Travail pratique # 3
La Guerre des Dés

Louis Fortier-Dubois

Date de remise : au plus tard le 31 mars 2022 à 23h59
Pondération de la note finale : 12 %

1 Objectifs

Ce travail a comme principaux objectifs de vous familiariser davantage avec la programmation orientée objet, via une modélisation déjà faite pour vous, que vous devez d'abord comprendre puis compléter. Ce travail vous permettra de vérifier votre compréhension de la matière des modules 1 à 6, inclusivement. La modélisation que nous vous fournissons tient pour acquis que vous comprenez très bien le principe de la décomposition fonctionnelle et la réutilisation de fonctions. Certaines méthodes pourront par exemple être programmées en trois ou quatre lignes de codes, lorsqu'on réutilise les méthodes programmées préalablement.

2 Organisation du travail en équipe

Comme mentionné en classe, ce travail est conçu pour être fait en équipe, car il s'agit d'un objectif de votre formation académique. Vous pourrez ainsi vous partager la charge de travail. Pour ceux qui n'ont pas encore trouvé de coéquipier ou qui veulent former une nouvelle équipe, nous vous invitons à utiliser le forum du cours dans la section prévue à cet effet. Vous avez **jusqu'au 10 mars 23h59 pour créer vos équipes**, après quoi les étudiants seuls seront jumelés au hasard par l'enseignant. Chaque coéquipier doit contribuer à parts égales au développement de ce travail. Laisser son coéquipier faire tout le travail (peu importe les raisons) est inacceptable : vous passerez à côté des objectifs de ce cours. De la même manière, il ne faut pas non plus trop en faire : il faut apprendre à travailler en équipe !

Nous fournissons sur le site web du cours un tutoriel pour utiliser des technologies permettant de partager votre code avec votre coéquipier. Chaque membre d'une équipe possède sur son ordinateur une copie locale du code, mais le code est synchronisé par un service dans le « *cloud* » et nous pouvons également voir l'historique (qui a fait quoi et quand). De plus, PyCharm intègre ces outils de manière très conviviale. Nous vous conseillons d'écouter toutes les capsules vidéo complémentaires concernant la technologie Git avant de débuter le travail.

3 Le prix Pierre Ardouin

Depuis l'automne 2013, le département d'informatique et de génie logiciel a mis en place un concours récompensant l'équipe qui aura produit le meilleur TP/projet dans le cadre d'un cours. Ces travaux de session ont l'envergure d'un mini-projet qui est admissible par rapport aux normes fixées par le département. À la suite des évaluations des travaux, l'enseignant du cours détermine l'équipe gagnante ; chaque membre de l'équipe gagnante reçoit alors une bourse de 50$ ainsi qu'une attestation remise par le département.

De plus, le département d'informatique et de génie logiciel a mis en place une bourse Élite, appelée bourse « Pierre Ardouin », qui vise à récompenser le meilleur projet de session parmi tous les cours de première année, pour toute l'année 2021-2022. Deux principaux critères guident le choix des évaluateurs dans l'identification du lauréat : l'excellence du travail (par rapport à ce qui est demandé dans l'énoncé) et l'aspect créativité/innovation. Il est actuellement prévu une bourse de 200$ pour récompenser chaque membre de l'équipe « élite » gagnante (pour un maximum de 1000$ pour toute l'équipe). Aussi, le département veille à publier l'information sur un site Web dédié : http ://www.ift.ulaval.ca/vie-etudiante/prix-pierre-ardouin.

À la deuxième moitié du mois de mai de chaque année universitaire, le département organise une cérémonie pour honorer les finalistes et le lauréat du prix « Pierre Ardouin » des sessions d'automne et d'hiver, et leur remettre une attestation.

Le travail pratique 4 sera une continuité du travail pratique 3, et l'ensemble de ces deux travaux sera évalué pour le prix Pierre Ardouin. Notez que l'aspect créativité/innovation sera particulièrement évalué sur les éléments du TP4, qui contiendra une interface graphique.

4 La Guerre des Dés

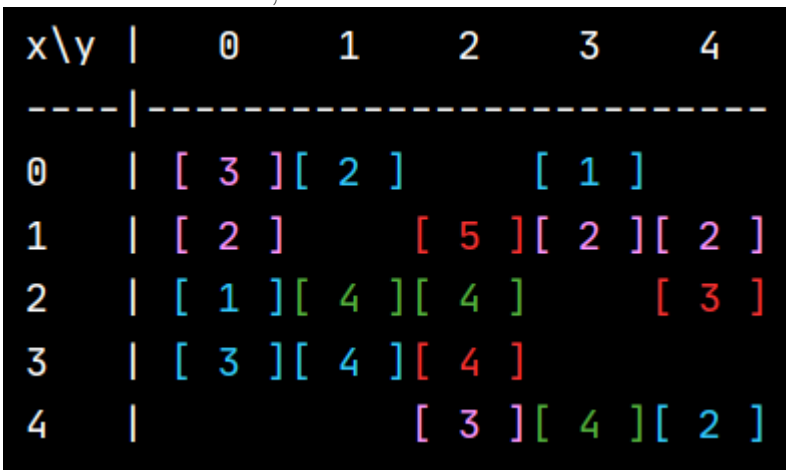
La Guerre des Dés est un jeu de stratégie où plusieurs joueurs se disputent la possession de territoires, un peu dans le style du célèbre jeu de société Risk. Les joueurs possèdent au départ un nombre égal de cases, qui sont répartis aléatoirement sur une carte. Chacun son tour, les joueurs doivent tenter d'envahir les cases appartenant à leur adversaire en se battant... à coup de dés ! La partie se termine quand un joueur a éliminé tous ses adversaires ou, en d'autres mots, lorsqu'il possède toutes les cases.

Avant toute chose, vous êtes encouragés à vous rendre sur le site web :
`https://www.gamedesign.jp/games/dicewars/`

Sur ce site ayant servi d'inspiration directe pour le TP, vous contrôlez l'équipe mauve. Les autres joueurs sont contrôlés par une intelligence artificielle. Tentez de jouer une partie à 7 joueurs pour comprendre les mécaniques générales du jeu. Vous comprendrez rapidement que plus une case a de dés, plus elle est puissante : elle est à la fois meilleure pour attaquer ses voisins et pour s'en défendre. Vous verrez aussi que vous avez droit à autant d'attaques que vous le souhaitez durant votre tour, mais que les cases n'ayant qu'un dé ne peuvent attaquer. Enfin, vous remarquerez qu'à la fin de votre tour, vous êtes réapprovisionnés en

dés selon le plus grand nombre de cases qui vous appartiennent et sont adjacentes les unes aux autres.

Les règles détaillées du jeu sont décrites plus bas dans ce document. Lors de ce travail pratique, nous programmerons **la totalité des règles du jeu**. La seule simplification que nous ferons par rapport au jeu disponible sur le site web est que *toutes les cases seront carrées*. Pour ce TP, la carte sera affichée en console ainsi :



Les cases, dont le nombre de dés présents est affiché en leur centre, seront sélectionnées en inscrivant explicitement leurs coordonnées et la carte sera réimprimée en console après chaque modification.

C'est au TP4 que le jeu sera porté vers un interface graphique où l'on pourra sélectionner les cases en cliquant dessus. Ne vous inquiétez pas, nous fournirons la solution du TP3 entre la remise de celui-ci et le début du TP4. Vous pourrez alors choisir entre prendre la solution ou votre propre code comme base pour le TP4.

5 Travail à faire

Une modélisation orientée-objet complète a été définie pour vous, et plusieurs méthodes sont déjà programmées. Cependant, le code de **35 méthodes** est manquant et il est de votre devoir de les implémenter. Pour chacune, une docstring expliquant clairement ce qui est attendu de vous est fournie à même les fichiers Python. Plusieurs de ces méthodes nécessitent d'appeler d'autres méthodes afin de bien réutiliser le code. Ainsi, la majorité de vos fonctions ne devraient contenir que quelques lignes de code.

Il vous sera **permis** de définir d'autres méthodes que celles qui vous sont fournies. Mais **attention**, vous devez vous assurer que toutes les méthodes qu'on vous demande d'implémenter le soient. Dans l'exemple suivant, la version de gauche est correcte mais pas celle de droite :

```python
def nouvelle_methode(self, x):
    # traitement

def methode_demandee(self, x):
    return self.nouvelle_methode(x)
```

```python
def nouvelle_methode(self, x):
    # traitement

def methode_demandee(self, x):
    pass # ne fait rien puisque
         # le code utile est dans
         # l'autre méthode
```

En d'autres termes, vous pouvez diviser les méthodes qui vous sont demandées en sous-méthodes, mais vous ne devez pas les remplacer. Vous ne devez pas créer de nouveaux fichiers ou de nouvelles classes.

6 Modélisation

La totalité du programme est divisé en **10 fichiers Python**, que voici en ordre alphabétique :

- **afficheur.py** : Contient des fonctions utilitaires pour contrôler l'affichage. Plutôt que de faire directement des *print* et *input* dans votre code, vous devriez appeler les méthodes *afficher* et *demander* de ce fichier. Cela vous permettra d'une part d'être directement compatible avec le TP4, et gèrera l'affichage colorisé. **Vous n'avez rien à programmer dans ce fichier.**
- **carte.py** : Contient la classe Carte, qui gère l'ensemble des cases du jeu. Les cases sont contenues dans un attribut de cet objet, dans un dictionnaire dont les clés sont les coordonnées d'une case et la valeur l'objet de type Case correspondant. Les méthodes les plus complexes sont déjà implémentées, mais **6 méthodes** restent à implémenter.
- **carte_autogeneree.py** : Contient la classe CarteAutogeneree, une classe enfant de la classe Carte où les cases sont créées au hasard selon une hauteur, une largeur et un nombre de trous. Par exemple, la carte présentée à la section précédente a été auto-générée avec hauteur 5, largeur 5, et 8 trous. La création d'une telle carte peut échouer si on demande trop de trous. En effet, il faut toujours que toutes les cases soient connectées, c'est-à-dire qu'il est possible de se rendre à toute case à partir de n'importe laquelle. **Vous n'avez rien à programmer dans ce fichier.**
- **case.py** : Contient la classe Case, c'est-à-dire un carré de la carte. Cette classe gère son appartenance à un joueur spécifique, sa liste de dés et son affichage (selon si elle est passive, en attaque ou en défense). **9 méthodes** restent à implémenter ici.
- **de.py** : Contient la classe De, qui permet de rouler un dé. Par défaut, un dé a 6 faces, mais il serait possible de créer des dés d'autres formes (ce n'est pas demandé !). **1 seule méthode** est à programmer ici.
- **guerre_des_des.py** : Contient la classe GuerreDesDes (Guerre des dés). Cette classe contrôle l'ensemble d'une partie à haut niveau. Sa méthode la plus importante est **deroulement_global**, qui contrôle la séquence d'action ayant lieu dans une partie. Cette méthode est déjà implémentée, et une explication du déroulement vous sera

donnée dans la prochaine section. Un total de **7 méthodes** demeurent à programmer dans ce fichier, concernant les attaques, les fins de tour et la fin de partie.

- **joueur.py** : Contient la classe Joueur, qui gère les stratégie de sélection des cases, ainsi que la gestion des dés correspondant à un seul joueur. Vous devez implémenter **8 méthodes** dans ce fichier.
- **joueur_humain_console.py** : Contient la classe JoueurHumainConsole, qui hérite de la classe Joueur. Il s'agit d'un joueur humain dans le cadre du TP3 (au TP4, les joueurs humains seront gérés différemment). Vous devez programmer les **2 méthodes** présentes dans ce fichier, c'est-à-dire la sélection d'un attaquant et la sélection d'un défenseur (la case que le joueur humain attaquera). Ces fonctions doivent demander des coordonnées valides à l'utilisateur.
- **joueur_ordinateur.py** : Contient la classe JoueurOrdinateur, qui hérite de la classe Joueur. Ici, la sélection de l'attaquant et du défenseur est faite automatiquement selon divers critères, qui sont décrits en détails dans le fichier. Si les stratégies globales sont déjà implémentées, vous devez toutefois implémenter **2 méthodes** utiles à la réalisation de la stratégie de l'intelligence artificielle.
- **principal.py** : Le point d'entrée du programme. C'est ce fichier que vous devez exécuter afin d'exécuter l'application. On vous demandera d'entrer des dimensions et le nombre de joueurs humains et ordinateurs, puis la partie sera démarrée. **Vous n'avez rien à programmer dans ce fichier.**

Pour résumer, on démarre une partie avec **principal.py**. Cela vous demandera d'entrer des dimensions (*hauteur, largeur, nombre de trous*) afin de créer la **Carte autogénérée**, un type de **Carte** qui contiendra *hauteur* × *largeur* − *nombre de trous* **Cases** générées au hasard. Ces cases seront attribuées aux **Joueurs**. Selon le déroulement donné par la **Guerre des dés**, à chaque tour les joueurs doivent sélectionner des attaquants leur appartenant et des défenseurs adverses parmi les cases de la carte. S'il s'agit d'un **Joueur humain**, les coordonnées sont demandées en console, via le fichier **afficher.py**. S'il s'agit d'un **Joueur ordinateur**, les coordonnées sont sélectionnées automatiquement par l'intelligence artificielle. La **Guerre des dés** contrôle l'attaque en fonction des **Dés** contenus dans les cases en conflit.

7 Règles du jeu détaillées

Voici une liste exhaustive des règles du jeu. La meilleure façon de comprendre la mécanique du jeu est toutefois de jouer sur le site en ligne. Nous vous conseillons d'y jouer jusqu'à ce que vous arriviez à remporter une partie à 7 joueurs. Vous devriez alors bien comprendre toutes les subtilités du jeu.

- Début de la partie :
  - Chaque joueur commence avec un nombre égal de cases (ou presque, si le nombre de cases n'est pas divisible équitablement).
  - Par défaut, chaque case contient un dé.

  - Chaque joueur démarre avec 10 dés supplémentaires répartis sur ces cases, selon la même procédure que lors de la fin du tour (voir plus bas).

- Déroulement :
  - Chaque joueur joue à tour de rôle, suivant un ordre prédéfini.
  - Lorsque c'est son tour, un joueur peut effectuer autant d'attaques qu'il le souhaite, jusqu'à ce qu'il décide lui-même de mettre fin à son tour ou lorsque aucune de ses cases n'est en mesure d'attaquer.
  - Un joueur est éliminé lorsque aucune case ne lui appartient.
  - Un joueur remporte la partie lorsque tous les autres joueurs sont éliminés ou, de façon équivalente, lorsqu'il possède toutes les cases de la carte.
  - Lorsqu'un joueur remporte la partie, un message de victoire est affiché et le programme s'arrête.
- Conditions pour attaquer :
  - Les cases ayant un seul dé ne peuvent pas attaquer.
  - Les cases ne peuvent attaquer que des cases d'un autre joueur.
  - Conséquemment, un joueur doit avoir au moins une case ayant au moins deux dés et un voisin ennemi pour pouvoir attaquer.
  - Les cases étant carrées, un voisin de la case x, y est une case qui est soit en haut $(x-1, y)$, à droite $(x, y+1)$, en bas $(x+1, y)$ ou à gauche $(x, y-1)$. **Les cases en diagonales ne sont pas considérées voisines**.
  - Un joueur ayant déjà sélectionné un attaquant peut revenir sa décision (lorsque la sélection d'un défenseur ennemi retourne None).
- Attaque :
  - Pour attaquer, une case doit posséder **au moins 2 dés**.
  - La case ennemie se défendant n'a pas de restrictions sur son nombre de dés.
  - Lorsqu'une case est sélectionnée pour attaquer, son affichage passe de `[ n ]` à `X n X`, où n est le nombre de dés que la case possède. Lorsqu'une case est sélectionnée pour se défendre, son affichage passe de `[ n ]` à `< n >`.
  - La force de la case attaquante est déterminée en lançant tous les dés qu'elle possède et en additionnant leur valeur. La force de la case qui se défend est aussi déterminée par la somme de ses dés.
  - Si la force de l'attaquant est **strictement plus grande** que la force du défenseur, **l'attaquant remporte la bataille**.
  - Par exemple, dans la figure, supposez que la case rouge en (1,2) attaque la case verte en (2,2). On lance les 5 dés de la case rouge et on obtient un total de 20. On

lance alors les 4 dés de la case verte et on obtient un total de 17. Alors l'attaque est réussie.

  - Si la force de l'attaquant est **plus petite ou égale** à la force du défenseur, **c'est le défenseur qui remporte la bataille**. On dit donc que le défenseur a un *bonus défensif* puisque, étant donné le même nombre de dé de chaque part, c'est celui-ci qui a le plus de chances de l'emporter.

  - Par exemple, supposez qu'en lançant les dés de la case verte à l'exemple précédent on ait obtenu 22, ou même 20. Alors l'attaque serait un échec.

- Après la bataille :

  - **Si l'attaque est réussie**, la case qui se défendait passe aux mains du joueur qui a attaqué. Les dés de la case qui se défendait sont supprimés. Les dés ayant servi à l'attaque sont transférés à la case qui se défendait, sauf un qui reste dans la case qui attaquait (et qui appartient toujours au même joueur). Par exemple, dans la figure, si la case rouge en (1,2) [5 dés] attaque la case magenta en (1,3) [2 dés] et *remporte* son attaque. Alors la case (1,2) reste rouge et ne contient qu'un seul dé, et la case (1,3) *devient* rouge et contient 4 dés.

  - **Si l'attaque est un échec**, la case qui se défendait reste inchangée, et les dés de la case qui attaquait sont tous supprimés sauf un. Par exemple, si la case rouge en (1,2) [5 dés] attaque la case magenta en (1,3) [2 dés] et *échoue* son attaque (ce qui est improbable, mais pas impossible !), alors la case (1,2) reste rouge mais ne contient qu'un dé, et la case (1,3) reste magenta et contient toujours deux dés.

- Fin du tour :

  - Un joueur termine son tour lorsque la sélection d'attaquant retourne None.

  - Le joueur est alors ravitaillé en dés. Le nombre de nouveaux dés est égal à la taille du plus grand territoire contigu du joueur. En d'autres termes, il obtient un nombre de dé égal au nombre de cases faisant partie de la plus grande région connectée des cases du joueur. Par exemple, dans la figure mentionnée précédemment, le joueur bleu obtiendrait 3 dés, alors que le joueur rouge n'en obtiendrait qu'un seul.

  - Les nouveaux dés sont répartis *au hasard* parmi toutes les cases du joueur (pas seulement celles de la région connectée), sauf celles qui sont déjà à capacité maximale.

  - Une case a une capacité maximale de 8 dés. Ainsi, si une case a déjà 8 dés, les nouveaux dés doivent être distribués parmi les autres cases seulement.

  - Si toutes les cases sont à capacité maximale, les nouveaux dés sont accumulés dans les dés en surplus du joueur. Ces dés pourront être répartis à travers les cases du joueur à la fin de son prochain tour, ou conservées dans les dés en surplus s'il demeure impossible de les répartir.

Cela peut vous sembler très gros, mais rappelez-vous

- que les règles du jeu vous viendront intuitivement si vous jouez suffisamment à la version en ligne ;
- qu'une bonne partie du code est déjà programmée pour vous ;
- que les méthodes que vous devez implémenter son bien décrites dans leur documentation ;
- et que vous êtes plusieurs pour réaliser ce travail !

8 Travail en équipe

Bien que ce ne soit pas formellement obligatoire, nous vous encourageons à utiliser Git pour effectuer le travail en équipe. Cet outil est très utilisé partout où du partage de code est nécessaire et sera obligatoire à connaître plus tard dans votre cheminement (si vous êtes au baccalauréat en informatique). Cela vous permettra de garder des sauvegardes de différentes versions de votre code et de vous synchroniser avec vos coéquipiers même si vous travaillez en même temps sur le TP. Plusieurs capsules vidéo ont été spécialement préparées pour vous aider à configurer et utiliser Git (et le site web GitHub). Vous les trouverez sur le site du cours, dans **Contenu et activités** puis **Capsules vidéo complémentaires**.

9 Exemple d'exécution

Un exemple d'exécution vous est fourni sous forme de capsule vidéo. Rendez-vous dans **Contenu et activités** puis **Capsules vidéo complémentaires**, et sélectionnez la capsule nommée *Exemple d'exécution du TP3*.

10 Ce que vous devez remettre

Votre programme doit être rédigé à même les 10 fichiers Python fournis, que vous devez compresser dans une archive Zip avec le fichier **correction.txt**. Sur le site du cours, dans Contenu et activités, Capsules vidéo complémentaires, une **capsule vidéo** nommée *Démarrer et remettre un TP* a été mise à votre disposition pour vous montrer comment y arriver.

Assurez-vous que vous remettez tous les fichiers nécessaires à l'exécution de votre TP. Nous vous recommandons fortement de créer un **dossier** dans lequel vous mettrez les fichiers relatifs à votre TP, et **rien d'autre**. Nous ne pourrons pas donner de points à votre travail si vous remettez le mauvais fichier. Les **capsules** *Les fichiers sous Windows* et *Les fichiers sous Mac OS* devraient vous aider à bien vous organiser.

Aussi notez qu'un programme qui n'est pas fonctionnel (qui ne s'exécute pas ou qui plante à l'exécution) pourrait recevoir une note de 0. Il en sera de même pour tous les étudiants qui ne respecteront pas l'interface des méthodes ou des classes.

11 Modalités d'évaluation

Ce travail sera évalué sur 100 points, et la note sera ramenée sur 12. Attention : 25% des points seront accordés pour la jouabilité générale (peu importe comment vous avez programmé le tout), 70% seront accordés selon si les méthodes demandées ont le résultat attendu, **d'où l'importance de respecter la décomposition fonctionnelle fournie**. Un barème plus détaillé est disponible au niveau du fichier `correction.txt`.

Voici la grille de correction :

Élément	Pondération
Jouabilité (affichage, respect des règles du jeu, etc.)	25 points
Méthodes (35 * 2 points)	70 points
Qualité du code (noms de variables, style, commentaires, documentation)	5 points

12 Remarques additionnelles

Plagiat : Tel que décrit dans le plan de cours, le plagiat est strictement interdit. Ne partagez pas votre code source à quiconque. Une politique stricte de tolérance zéro est appliquée en tout temps et sous toute circonstance. Tous les cas détectés seront référés à la direction de la faculté. Des logiciels comparant chaque paire de TPs pourraient être utilisés pour détecter les cas de plagiat.

Retards : Tout travail remis en retard peut être envoyé par courriel à l'enseignant si le portail des cours n'accepte pas la remise. Une pénalité de 10% sera appliquée pour un travail remis le lendemain de la remise, puis une pénalité de 25% sera attribuée à un TP remis le surlendemain. Une note de 0 sera attribuée pour un plus long retard.

Remises multiples : Il vous est possible de remettre votre TP plusieurs fois sur le portail des cours. La dernière version sera considérée pour la correction.

Respect des normes de programmation : Nous vous demandons de prêter attention au respect des normes de programmation établies pour le langage Python, notamment de nommer vos variables et fonctions en utilisant la convention suivante : `ma_variable`, `Carte`, etc. Utiliser **PyCharm** s'avère être une très bonne idée ici, car celui-ci nous donne des indications sur la qualité de notre code (en marge à droite, et souligné).

Bon travail !