
SA1.02 Comparaison d'approches algorithmiques

# 1 Compétences ciblées

Cette SAÉ vise à travailler sur la **compétence 2** *Appréhender et construire des algorithmes.*

Les apprentissages critiques visés sont les suivants :
- **AC12.01** Analyser un problème avec méthode (découpage en éléments algorithmiques simples, structure de données...)
- **AC12.02** Comparer des algorithmes pour des problèmes classiques (tris simples, recherche...)

Cette SAÉ se fera en groupe de 3 ou 4. Ces groupes vous sont imposés. Il y aura plusieurs rendus à faire :
- un rendu le **mardi 13 janvier 18h** sur l'implémentation des fonctions et API demandées,
- un rendu le **vendredi 16 janvier 10h30** sur l'IA que vous aurez développée,
- un devoir sur table le **vendredi 16 janvier 11h00** portant sur la SAÉ,
- un tournoi le **vendredi 16 janvier 14h** où vos IA s'affronteront.

### 📎 IMPORTANT ! Utilisation des IA génératives

L'utilisation d'IA génératives est fortement déconseillée, si vous y avez recours vous devez l'utiliser exclusivement pour apprendre et non pour faire à votre place. Tout groupe faisant passer pour son travail un contenu (même mineur) généré par une IA sera sanctionné par un zéro.

Notez bien le devoir sur table compte pour 50% de la note de la SAÉ, vous avez donc tout intérêt à maitriser le contenu de vos rendus afin de pouvoir répondre aux questions du devoir.

# 2 Présentation du sujet

Splat IUT'O est un jeu librement inspiré et largement simplifié de Splatoon. Le jeu se joue sur un plateau où les cases sont
- soit des couloirs où les joueurs peuvent se déplacer,
- soit des murs qui ne sont pas franchissables.

Chaque joueur est associé à une couleur et son objectif est d'arriver à peindre un maximum de cases avec sa couleur. Pour cela à chaque tour, les joueurs doivent prendre deux décisions
- choisir entre peindre dans une direction ou ne pas peindre,
- se déplacer sur une case adjacente (le déplacement est obligatoire).

Les joueurs ont une réserve de peinture qui est remplie à sa capacité maximale au début du jeu. Cette réserve diminue à chaque fois que le joueur lance de la peinture. Repeindre une case non peinte ou déjà peinte par le joueur coûte une unité dans la réserve. Repeindre une case déjà peinte par un autre joueur coûte deux unités de la réserve. Les murs ne peuvent pas être repeints sauf dans le cas où on obtient un objet spécial (voir plus loin). Lorsque le joueur touche d'autres joueurs lors de son action, il leur *vole* des unités de peinture qui iront dans sa réserve. Toucher un joueur qui n'a plus de peinture ne rapporte rien. La portée du jet de peinture est limitée.

Pour remonter sa réserve de peinture un joueur doit se déplacer sur des cases qui sont peintes de sa couleur. Se déplacer sur une case peinte d'une autre couleur que la sienne fait baisser la réserve de peinture.

Au cours du jeu des objets spéciaux peuvent apparaître de manière aléatoire sur le plateau. Pour obtenir un objet, il suffit de se déplacer sur la case où se trouve l'objet. Le joueur qui obtient un objet gagne un bonus en unités de peinture. Un joueur ne peut avoir qu'un seul objet à la fois et cet objet a une durée de vie limitée une fois qu'il est pris par un joueur. Voici la liste des objets et leur pouvoir :

- La bombe  : cet objet permet de peindre dans toutes les directions en commençant par celle choisie par le joueur et en tournant dans le sens horaire.
- Le pistolet  : cet objet permet de peindre les murs. L'avantage de peindre les murs est qu'il est plus difficile à un autre joueur de repeindre ces murs puisqu'il doit trouver lui-même un pistolet.
- Le bouclier  : cet objet permet au joueur d'être protégé si un autre joueur lui lance de la peinture en évitant d'être volé.
- Le bidon  : Cet objet permet aux joueurs qui ont une réserve négative de la remettre à 0. Le bidon a un effet immédiat et si le joueur possédait déjà un objet, il ne le perd pas.

Lors de la création de la partie la durée (en nombre de tours) de celle-ci est fixée ainsi que tous les paramètres de bonus et malus. Le tableau ci-dessous donne les valeurs qui seront utilisées lors du tournoi. Les bonus/malus sont appliqués à la réserve de peinture du joueur.

| | | | | | |
|---|---|---|---|---|---|
| Durée partie | 200 | Durée vie objet | 5 | Portée peinture | 5 |
| Bonus recharge | 3 | Bonus joueur touché | 5 | Bonus objet | 5 |
| Pénalité | 2 | Capacité réservoir | 20 | | |

Au début de la partie, les joueurs sont placés aléatoirement sur le plateau. La partie se déroule tour par tour où chaque joueur effectue une action de peinture et un déplacement. Afin de ne pas privilégier un joueur par rapport aux autres, l'ordre des joueurs est pris aléatoirement à chaque tour. A chaque tour de jeu le joueur accumule des points qui correspondent au nombre de cases peintes de sa couleur après avoir exécuté ses actions. A la fin de la partie, les joueurs sont classés en fonction de leur nombre de points.

# 3 Principes du serveur de jeu

L'objectif de la SAÉ n'est pas d'implementer entièrement le jeu mais plutôt de programmer une intelligence artificielle permettant de défier les autres équipes lors du tournoi. Une partie de Splat'IUTO est gérée par un serveur auquel les joueurs vont se connecter. Le serveur de jeu effectue les actions suivantes :

— crée un jeu à partir d'une carte qui définit les murs et les couloirs du plateau,
— attend l'enregistrement de 4 joueurs,
— lance la partie.

Au cours de la partie, le serveur

— envoie à chaque joueur, l'état du plateau et des joueurs,
— attend les ordres envoyés par chaque joueur,
— applique les ordres de chaque joueur en tirant au sort l'ordre de passage.

Chaque joueur est un programme client qui va se connecter au serveur. Au début du jeu, le joueur va s'enregistrer en indiquant son nom. Au cours du jeu le joueur

— attend l'état du jeu envoyé par la serveur,
— exécute une fonction spéciale appelée `mon_IA` qui va déterminer les actions à exécuter pour ce tour,
— envoie ses actions au serveur.

Votre travail va donc consister à implémenter la fonction `mon_IA` . Pour cela vous aurez à comprendre les API qui vous sont fournies et créer des fonctions qui permettent d'exploiter l'état du plateau pour prendre les bonnes décisions.

# 4 Structure de l'archive et lancement du jeu

Voici la structure du répertoire `SAE_splat_iuto` contenu dans l'archive

```
SAE_splat_iuto
+-- sujet_sae_splat_iuto.pdf
+-- lancer_joueurs.sh
+-- pyproject.toml
+-- affichage
+-- bot_ia
|   +-- __init__.py
|   +-- EQUIPE
|   +-- case.py
|   +-- client.py
|   +-- client_joueur.py
|   +-- const.py
|   +-- joueur.py
|   +-- plateau.py
+-- cartes
|   +- carte.txt
+-- images
+-- serveur
+-- tests
    +-- test_joueur.py
    +-- test_plateau.py
```

Dans le répertoire principal, vous trouverez le sujet, ainsi qu'un script bash pour lancer l'application et le fichier `pyproject.toml` qui est liée à l'architecture de l'application organisée en packages. Ce répertoire contient les 6 sous-répertoires suivants :

— `cartes` qui contient des cartes de plateau pour le jeu sous forme de fichier texte,
— `images` qui contient les images utilisées pour l'affichage graphique du jeu,
— `tests` qui contient les deux fichiers de tests pour votre premier rendu,
— `serveur` qui contient le package implémentant le serveur du jeu,
— `affichage` qui contient le package pour l'affichage graphique du jeu,
— et enfin `bot_ia` qui contient le package sur lesquel vous aurez à travailler.

L'application est organisée sous forme de packages (serveur, affichage et bot_ia), c'est la raison de la présence du fichier `pyproject.toml` et des fichiers `__init__.py` dans les répertoires `serveur` , `affichage` et `bot_ia` . Cela expliquera aussi la manière dont les programmes sont lancés.

Les fichiers sur lesquels vous allez travailler se trouvent dans le répertoire `bot_ia` .

— `EQUIPE` : le plus simple, il s'agit du fichier où vous indiquerez le nom d'équipe que vous vous êtes choisi ainsi que les noms des membres de l'équipe (dans le format indiqué dans le fichier).
— `joueur.py` : c'est le module de gestion des joueurs et que vous devrez implémenter.
— `plateau.py` : c'est le module contenant l'API de gestion du plateau de jeu. Dans ce fichier vous aurez un certain nombre de fonctions à implimenter.
— `client_joueur.py` : qui contient le programme principal du joueur. Dans ce fichier vous aurez à implémenter la fonction `mon_IA` qui définira le comportement votre bot.

Des détails plus précis sur le travail à faire vous sont donnés dans la suite.

## 4.1 Lancement d'une partie

Pour lancer une partie, il faut ouvrir au moins deux fenêtres dans le répertoire du projet. Comme l'application est conçue sous forme de packages le lancement se fait avec l'option `-m` de `python3` et le nom du programme est `nom_package.nom_progr` Dans la première fenêtre, vous lancez le serveur avec la commande `python3 -m serveur.serveur` . Dans la seconde, vous allez lancer le client d'affichage puis quatre joueurs en arrière plan

```
1    python3 -m affichage.affichage&
2    python3 -m bot_ia.client_joueur --equipe joueur1&
3    python3 -m bot_ia.client_joueur --equipe joueur2&
4    python3 -m bot_ia.client_joueur --equipe joueur3&
5    python3 -m bot_ia.client_joueur --equipe joueur4&
```

Vous pouvez aussi utiliser le script bash `lancer_joueurs.sh` . Pour commencer la partie, il suffit d'appuyer sur la touche `Entrée` dans la fenêtre du serveur.

Ces instructions lancent 4 fois le même client qui joue de manière complètement aléatoire. Il ne vous reste plus qu'à modifier la fonction `mon_IA` .

## 4.2 Quelques détails sur le code

Comme indiqué précédemment, l'objectif de la SAÉ est d'aller au-delà du joueur aléatoire et de construire une IA qui prenne des décisions en fonction de l'état du plateau à un instant donné. Pour cela, il va falloir comprendre les API servant à implémenter le plateau de jeu et les joueurs afin de pouvoir les utiliser pour consulter l'état du jeu. La structure principale est le *plateau* qui est implémenté comme une matrice de *cases*. Les *joueurs* sont représentés par un dictionnaire qui contient leurs propriétés. Vous aurez à implémenter cette dernière API.

### case.py

Cette API est un simple dictionnaire permettant de représenter une case du plateau avec son contenu. Vous aurez sans doute à utiliser des fonctions observatrices de cette API.

### plateau.py

Cette API est en fait une matrice 2D similaire à celles vues en TP. Les éléments de cette matrice sont des cases de l'API `case.py` . La construction d'un plateau se fait avec la fonction `plateau_from_str` qui prend une chaine de caractères contenant les informations de l'état du plateau. Vous n'avez pas besoin de comprendre les détails de cette fonction. Cette API contient les mêmes fonctions d'observation que celles des matrices. Quelques fonctions utilisées par la serveur vous sont aussi données. Elles permettent par exemple de déplacer un joueur, poser ou enlever un objet. La fonction permettant de peindre dans une direction est aussi donnée. Dans cette API vous aurez un certain nombre de fonctions à écrire dont les spécifications sont données dans le dans `plateau.py` . Ces fonctions pourront vous être utiles pour créer votre IA.

### joueur.py

Cette API que vous devrez implémenter, représente un joueur sous la forme d'un dictionnaire. Les données stockées dans ce dictionnaire sont celles fournies en paramètres de la fonction `Joueur` Les fonctions de cette API sont principalement des *setters* et des *getters* permettant de mettre à jour ou de consulter ces propriétés du joueur. Les fonctions `ajouter_objet` et `maj_temps` gèrent l'ajout et la fin des objets gagnés par le joueur. La fonction `modifie_reserve` gère le contenu du reservoir dont la capacité est limitée. La fonction `joueur_from_str` permet de créer un joueur à partir d'une chaine de caractères comme celle-ci `"A;15;4;2;3;0;1;le peintre"` où les propriétés sont séparées par des `;`. L'ordre des propriétés est donné dans la spécification de la fonction. Notez que cette fonction est appelée dans le programme principal de `client_joueur.py`

### client_joueur.py

Ce fichier contient le programme principal du joueur. Ce script s'occupe de la connexion au serveur et des interactions avec celui-ci. Notamment, à chaque tour de jeu, le programme appelle la fonction `mon_IA` avec comme paramètres les informations envoyées par le serveur sur l'état des joueurs, du plateau et les caractéristiques de la partie. Dans ce fichier, vous n'aurez que la fonction `mon_IA` à modifier.

## 5 Rendus de la SAÉ

#### 1er rendu : Implémentation des API 13 janvier 18h

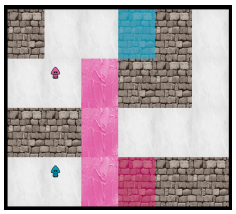
Pour la première partie (structure de données et API), il vous est demandé d'implémenter :

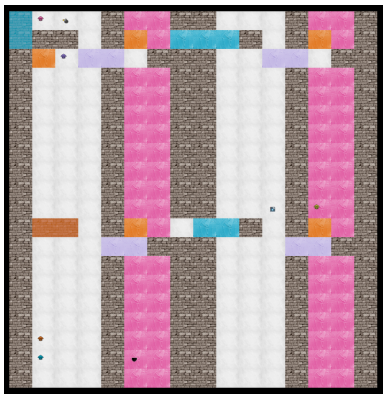
- votre fichier EQUIPE complété avec le nom de votre équipe et ses membres
- l'API `joueur.py` qui gère les informations liées à un joueur donné.
- les fonctions de `plateau.py` : `surfaces_peintes` , `directions_possibles` , `nb_joueurs_direction` et `distances_objets_joueurs`

La spécification des fonctions est indiquées dans les fichiers Python `joueur.py` et `plateau.py`.

Notez bien que l'implémentation de ces fonctions ne sont pas strictement indispensables à la création de l'IA mais peuvent malgré tout vous aider à comprendre les structures de données et à trouver des idées de fonctions à mettre en oeuvre pour établir votre stratégie.

Des jeux de tests vous sont fournis afin que vous puissiez vérifier la correction de vos fonctions sont correctes. Pour lancer ces tests, il vous suffit de taper la commande `pytest -v` dans le répertoire principal du projet. Les deux plateaux utilisés pour les tests sont les suivants :





### 1er rendu : sur CELENE

Votre rendu doit être une archive zip de votre répertoire `bot_ia`

**2ème rendu : Définition de l'IA et son implémentation 16 janvier 10h30**

Afin d'implémenter votre IA vous allez devoir établir une statégie qui va reposer sur une observation de l'état du plateau. Il va donc falloir définir un certain nombre de fonctions qui vont donner des informations vous permettant de prendre des décisions. Vous pouvez imaginer toutes sortes de stratégies dépendant de l'état du plateau, de votre réserve de peinture, de la durée restante etc. Il vous est demandé que les décisions de votre IA concernant la peinture et le déplacement se basent sur une évaluation les avantages et les inconvénients de chaque choix possible et de faire un classement de ces choix afin de prendre la décision considérée comme la meilleure.

Au dela de l'efficacité de votre IA qui sera évaluée lors du tournoi, votre rendu sera évalué au vu des apprentissages critiques de concernés par cette SAÉ. Notamment :

- le découpage en fonctions de votre code,
- la bonne utilisation des structures de données et la création d'API si nécessaire,
- l'évaluation de la complexité de vos algorithmes et la justification des choix d'implémentation,
- les commentaires et les tests unitaires qui auront été effectués.

### 2ème rendu sur CELENE

Votre rendu doit être une archive zip de votre répertoire `bot_ia` contenant

- le fichier `EQUIPE` duement complété,
- tous les fichiers `.py` nécessaires à votre joueur,
- les fichiers de tests si vous en avez développés,
- votre rapport au format pdf contenant
  - la stratégie de votre IA,
  - les principaux algorithmes que vous aurez implémentés. Pour chaque algorithme, il faudra donner une approximation de sa complexité et justifier les choix de structure de données et algorithmiques,
  - un état de ce que vous avez réussi à faire et de ce qui ne marche pas.