Projet MINI_POO – 2021_S2

QUALITES ATTENDUES DU PROJET MINI_POO

Fiabilité : Il doit donner les résultats corrects attendus : Initialiser les variables même si Python le fait dans certains cas, ce qui évite certaines surprises et des débogages laborieux…

Robustesse : Il doit gérer les erreurs évidentes de manipulation des utilisateurs (test de type, de valeur…).

Convivialité : Il doit être agréable à utiliser (souris, icônes, menus…), et facile à prendre en main (le scenario d'utilisation semble « logique » à l'utilisateur).

Efficacité : Il doit donner des réponses rapides et claires.

Compacité : Il doit occuper une place modérée en mémoire et sur le disque !

Lisibilité : Il doit être structuré en classes, fonctions et procédures toutes commentées et présentées clairement. Vos variables doivent être explicites. N'hésitez pas à développer vos propres classes pour plus de lisibilité !

Portabilité : Il doit être aisément transférable sur une machine d'un autre type (attention aux widgets ou aux librairies spécifiques, aux chemins d'accès codés spécifiquement pour votre machine…).

Flexibilité : Une partie de votre travail doit pouvoir être aisément utilisable par d'autres applications. Pour cela nous vous préconisons de dissocier fortement vos interfaces de vos traitements et de généraliser vos fonctions et procédures.

Et Enfin : **Privilégiez le bon fonctionnement du programme à son aspect !** Le résultat n'est pas le seul point évalué : votre autonomie, votre travail personnel, l'originalité et la bonne exécution de vos solutions, le respect des consignes… sont d'autant d'éléments qui participent à votre évaluation !

Attention ! Le présent document peut être amené à subir des modifications, vérifiez fréquemment sur l'espace SAVOIR que vous travaillez bien sur la dernière version de ce document !

Edition	Nature de l'évolution	Evolution	Date
V1.0	Création	Première rédaction complète	18/04/2022
V1.1	Correction	Correction pattern de la bombe	22/04/2022
V1.3	Précisions	Multiples précisions et corrections	28/04/2022
V1.4	Précisions	Multiples corrections et précisions + mode emploi générateur de cartes	09/05/2022

REALISATION D'UN JEU DE MATCH-3

★★☆ Algorithmes de traitement
★★☆ Interface
★☆☆ Outils non vus en cours

Présentation du problème :

On souhaite développer un jeu de type match-3 (https://fr.wikipedia.org/wiki/Jeu_de_tile-matching) dont Candy Crush™ en est le plus célèbre représentant. Dans ce type de jeu, le joueur doit associer trois éléments pour les faire correspondre (couleurs, formes ou autre) et disparaître. La partie est gagnée quand les objectifs de la carte sont complétés (détruire ou faire disparaitre k éléments d'un type, utiliser n bonus, …) en un nombre de coup limité propre à la carte (dont un exemple est illustré par la Figure 1).

Etant donné que le nombre de coup est limité, il peut être intéressant de jouer sur les mécanismes destructeurs offerts par plusieurs bonus et si possible aboutir à des effets boule de neige dévastateurs (et très satisfaisants). Ainsi à chaque tour de jeu, le joueur doit donc choisir entre : intervertir deux éléments de la grille ou activer un bonus (voire les deux si l'un de deux éléments intervertis est un bonus).

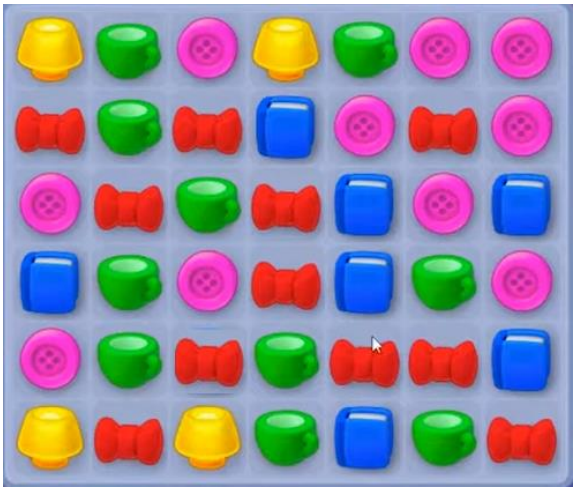
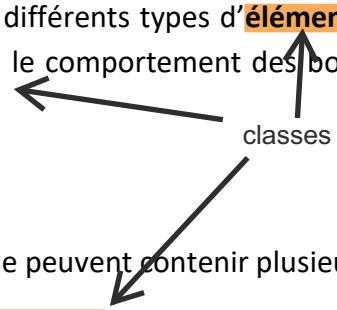


Figure 1: Exemple d'une grille d'un match-3, avec différents éléments à combiner (match)

Les sections suivantes détaillent chaque partie essentielle à ce type de jeu et que vous devez gérer : la **grille** et les différents types d'**éléments** qui la constituent, le matching (simple ou générant des **bonus**), le comportement des bonus et le déplacement des éléments dans la grille de jeu.



Les éléments :

Les cellules d'une grille peuvent contenir plusieurs types d'éléments :

- Les éléments **classiques** tout d'abords qui n'ont comme caractéristique distinctive que leur couleur (suivant le niveau, qui peut être affectée parmi 2 et 5 couleurs différentes).

- Les bonus, qui lorsqu'ils sont activés (par le joueur ou par l'effet d'autre bonus) détruisent une partie du contenu de la grille en fonction de leur type :
    - La **roquette** : détruit tous les éléments disposés sur la ligne (ou la colonne en fonction du type de roquette) où elle a été activée ou combinée.
    - La **bombe** : détruit tous les éléments disposés dans un rayon de 3 cellules (illustré sur la Figure 2) de là où elle est activée ou combinée.

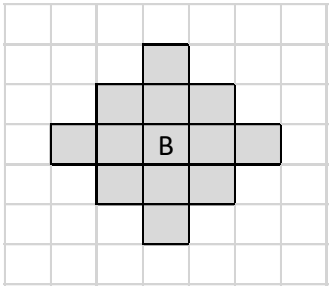


Figure 2 : Rayon d'action de la bombe

    - L'**avion** : détruit les 4 cellules adjacente lors de son décollage et va détruite un objectif (du niveau) aléatoire ou une cellule aléatoire si tous les objectifs ont déjà été atteints ou qu'aucun n'est disponible.
    - Le **déflagrateur** : détruit tous les éléments classiques de la grille du type de celui avec lequel il a été combiné. Si le bonus a été activé (juste en cliquant dessus) le déflagrateur choisit aléatoirement comme cible le type d'un élément classique disposé sur une cellule adjacente. Dans le cas où aucun élément classique n'est présent dans les cellules adjacentes, ce type est choisi aléatoirement parmi ceux disponibles dans le niveau.

  On considère que lorsqu'un bonus déclenche un ou plusieurs autres, leurs effets sont gérés de façon « simultanée ». La grille n'est donc pas mise à jour entre leur différente activation.
- Les **étoiles** qui ne peuvent pas être considérées dans les combinaisons, sont indestructibles mais déplaçables. L'objectif dans certains niveaux les exploitant est de les évacuer de la grille par une cellule puit.

A noter que tous les éléments contenus dans une cellule normale d'une grille peuvent se déplacer en fonction des cellules libérées par l'activation de bonus ou la combinaison d'un match-3.

Combinaisons et génération de bonus :

Le mécanisme le plus important de ce type de jeu est la combinaison des éléments de la grille. C'est d'ailleurs ce que recherche le joueur lorsqu'il réalise une permutation entre deux éléments qui la constitue.

Plusieurs cas de figure peuvent alors intervenir ce que votre programme doit être capable de reconnaitre (*pattern matching*) et de gérer :

- Cas simple (et classique) ou 3 éléments classiques identiques sont alignés (horizontalement ou verticalement) : c'est le match-3. Dans ce cas les 3 éléments disparaissent des cellules concernées qui se retrouvent vides. Elles doivent être à nouveau remplies en utilisant le principe détaillé les sections suivantes.

- Cas de génération de bonus en fonction de la combinaison réalisée. Lorsque des configurations d'éléments classiques plus complexes sont réalisées, elles aboutissent à la génération du bonus associé (comme illustré par la Figure 3) :
    - Lorsque 4 éléments classiques identiques sont alignés ils disparaissent et génèrent une roquette :
        - Si les 4 éléments sont alignés horizontalement, la deuxième cellule de cette combinaison laissera la place à une roquette verticale.
        - A l'inverse, si les 4 éléments sont alignés verticalement, la seconde cellule (en partant du haut de cette alignement) laissera la place à une roquette horizontale. Les 3 autres cellules sont vidées des éléments fraichement combinés.
    - Quand 4 éléments classiques identiques sont combinés sous la forme d'un carré, ils génèrent un avion dans le coin haut gauche de cette forme. Les 3 autres cellules sont vidées de leur contenu.
    - Lorsque 5 éléments classiques sont combinés sous la forme de deux matchs-3 horizontaux et verticaux ayant un élément commun (les 4 configurations considérées sont illustrées au centre de la Figure 3), cela génère une bombe.
    - Lorsque 5 éléments classiques identiques sont alignés (horizontalement ou verticalement) ils disparaissent et génèrent un déflagrateur au centre de cette configuration. Comme pour les autres combinaisons, les 4 autres cellules sont vidées de leur contenu.

si gelé ?
---> bonus gelé qui récupère le niveau de gel de la case gelée: ne peut être dégelé que par les effets d'un autre bonus

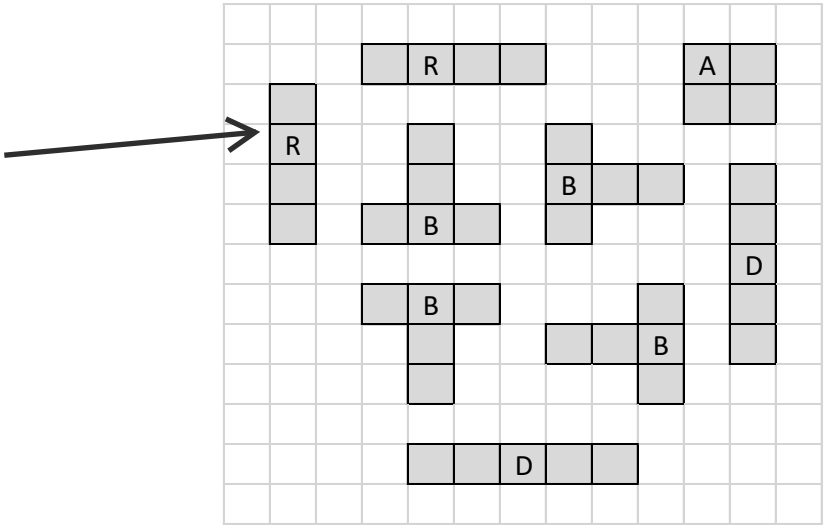


Figure 3 : Génération de bonus - différentes combinaisons possibles considérées

- Cas de combinaison de deux bonus. Afin d'obtenir des effets dévastateurs, un joueur chanceux ou très doué peut arriver à combiner des bonus entre eux (bombe + roquette par exemple). Par soucis de simplification, il n'est pas demandé dans ce projet de gérer ce cas de figure, on considère alors que les deux bonus ainsi combinés sont activés.

A noter que le programme, lorsqu'il détecte plusieurs possibilités de combinaisons, doit prendre celle qui est la plus avantageuse pour le joueur (génération de déflagrateur > bombe > roquette > avion > combinaison classique). Il est également important de préciser

ordre de priorité de test des patterns

que les couleurs ainsi combinées ne typent pas le bonus obtenu : il n'existe donc pas de bonus rouge ou vert, quelques soit les éléments combinés il donneront un bonus identique.

Grille et cellules :

Chaque niveau est composé d'une **grille**, que l'on considère rectangulaire, composée de **cellules** de plusieurs types (limité à 3 dans ce sujet) en fonction des comportements qu'elles peuvent porter :

- Le premier type de cellule est le type « **vide** » : il ne peut contenir aucun élément et le joueur ne peut pas interagir avec ce dernier. Il est exclusivement utilisé pour ne pas avoir que des formes de grille rectangulaires mais d'autres un peu plus complexes et/ou esthétiques.
- Le type de cellule le plus classique, qui est nommé « **normal** » dans cet énoncé, peut contenir (et faire circuler) des éléments de tous types et peut entrer en interaction avec le joueur (pour effectuer une permutation).
- Le dernier type est celui intitulé « **gelé** ». Ce dernier peut contenir un élément (de tout type aussi bien classique qu'un bonus) qui est gelé (avec une force paramétrée allant de 1 à 3). Le joueur ne peut déplacer l'élément ainsi gelé et contenu dans cette cellule, mais ce contenu est affecté par des combinaisons et les effets de bonus. Ainsi une fois touché par un effet d'un bonus ou une combinaison classique (3 éléments classiques de même type combinés), la force est réduite de 1 jusqu'à atteindre à la valeur de 0. A cet instant la cellule de type gelé devient une cellule de type « normal » jusqu'à la fin de la partie.

A noter que toute cellule, à part celle de type « vide », ne peut rester sans contenu. Lorsque qu'un bonus a été activé ou lorsqu'une combinaison a été détectée et activée, les éléments sont déplacés au sein de la grille passant ainsi de cellules en cellules. Ce mécanisme est détaillé dans la section suivante.

peut-être instaurer un test : si la cellule est vidée, alors il faut la reremplir. Si elle n'a pas qqn qui peut la remplir par flux principal, alors chercher dans les flux avancés

Déplacements des éléments sur la grille :

destruction cases et génération boni

Comme précisé en conclusion de la section précédente, une fois les effets de combinaisons ou de bonus gérés, les éléments se déplacent dans la grille, celle-ci ne pouvant tolérer de vide. A noter qu'il est possible que ce déplacement aboutisse à de nouvelles combinaisons, voire destructions, nécessitant à nouveau de remplir les cellules vides (effet boule de neige habituellement recherché par le joueur). Le joueur ne peut reprendre la main et dépenser une action qu'uniquement la grille à nouveau complétée et ainsi stabilisée.

Le déplacement des éléments (appelé flux) dans la grille est géré par les cellules elles-mêmes qui possèdent comme caractéristique le sens d'écoulement des éléments qui les traversent (haut, bas, droite ou gauche). L'écoulement des éléments n'est donc pas nécessairement du haut vers le bas ! Afin de déterminer le début et la fin de ce flux d'éléments, certaines cellules sont considérées comme des sources et d'autres des puits. Une source est ainsi un générateur aléatoire d'élément classique, si celle-ci s'avère vide. A l'inverse un puits est un élément terminal : tout élément y arrivant y restera bloqué. Ce type de cellule est utile pour

hypothèse : une case max pointe vers une autre case

méthode des grilles : remplissage : si vide et non source, chercher dans les voisins qui pointe vers elle
si vide et source : générer aléatoirement un élément classique
si vide et puits : rien ne se passe

la gestion des éléments de type étoile qui disparaissent de la carte une fois arrivée sur un puits.

Ainsi, après une action du joueur, il est nécessaire que chaque cellule vérifie si la suivante dans le flux est vide. Si c'est le cas elle doit transmettre son contenu à celle-ci, et ainsi de suite jusqu'à atteindre une cellule source, qui génère un nouvel élément qui peut à son tour se déplacer. A noter qu'il peut être intéressant de développer un algorithme permettant aux cellules fraichement vidées de « demander » un nouvel élément aux cellules qui les précèdent dans le flux d'élément. C'est un choix de conception que vous devez prendre et un traitement clef dans ce type de jeu.
Remarque : Dans cet énoncé on considère que la carte est bien conçue et qu'il est toujours possible d'atteindre une cellule vide. Il n'est donc pas demandé à votre programme de le vérifier.

Déplacements des éléments sur la grille - Avancé :

Il est également à préciser que, pour limiter les situations dans lesquelles des cellules ne peuvent être remplies, ne sont considérées comme cellules suivantes non pas uniquement la cellule adjacente directe mais également les deux en diagonales (si elles existent ou ne sont pas des cellules de type « vide ») comme le montre le schéma de la Figure 4.
Dans cet exemple on peut remarquer que la cellule fait glisser ses éléments vers la droite. Ainsi si sa voisine de droite est vide, elle transfert son élément à cette dernière dans un premier temps. Lorsque l'ensemble des flux principaux ont été exécutés, on s'intéresse dans un second temps aux cellules 2 et 3. Si ces dernières ne peuvent être remplies de façon directe, elles pourront, dans cet ordre de priorité se voir remplies.

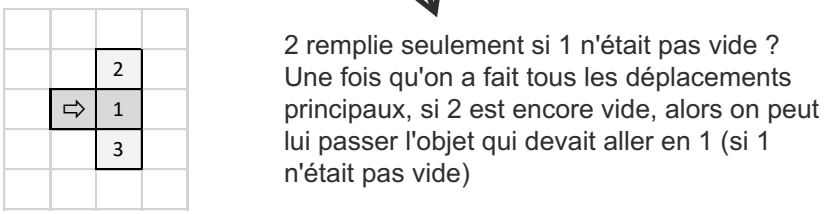


Figure 4 : Priorisation du remplissage des cellules suivantes dans un flux d'éléments

Pour certaines grilles plus complexes, on souhaite également définir des portails de téléportation d'éléments. Ainsi deux cellules sont liées de façon orientées sans pour autant être adjacentes. Le comportement de déplacement des éléments reste identique dans ce cas.

Niveau et objectifs :

Une **grille**, que l'on peut assimiler à un niveau de jeu, est associée à un ou plusieurs objectifs et caractéristiques clefs :

- Le nombre de coups que le joueur a le droit d'utiliser pour finir le niveau, au-delà si les objectifs ne sont pas atteints, le niveau est perdu.
- La liste des types d'éléments classiques disponibles sous forme d'une liste de chaînes de caractères.
- Un ou des objectifs du niveau. Le niveau est validé par le joueur lorsque ces objectifs sont tous atteints. Un objectif peut être :
  - Un nombre donné d'éléments classiques d'un type donné détruits
  - Un nombre donné de bonus activés (par le joueur ou activés par les effets d'autres bonus)
  - La disparition de la totalité des étoiles disposées dans le niveau.

  Il est donc nécessaire de détecter l'avancement de ces objectifs et d'en informer le joueur en temps réel.

compteur d'éléments détruite pour chaque couleur

compteur boni activés

compteur étoiles dans les puits et nb total d'étoiles

partie

On n'effectue les permutations qu'après avoir vérifié qu'elles fonctionnaient ?

Déroulement d'un ~~tour de jeu~~ :

ou bien on "dépermutte" à la fin, si on voit que rien ne se passe

Le déroulement d'une partie suit ces différentes étapes :

- Chargement de la grille (configuration, de ses caractéristiques et objectifs)
- Puis jusqu'à la réalisation de ces objectifs ou qu'il ne reste plus de coup à jouer :
  - Le joueur peut réaliser une action (une inversion du contenu de deux cellules ou l'activation d'un bonus de la grille). Si le coup proposé par le joueur n'aboutit à rien (aucune match 3 ni activation d'un bonus), il n'est pas décompté et la modification proposée n'est pas réalisée.
  - Le système doit reconnaitre les différentes combinaisons possibles et appliquer les effets (disparition et/ou effet des bonus)
  - Le système met à jour la grille en effectuant le déplacement des éléments contenus dans la grille (en gérant d'éventuelles nouvelles combinaisons et générations) jusqu'à ce que celle-ci soit complétée et stable (plus d'activation de match-3 ou autres bonus)
- Une fois la partie terminée en donner l'issue (victoire ou défaite + statistique du nombre d'éléments de chaque type détruits)

Il n'est pas demandé que votre solution puisse détecter une situation de blocage pour la corriger (c'est-à-dire qu'aucun coup n'est jouable (pas de match-3 possible dans cette configuration ni aucun bonus à jouer).

Interface utilisateur

L'interface est à réaliser lorsque la structure objet est fonctionnelle (vous avez ainsi testé toutes les classes et les relations possiblement admissibles). Cette interface doit permettre de :

- Charger et afficher un niveau à jouer, préalablement enregistré au format XML,

- Permettre au joueur d'activer bonus et/ou de réaliser les permutations entre le contenu de deux cellules « classiques »,
- Afficher le résultat des effets du tour de jeu du joueur. Attention, il n'est pas demandé ici de réaliser l'animation de cette résolution. Néanmoins mettre dans la console les actions réalisées peut vous aider à vérifier que votre programme est bien fonctionnel.
- Afficher le résultat de la partie.

N'hésitez pas à exploiter l'ensemble des widgets disponibles dans la librairie `tkinter` et ses sous librairies.

Travail à réaliser :

Cette section propose une décomposition du travail à réaliser. Certaines étapes sont indépendantes des autres. Les mots écrits **en gras** dans cet énoncé doivent apparaitre dans votre programme sous la forme d'une classe. Vous pouvez en ajouter d'autres si vous en avez besoin ou si vous les jugez nécessaires. Il est attendu un code respectant les bonnes pratiques détaillées en cours (nommage et portées des variables, accesseurs, héritage…).

Les principales fonctionnalités attendues du programme sont :

OK
- Concevoir l'ensemble des classes et leurs attributs permettant de gérer la grille, ses cellules et leur contenu dans un premier temps.

OK
- Concevoir et tester le bon fonctionnement de l'identification d'une combinaison (allant du simple Match-3 aux différents bonus définis précédemment dans cet énoncé). =Pattern Matching Match 3 et bonus

en cours
- Concevoir et tester le bon fonctionnement du déplacement des éléments au sein d'une grille (simple dans un premier temps, puis de complexité plus avancée).

en cours
- Concevoir et tester le décompte des objectifs de la partie.

en cours
- Réaliser une interface utilisateur permettant de réaliser les fonctionnalités précisées dans la section « interface utilisateur ».

Toutes les initiatives peuvent être intéressantes. Si vous souhaitez en prendre n'hésitez pas à en parler à l'enseignant pour avis et validation avant de débuter de longs développements inutiles ou trop complexes pour les objectifs pédagogiques de ce module !

Annexe – Utilisateur de l'outil de création de carte :

Afin de générer des cas spécifiques ou des cartes complexes sans avoir à faire le fichier XML à la main, un outil de création de carte est proposé. La Figure 5 illustre ce prototype (moche soyons honnête) et les zones de saisies principales et les informations manipulées.

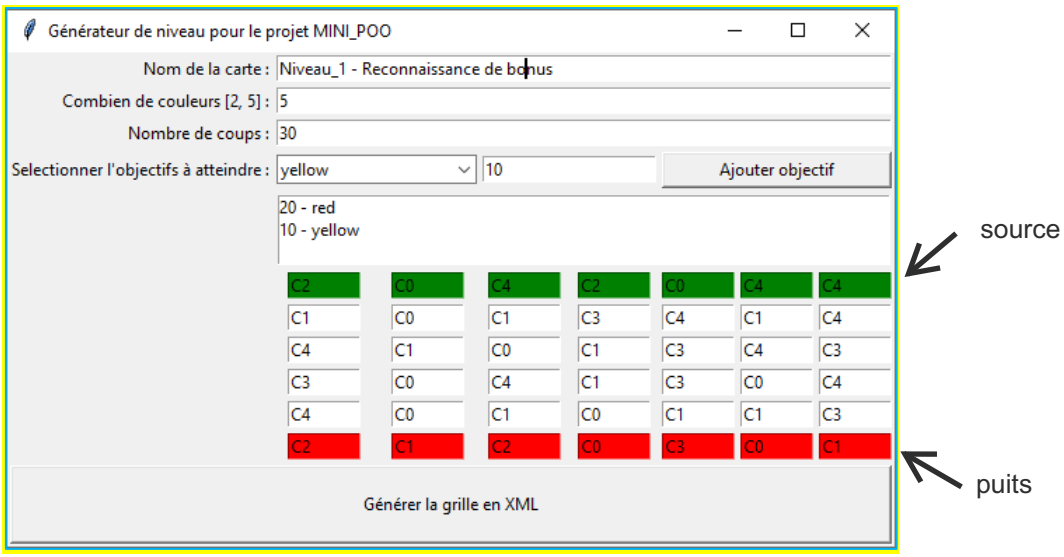


Figure 5 : Impression d'écran de l'application de définition de carte

L'interface est instanciée à l'aide de la seule ligne de code suivante dans laquelle le premier argument est le nombre de ligne de la carte et le second, les colonnes. On suppose ainsi que toute carte est inscrite dans un rectangle de cette dimension : `mon_interface = Interface(6,7)`. Une fois le programme lancé, l'interface illustrée par la Figure 5 apparait.

Plusieurs informations doivent être saisies :

- Tout d'abord le nom de la carte, voire les objectifs de test de celle-ci
- Le nombre de couleurs qui seront utilisées dans la carte. Cette information est utile pour les cellules de type source, qui doivent générer des éléments classiques aléatoirement lorsqu'elles sont vides. A noter que les couleurs sont stockées dans une liste dans cet ordre : `["green", "red", "yellow", "blue", "magenta"]`
- Le nombre de coups accordés au joueur
- Les objectifs de la partie pour l'emporter. Le menu déroulant permet de sélectionner la cible et la zone de texte adjacente, la quantité. Par l'appui sur le bouton le plus à droite, l'objectif est ajouté. Il n'y a pas de limite au nombre d'objectifs (d'ailleurs peu de contrôle de vérification sont opérés ici)
- La saisie la plus importante et la plus longue également concerne la matrice de zones de texte qui permet d'en définir le contenu. Deux modes de définition des informations :
    - Par un double clic sur le bouton droit, le type de la cellule (puits, source) est modifié : la cellule change de couleur en conséquence (vert = source, rouge = puits)
    - Par une saisie dans la zone de texte, le contenu de la cellule est défini. Cette définition fonctionne par paramètres non ordonnés et séparés d'un tiret, comme par exemple : « `C1 – G3 – OD` ». La convention à respecter est la suivante :
        - Paramètre « C », suivi d'un chiffre ou d'une lettre défini l'élément contenu dans la cellule. Si c'est un entier n, le contenu est un élément classique dont la couleur est celle d'index n dans la liste `["green", "red", "yellow", "blue", "magenta"]`. Si c'est un caractère, le contenu suivra cette correspondance : `"B" = "Bombe", "A" ="Avion", "RV" =`

ex: elt rouge, gelé de force 3, avec orientation du flux vers la droite

même si la map ne présente que 3 couleurs, p.ex: ["red","yellow","magenta"] ?

"Roquette verticale", "RH" = "Roquette Horizontale", "D" = "Deflagrateur" et "E":"Etoile"

A noter que si aucune valeur n'est associée au paramètre C de la zone de texte, le programme choisira une couleur parmi celles autorisées pour ajouter un élément classique à cette cellule.

- Paramètre « G », suivi d'un entier spécifie que la cellule est de type gelé dont la force est l'entier saisi.
- Paramètre « O », suivi d'un caractère spécifie le sens d'écoulement des éléments au travers cette cellule. Si cette cellule est de type puits, elle n'aura pas d'orientation spécifiée (c'est la fin du flux). La convention est la suivante : "H" = "Haut", "B" = "Bas", "D" = "Droit" et "G" = "Gauche"
- Paramètre « TP », suivi de coordonnées sous la forme (ligne, colonne), permet de définir une téléportation entre cette cellule et celle de coordonnées spécifiées. Cela est utile pour la gestion des flux avancés. Le programme ne vérifie pas la validité de la cible de la téléportation.
  - Toute zone de texte laissée vide est considérée comme une cellule de type vide.

les cellules qui téléportent n'ont pas d'orientation ?

Ainsi la saisie visible dans Figure 5 modélise la carte illustrée Figure 1. Une fois l'appui sur le bouton « Générer la grille en XML », une fenêtre s'ouvre pour choisir le nom du fichier qui est codé en (pretty) XML et son emplacement.

Conseils et méthodologie de travail :

Dans le contexte de conception et d'implémentation en groupe, le travail en binôme peut être complexe à mettre en place : il est donc nécessaire d'adopter une méthode de travail efficiente. Pour cela :

- **Travaillez de façon continue**, évitez le stress d'une réalisation en dernière minute (bannissez la nuit de la POO) ! Vous avez plusieurs **séances encadrées**, **mettez-les à profit** en arrivant avec des questions et/ou des propositions ! Venez à la première séance en ayant au moins lu cet énoncé, voire avec des questions en cas d'incompréhension !
- **Définissez ensemble** le **diagramme de classes** que vous souhaitez implémenter à partir de la lecture attentive de l'énoncé. Cherchez à ce que ce diagramme soit le plus détaillé possible (les attributs doivent être typés, les méthodes également (arguments et retours)). Pour travailler sur le même modèle, vous pouvez utiliser le site https://www.lucidchart.com ou Microsoft Visio pour concevoir de façon collaborative ce modèle structurant commun.
- **Répartissez-vous l'implémentation de la structure des classes**. Par structure, on entend : l'ajout de la classe et des éventuelles relations d'héritage, le constructeur, la définition et le typage des attributs, la gestion des pointeurs, les propriétés (et donc par extension : les accesseurs et mutateurs) et l'ajout des méthodes définies dans le diagramme de classe. Pour ces méthodes, il n'est pas encore question de rédiger tout le code, mais uniquement la déclaration, la documentation (avec les symboles ''' en début de fonction) et les vérifications des types des arguments passés.
- Maintenant que le squelette est ainsi réalisé, **répartissez-vous** ensuite la réalisation de **chaque méthode**. Définissez ensemble comment faire pour les tester de façon

```
class Element():
    def __init__(self, monargt1, monargt2):
        self.argt1 = list(monargt1)
        self.argt2 = int(monargt2)
```

indépendante. N'hésitez pas à documenter votre code (non pas toutes les lignes, mais plutôt le principe employé, les contraintes et/ou spécificités).

- Cherchez à faire de la **vérification croisée** : cela vous permettra de comprendre le travail de votre binôme mais également de permettre de lever des erreurs qu'il ne voit plus ou qu'il n'aurait pas vues. N'hésitez pas à mettre en place des fonctions générant des situations à tester. Ce type de démarche peut être valorisée lors de l'évaluation.
- Ne cherchez pas à réaliser toutes les fonctionnalités demandées d'un coup, travaillez par étapes (l'ordre proposé dans la section « Travail à réaliser » peut vous aider en cela), que vous validez au fur et à mesure avant de passer à la fonctionnalité suivante.