s'inspire de ce programme mais avec des différences et vous devez impérativement suivre les questions posées.

Nous travaillerons avec des grilles toujours carrées mais qui pourront être de taille variable (6 pour les exemples ici).

Partie 1A : travail sans interface graphique

Il s'agit de faire en mode console une première version du jeu.

On travaillera avec des grilles carrées qui auront n lignes et n colonnes ; comme en cours une grille sera une liste de listes. Les fonctions réalisées fonctionneront quelque soit la valeur de n entier strictement positif.

Question1 : Écrire la fonction **creeGrille(n,val)** qui renvoie une grille avec n lignes et n colonnes où toutes les « cases » contiennent val.

Question2 : Écrire la fonction **affiche(grille)** qui étant donnée une grille fait l'affichage en mode console.

Attention ici les grilles contiennent des entiers entre 0 et 2048. Il faudra tenir compte des entiers pour que les nombres dans les colonnes ne soient pas décalés.

Exemple :

```
>>> g1
[[2, 2, 2, 2], [4, 0, 2, 8], [0, 16, 32, 0], [128, 256, 1024, 2048]]
>>> affiche(g1)
2      2      2      2
4      0      2      8
0      16     32     0
128    256   1024   2048
```

Question3 :

A. Écrire la fonction **appartient(x,g)** qui teste si l'élément x est dans une des cases de la grille g.

```
>>> appartient(1024,g1)
```

```
True
```

```
>>> appartient(512,g1)
```

```
False
```

```
>>> appartient(0,g1)
```

```
True
```

B. En déduire la fonction **gagnante(g)** qui teste si la grille g est une grille gagnante (c'est à dire qui contient 2048), et la fonction **pleine(g)** qui teste si la grille g est « pleine » c'est à dire ne contient plus de 0. Dans les deux cas, on renverra un booléen.

Question4 :

Écrire la fonction **max(g)** qui étant donnée une grille renvoie le plus grand entier présent dans la grille.

```
>>> g2=[[2, 2, 2, 2], [4, 0, 2, 8], [0, 16, 32, 0], [128, 256, 16, 32]]
```

```
>>> max(g2)
```

```
256
```

Question5 :

A. Écrire la fonction **lescases(g, val)** qui renvoie la liste des cases de la grille g qui contiennent la valeur val. Une case sera représentée par un couple de coordonnées [i, j].

B. En déduire la fonction **vides(g)** qui renvoie la liste des cases de g qui sont « vides » c'est à dire qui contiennent 0.

```
>>> vides(g1)
```

```
[[1, 1], [2, 0], [2, 3]]
```

C. Écrire la fonction **ajouteAlea(g,val)** qui modifie la grille g pour mettre la valeur val dans une case choisie aléatoirement parmi les cases vides.

D. Écrire la fonction **init(n)** qui crée une grille de taille n contenant des 0 dans toutes les cases sauf deux choisies aléatoirement qui contiennent elles soit 2 soit 4 aléatoirement (il peut y avoir 2 valeurs 2, 2 valeurs 4 ou une de chaque) ; la fonction renvoie la grille obtenue.

```
>>> g3=init(6)
```

```
>>> g3
```

```
[[0, 0, 0, 0, 2, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 4], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0]]
```

```
>>> affiche(g3)
```

```
0  0  0  0  2  0
```

```
0  0  0  0  0  0
```

```
0  0  0  0  0  4
```

```
0  0  0  0  0  0
```

```
0  0  0  0  0  0
```

```
0  0  0  0  0  0
```

Question6

Il s'agit maintenant d'écrire les fonctions **haut(g)**, **bas(g)**, **droite(g)**, **gauche(g)** qui modifient la grille g pour tenir compte d'un coup joué respectivement vers le haut, le bas, la droite ou la gauche .

On rappelle la règle : *Lors d'un mouvement, l'ensemble des tuiles du plateau sont déplacés dans la même direction jusqu'à rencontrer les bords du plateau ou une autre tuile sur leur chemin. Si deux tuiles, ayant le même nombre, entrent en collision durant le mouvement, elles fusionnent en une nouvelle tuile de valeur double (par exemple deux tuiles de valeur « 2 » donnent une tuile de valeur « 4 »). À chaque mouvement, une tuile portant un 2 ou un 4 apparaît dans une case vide de manière aléatoire.*

Exemple : une situation après quelques coups de jeu :

			2	4	8
					2
					4
					4
			2		

après un « haut » : colonne 3 : un 2 est remonté, colonne 5 les deux 4 en bas ont fusionné (c'est un 2 qui est apparu aléatoirement colonne 4). Les colonnes étant numérotées de 0 à 5.

			2	4	8
			2		2
					8
				2	

après un autre « haut » :

		4	4	4	8
				2	2
					8

un 4 est apparu aléatoirement colonne 2, les deux ont fusionné en 4 colonne 3, un 2 est remonté colonne 4.

après un « droit » :

			4	8	8
					4
					8
					4

les deux 4 les plus à droite ont fusionné en 8

				4	16
					4
					8
		4			4

puis après un autre « droit »

etc.....

Question7

Écrire la fonction qui lance une partie. On écrira la fonction **partie(n)** qui lance un jeu sur une grille de taille n. La fonction initialise le jeu puis le gère : à chaque tour la grille est affichée et on demande un mouvement à l'utilisateur (ce mouvement pouvant être H, G, D, ou B).

Le jeu se termine soit dès que 2048 est dans la grille (et là le joueur gagne) ou quand après un mouvement la grille est pleine (et là le joueur perd). Quand le joueur perd, son

score est le plus grand entier de la grille.

Vous trouverez un exemple de partie en annexe.

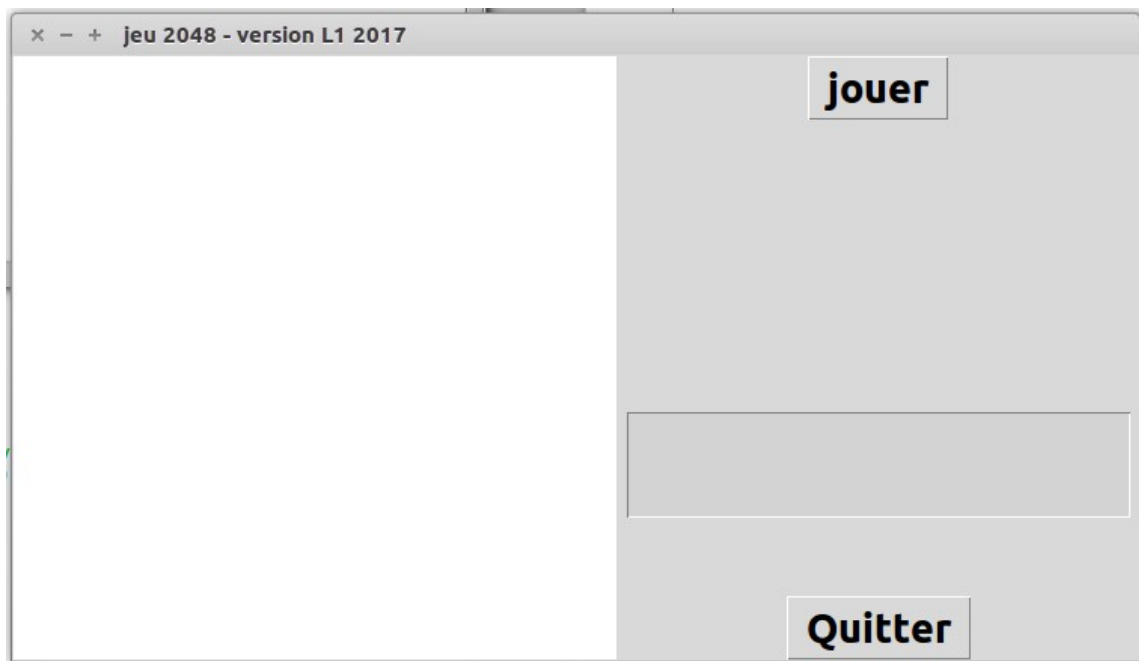
Si lors d'un mouvement l'utilisateur saisit autre chose que H, B, D ou G alors il ne se passe rien.

Partie 1B : travail avec interface graphique

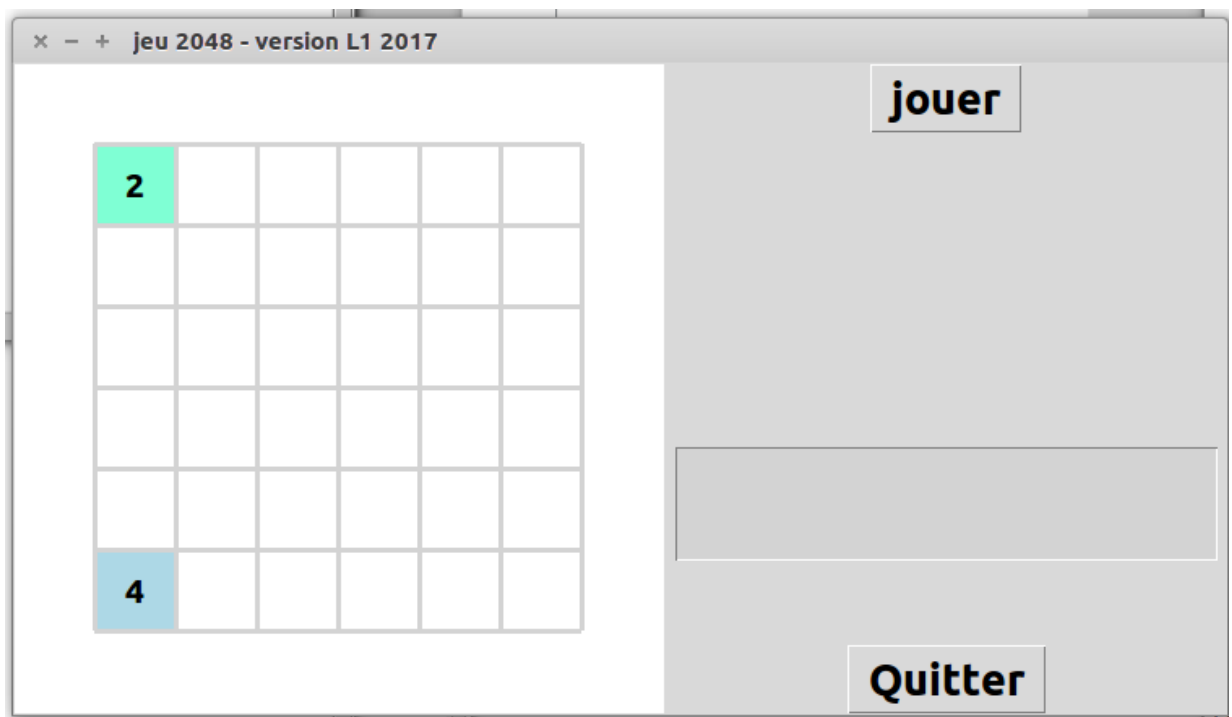
Il s'agit de faire avec une interface graphique une autre version du jeu précédent ; on utilisera Tkinter impérativement.

Bien entendu vous allez ré-utiliser un certain nombre de fonctions de la première partie, notamment tout ce qui concerne la gestion des grilles.

L'interface se présente de la manière suivante au départ :



Un click sur le bouton jouer lance une partie.



Exemple de fin de jeu :





Dans cette partie, on mettra n , taille de la grille, en variable globale et par défaut $n=6$

Question 1 Mettre en place les widgets nécessaires . On mettra en place notamment une zone où pour afficher des messages au joueur.

Question 2 Écrire la fonction **message(m)** qui efface la zone de message puis affiche dans cette zone la chaîne m .

Question 3 Écrire la fonction **dessineGrille(g)** où g est une grille comme celles de la partie 1A. On doit obtenir un dessin comme celui des figures ci-dessus.

Chaque case qui ne contient pas 0 est coloriée en fonction de l'entier contenu (toutes les cases de même valeur doivent avoir la même couleur). Vous avez le choix des couleurs. Vous mettrez les couleurs choisies dans un dictionnaire où les clés seront les puissances de 2 et les valeurs les noms des couleurs associées. Ce dictionnaire doit être de longueur 11. Par exemple dico1={2 : 'blue', 4 : 'lime', etc...}

On écrira une fonction couleur(nb) qui renvoie la couleur correspondant à l'entier nb. Selon votre choix de couleurs on pourra écrire en blanc, en noir ou changer selon la valeur de la couleur.

Question 4 Écrire la fonction qui va déclencher les mouvements : le joueur déclenchera un mouvement en utilisant les touches de direction. Écrire la fonction **clavier(event)** qui réalise un mouvement correspondant à la touche utilisée. Il faut donc déclencher le mouvement puis afficher la nouvelle grille obtenue et éventuellement afficher un message de fin de jeu. On utilisera bien entendu les fonctions de la première partie

(droite, gauche, ajouteAlea ...)

Question 5 Mettre en place le jeu. Pour l'instant chaque click sur le bouton jouer permet de recommencer une autre partie.

Partie Améliorations possibles

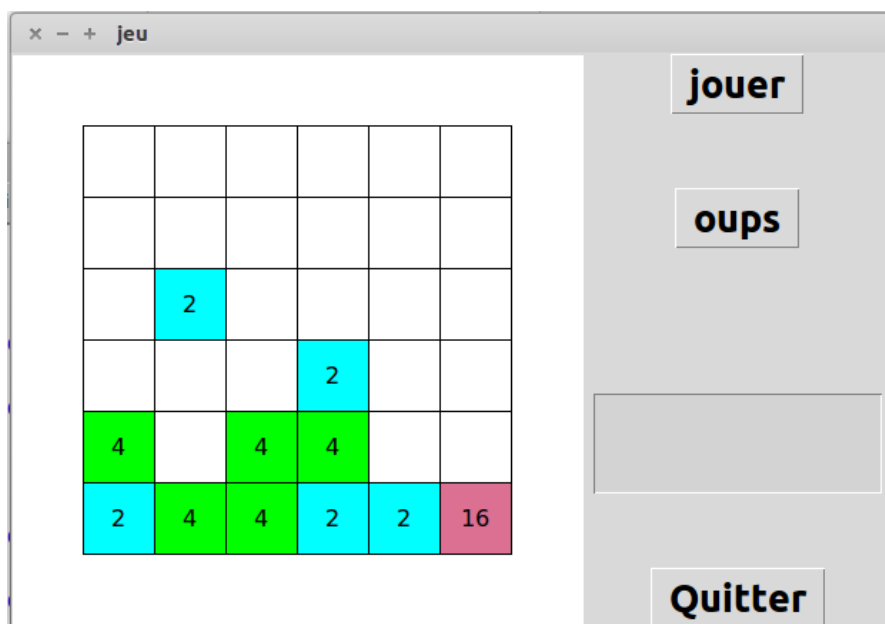
→ **mémoire** : souvent dès qu'on a joué on regrette.... Mais « oups » c'est trop tard.

Mettre en place un bouton « oups » qui permet de revenir en arrière.

Dans un premier temps le click sur oups permettra de revenir à la grille précédente et uniquement elle.

Dans un second temps chaque click reviendra en arrière d'une étape jusqu'à remonter au départ du jeu.

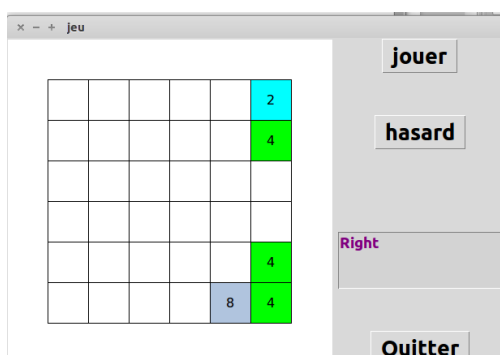
On utilisera la fonction deepcopy du module copy.



→ **mode demo** : on va mettre en place un mode démo qui se mettra en place que si l'utilisateur a déjà cliqué sur jouer (sans test).

- première étape :

mettre en place un bouton hasard qui réalise un coup choisi aléatoirement (et affiche ce coup). On jouera aléatoirement une direction sans test (il est donc possible que le mouvement ne déclenche rien).



- deuxième étape

mettre en place un bouton demo qui permet d'enchaîner les coups aléatoirement jusqu'à ce que la partie soit finie.

À chaque étape un coup sera choisi aléatoirement et affiché puis le canvas sera mis à jour avec update et une pause sera assurée par `can.after(400)` puis la fonction pourra être rappelée...

- troisième étape

rajouter un bouton stop. Grâce à un booléen il permettra d'arrêter la demo quand on clique dessus (la démo se rappelle tant que le booléen est à True)

On doit pouvoir reprendre la démo en cliquant à nouveau sur demo.

