INFO3 – Année 2025/2026
Méthodes numériques pour l'Apprentissage Machine
Énoncé du projet

Mise en garde : Pour tous les TPs et projets que vous réalisez dans votre scolarité, il est attendu un travail original, et non une copie. Il est donc interdit de copier même partiellement des TP/Projets d'autres étudiants, et ceci également d'une année sur l'autre. De même, il est interdit de communiquer des TP/Projets à d'autres étudiants que des membres de son équipe. Il est interdit de réutiliser tout ou partie d'un ancien TP ou projet ; ou d'inclure dans son projet des portions de fichiers en provenance de sites web ou de logiciels (même libres) sans autorisation explicite de vos enseignants. Si vous le faites alors citez vos sources dans votre documentation. Notez que les supports de cours, ainsi que les énoncés de TPs, de TDs et de projets fournis par les enseignants sont la propriété intellectuelle des enseignants. Il est donc interdit de les communiquer ou de les téléverser sur des applications ou sites web (*e.g.* ChatGPT, ou autres) sans consentement explicite de l'enseignant.

Objectifs du projet

L'objectif de ce projet est d'implémenter un environnement d'apprentissage par renforcement (RL) simulant le comportement d'un drone autonome de patrouille. Le drone évolue dans un environnement discret (une grille 2D), dispose d'une autonomie limitée, doit éviter des obstacles, revenir se recharger lorsque nécessaire, et chercher à maximiser la couverture de zones de patrouille sans provoquer de crash. Le projet met l'accent sur :

- la compréhension du rôle d'un environnement dans l'apprentissage par renforcement,
- la modélisation correcte des états, des actions et des transitions,
- l'implémentation d'un environnement respectant une spécification donnée,
- l'expérimentation avec un algorithme de RL vu en cours (ce point sera abordé ultérieurement, lorsque les différents algorithmes auront été présentés).

Le drone est utilisé pour la surveillance d'un site (industriel, naturel ou sensible). Il décolle depuis une base fixe, patrouille une zone discrétisée en cellules, détecte des zones dangereuses (ou obstacles) et consomme de l'énergie lors de ses déplacements. Le drone peut retourner à la base pour se recharger et subit un crash s'il entre dans une zone dangereuse. Le drone est supposé autonome : à chaque étape, un agent décide quelle action exécuter. Votre rôle est d'implémenter **l'environnement de simulation** sur lequel cet agent pourra apprendre.

1 Cadrillage et notion de zones

L'environnement dans lequel évolue le drone est modélisé comme une grille 2D finie. Chaque cellule de la grille correspond à une position possible du drone. Les coordonnées de ces cellules sont entières et bornées : le drone se déplace donc dans un espace discret. Certaines cellules de la grille peuvent être considérées comme dangereuses (présence d'obstacles, zones interdites, reliefs difficiles, etc.). Si le drone entre dans une telle cellule, un risque de crash est détecté.

1.1 Découpage en zones de patrouille

Afin de définir un objectif de patrouille réaliste et exploitable, la grille est découpée en un nombre fini de **zones de patrouille**. Chaque cellule de la grille appartient à exactement une zone. Les zones constituent une abstraction du terrain : elles peuvent regrouper plusieurs cellules adjacentes et représenter, par exemple, des secteurs géographiques, des zones fonctionnelles du site surveillé ou des régions d'intérêt particulier. Ce découpage permet de raisonner à un niveau plus grossier que celui des cellules individuelles, tout en conservant une représentation discrète de l'environnement. Il constitue la base de l'objectif de patrouille, qui sera défini ultérieurement. Certaines cellules de la grille sont considérées comme dangereuses. Elles correspondent à des obstacles, des zones interdites ou des régions à risque pour le drone. Voici une définition formelle des constantes liées à la grille, aux zones de patrouille et aux cellules dangereuses.

CONSTANTS
$maxX$, $maxY$, $Danger$, $zoneCount$, $zoneOf$

PROPERTIES
$$
\begin{aligned}
& maxX \in \mathbf{NAT1} \\
& \land\ maxY \in \mathbf{NAT1} \\
& \land\ Danger \subseteq (0\,..\,maxX) \times (0\,..\,maxY) \\
& \land\ zoneCount \in \mathbf{NAT1} \\
& \land\ zoneOf : (0\,..\,maxX) \times (0\,..\,maxY) \to (1\,..\,zoneCount) \\
& \land\ zoneCount = 4 \\
& \land\ zoneOf = \{\ xx,yy,zz \mid xx \in 0\,..\,maxX \land yy \in 0\,..\,maxY \land zz \in 1\,..\,zoneCount \\
& \qquad\qquad \land\ (\\
& \qquad\qquad\quad (xx \le maxX\ /\ 2 \land yy \le maxY\ /\ 2 \land zz = 1) \\
& \qquad\qquad\ \lor (xx \le maxX\ /\ 2 \land yy > \ maxY\ /\ 2 \land zz = 2) \\
& \qquad\qquad\ \lor (xx > \ maxX\ /\ 2 \land yy \le maxY\ /\ 2 \land zz = 3) \\
& \qquad\qquad\ \lor (xx > \ maxX\ /\ 2 \land yy > \ maxY\ /\ 2 \land zz = 4) \\
& \qquad\qquad\ ) \\
& \qquad\ \}
\end{aligned}
$$

Dans cette spécification, la constante `zoneCount` fixe le nombre total de zones de patrouille. La fonction `zoneOf` associe à chaque cellule (x, y) de la grille un identifiant de zone compris entre 1 et `zoneCount`. Cette fonction est totale et déterministe : toute cellule appartient à une unique zone.

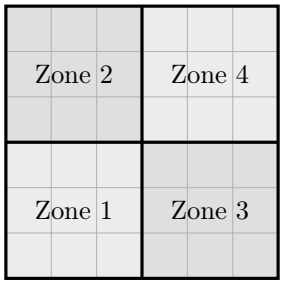

Figure 1: Exemple de grille découpée en zones de patrouille.

Exemple. Considérons une grille 6×6, donc `maxX = 5` et `maxY = 5`. Le nombre de zones est fixé à `zoneCount = 4`. La division entière donne alors `maxX/2 = 2` et `maxY/2 = 2`. La fonction `zoneOf`

associe à chaque cellule (x, y) une zone selon les règles suivantes :

$$\texttt{zoneOf}(x, y) = \begin{cases} 1 & \text{si } x \leq 2 \text{ et } y \leq 2 \\ 2 & \text{si } x \leq 2 \text{ et } y > 2 \\ 3 & \text{si } x > 2 \text{ et } y \leq 2 \\ 4 & \text{si } x > 2 \text{ et } y > 2 \end{cases}$$

Par exemple, la cellule $(1, 4)$ appartient à la zone 2, la cellule $(4, 1)$ à la zone 3, et la cellule $(5, 5)$ à la zone 4.

Dans cette spécification formelle, l'ensemble `Danger` représente l'ensemble des cellules dangereuses de la grille. Il est uniquement contraint par le fait qu'il s'agit d'un sous-ensemble des cellules valides de la grille :

$$Danger \subseteq \{0, 1, \ldots, maxX\} \times \{0, 1, \ldots, maxY\}.$$

Aucune hypothèse supplémentaire n'est faite à ce stade sur la forme ou la taille de cet ensemble.

1.2 État de l'environnement

L'environnement de simulation est décrit par un ensemble de **variables d'état** qui caractérisent à chaque instant la situation du drone. Ces variables représentent à la fois la position du drone, son état opérationnel, son niveau d'énergie, ainsi que des informations liées à la patrouille et à la sécurité.

Variables de position. Les variables `posX` et `posY` représentent la position courante du drone dans la grille. Les variables `baseX` et `baseY` désignent la position de la base fixe depuis laquelle le drone décolle et où il peut se recharger. Toutes ces variables prennent des valeurs entières correspondant à des coordonnées valides de la grille.

Énergie et perception du danger. La variable `battery` modélise le niveau de batterie du drone. Elle est bornée entre 0 et 100 et représente une ressource limitée, consommée au cours de la patrouille. La variable `obstacle_distance` représente une information abstraite liée à la proximité d'un obstacle ou d'une zone dangereuse. À ce stade, seule sa nature discrète et non négative est imposée ; son interprétation précise sera définie lors de l'étude de la dynamique de l'environnement.

État opérationnel du drone. Les variables `docked`, `airborne` et `crashed` décrivent l'état opérationnel du drone. Elles sont booléennes (valeurs 0 ou 1) et indiquent respectivement si le drone est arrimé à la base, en vol, ou dans un état de crash. Certaines combinaisons d'états sont interdites. En particulier, un drone crashé ou arrimé ne peut pas être en vol. De plus, lorsqu'il est arrimé, le drone se trouve nécessairement à la position de la base.

Couverture des zones. La variable `visZones` est un ensemble d'identifiants de zones de patrouille. Elle représente les zones qui ont déjà été visitées par le drone depuis le début de l'épisode. Par construction, cet ensemble est inclus dans l'ensemble des zones définies par `zoneCount`.

Invariant d'état. L'ensemble des contraintes précédentes est formalisé par un invariant, qui garantit la cohérence de l'état de l'environnement. Cet invariant impose notamment :

- des bornes sur les positions et le niveau de batterie ;
- la validité des indicateurs d'état du drone ;
- la cohérence entre les états `docked`, `airborne` et `crashed` ;
- la validité de l'ensemble des zones couvertes.

État initial. L'environnement est initialisé dans un état où le drone se trouve à la base, au sol, avec une batterie pleine. Aucune panne n'est présente et aucune zone n'a encore été explorée, à l'exception de la zone contenant la base, qui est considérée comme visitée dès l'initialisation.

VARIABLES
$posX$, $posY$,
$baseX$, $baseY$,
$battery$,
$obstacle\_distance$,
$docked$,
$airborne$,
$crashed$,
$visZones$

INVARIANT
$posX \in 0 \,..\, maxX$
$\wedge\ posY \in 0 \,..\, maxY$
$\wedge\ baseX \in 0 \,..\, maxX$
$\wedge\ baseY \in 0 \,..\, maxY$
$\wedge\ battery \in 0 \,..\, 100$
$\wedge\ obstacle\_distance \in \mathbf{NAT}$
$\wedge\ docked \in 0 \,..\, 1$
$\wedge\ airborne \in 0 \,..\, 1$
$\wedge\ crashed \in 0 \,..\, 1$
$\wedge\ visZones \subseteq 1 \,..\, zoneCount$
$\wedge\ (crashed = 1 \Rightarrow airborne = 0)$
$\wedge\ (docked = 1 \Rightarrow airborne = 0)$
$\wedge\ (docked = 1 \Rightarrow posX = baseX \wedge posY = baseY)$

INITIALISATION
$posX$, $posY := 0, 0 \ ||$
$baseX$, $baseY := 0, 0 \ ||$
$battery := 100 \ ||$
$docked := 0 \ ||$
$airborne := 0 \ ||$
$crashed := 0 \ ||$
$obstacle\_distance := 2 \ ||$
$visZones := \{\ zoneOf(posX,posY)\ \}$

2 Dynamique de l'environnement

L'environnement évolue en réponse aux actions exécutées par l'agent contrôlant le drone. Chaque action correspond à une opération qui modifie l'état de l'environnement selon des règles spécifiques. Ces opérations incluent le décollage, l'atterrissage, les déplacements dans les quatre directions cardinales, et le retour à la base pour se recharger. Chaque opération est soumise à des préconditions qui doivent être satisfaites pour qu'elle puisse être exécutée. Lorsqu'une opération est exécutée, elle met à jour les variables d'état de l'environnement en fonction de la logique définie ci-dessous.

2.1 Opération `takeoff_cmd`

L'opération `takeoff_cmd` modélise le décollage du drone. Elle ne peut être exécutée que si le drone est dans un état compatible avec un décollage : il ne doit pas être crashé, doit être au sol (non en vol), ne doit pas être arrimé à la base, et doit disposer d'une batterie strictement positive. Lorsque

l'opération est exécutée, l'état du drone passe en vol (`airborne := 1`). L'environnement met ensuite à jour l'information de danger `obstacle_distance` en fonction de la cellule courante : si la position actuelle $(posX, posY)$ appartient à l'ensemble `Danger`, alors `obstacle_distance` est fixé à 0 (danger immédiat) ; sinon, `obstacle_distance` est fixé à 2. Cette opération n'entraîne aucun déplacement ; elle ne modifie ni la position du drone, ni la base, ni l'ensemble des zones couvertes.

takeoff_cmd =
  PRE
    $crashed = 0 \wedge airborne = 0 \wedge docked = 0 \wedge battery > 0$
  THEN
    $airborne := 1$ ||
    IF $posX$, $posY \in Danger$ **THEN**
      $obstacle\_distance := 0$
    ELSE
      $obstacle\_distance := 2$
    END
  END;

2.2 Opération `move_step`

L'opération `move_step` modélise un déplacement élémentaire du drone dans la grille. Elle n'est exécutable que si le drone est en état de mission : il ne doit pas être crashé, doit être en vol (`airborne = 1`), ne doit pas être arrimé (`docked = 0`), et doit disposer d'une batterie strictement positive.

Le déplacement est défini de manière **non déterministe**. L'opération choisit une position suivante $(nextX, nextY)$ parmi les voisins directs de la position courante, c'est-à-dire une cellule atteignable par un déplacement d'une unité vers le haut, le bas, la droite ou la gauche, tout en restant dans les bornes de la grille. Cette modélisation capture l'idée d'un déplacement abstrait, sans imposer une direction choisie explicitement à ce niveau. Une fois la cellule suivante choisie, l'environnement met à jour l'état :

- la position du drone devient $(posX, posY) := (nextX, nextY)$;
- la batterie diminue d'une unité (`battery := battery - 1`) ;
- la zone correspondant à la nouvelle cellule est ajoutée à l'ensemble des zones couvertes :

$$\texttt{visZones} := \texttt{visZones} \cup \{\texttt{zoneOf}(nextX, nextY)\}.$$

L'information de danger est mise à jour en fonction de la cellule atteinte : si $(nextX, nextY) \in$ `Danger`, alors `obstacle_distance` est fixé à 0 ; sinon, `obstacle_distance` est fixé à 2.

move_step =
  PRE
    $crashed = 0 \wedge airborne = 1 \wedge docked = 0 \wedge battery > 0$
  THEN
    ANY $nextX$, $nextY$ **WHERE**
      $nextX \in 0 \,..\, maxX$
      $\wedge\ nextY \in 0 \,..\, maxY$
      $\wedge\ (\ (nextX = posX \wedge nextY = posY + 1 \wedge posY < maxY)$
        $\vee\ (nextX = posX \wedge nextY = posY - 1 \wedge posY > 0)$
        $\vee\ (nextX = posX + 1 \wedge nextY = posY \wedge posX < maxX)$
        $\vee\ (nextX = posX - 1 \wedge nextY = posY \wedge posX > 0)\ )$
    THEN
      $posX := nextX \ ||$
      $posY := nextY \ ||$
      $battery := battery - 1 \ ||$
      $visZones := visZones \cup \{\ zoneOf(nextX,nextY)\ \} \ ||$
      IF $nextX$, $nextY \in Danger$ **THEN**
        $obstacle\_distance := 0$
      ELSE
        $obstacle\_distance := 2$
      END
    END
  END;

2.3 Opération `avoid_maneuver`

L'opération `avoid_maneuver` modélise un déplacement d'évitement lorsque le drone cherche à réduire son exposition au danger. Elle n'est exécutable que si le drone est en vol, non arrimé, non crashé, et dispose d'une batterie strictement supérieure à 1. Cette contrainte reflète le fait qu'une manœuvre d'évitement est plus coûteuse en énergie qu'un déplacement standard.

Le comportement de l'opération dépend de l'existence ou non d'une cellule voisine non dangereuse. Dans un premier temps, l'environnement teste s'il existe au moins une cellule voisine atteignable qui n'appartient pas à l'ensemble `Danger`. Cette vérification est réalisée à l'aide d'un test de cardinalité sur l'ensemble des voisins sûrs.

Cas favorable : voisin sûr disponible. Si au moins un voisin non dangereux existe, alors l'environnement choisit une cellule voisine $(nextX, nextY)$ quelconque qui n'appartient pas à `Danger`. Le drone se déplace alors vers cette cellule. La position est mise à jour, la batterie diminue de deux unités, et la zone correspondant à la nouvelle position est ajoutée à l'ensemble des zones couvertes. Dans ce cas, l'information de danger est mise à jour de manière favorable, et `obstacle_distance` est fixé à 2.

Cas défavorable : aucun voisin sûr. Si aucun voisin non dangereux n'est disponible, le drone effectue malgré tout un déplacement vers l'une des cellules voisines atteignables, choisie de manière non déterministe. La mise à jour de la position, de la batterie et des zones couvertes est identique au cas précédent. En revanche, l'information de danger est recalculée en fonction de la cellule atteinte : si celle-ci appartient à `Danger`, alors `obstacle_distance` est fixé à 0, sinon à 2.

Cette opération illustre le compromis entre sécurité et consommation d'énergie. Elle permet au drone de privilégier des déplacements sûrs lorsqu'ils existent, au prix d'un coût énergétique plus élevé, tout en garantissant un comportement défini même dans des situations où aucun déplacement sûr n'est possible.

avoid_maneuver =
  PRE
    $crashed = 0 \wedge airborne = 1 \wedge docked = 0 \wedge battery > 1$
  THEN
    SELECT
      card({ $nextX$, $nextY$ |
          $nextX \in 0 \,..\, maxX \wedge nextY \in 0 \,..\, maxY$
          $\wedge$ (($nextX = posX \wedge nextY = posY + 1 \wedge posY < maxY$)
            $\vee$ ($nextX = posX \wedge nextY = posY$ - $1 \wedge posY > 0$)
            $\vee$ ($nextX = posX + 1 \wedge nextY = posY \wedge posX < maxX$)
            $\vee$ ($nextX = posX$ - $1 \wedge nextY = posY \wedge posX > 0$))
          $\wedge$ ($nextX$, $nextY$) $\notin$ $Danger$
        }) > 0
    THEN
      ANY $nextX$, $nextY$ **WHERE**
        $nextX \in 0 \,..\, maxX$
        $\wedge$ $nextY \in 0 \,..\, maxY$
        $\wedge$ (($nextX = posX \wedge nextY = posY + 1 \wedge posY < maxY$)
          $\vee$ ($nextX = posX \wedge nextY = posY$ - $1 \wedge posY > 0$)
          $\vee$ ($nextX = posX + 1 \wedge nextY = posY \wedge posX < maxX$)
          $\vee$ ($nextX = posX$ - $1 \wedge nextY = posY \wedge posX > 0$))
        $\wedge$ ($nextX$, $nextY$) $\notin$ $Danger$
      THEN
        $posX := nextX$ ||
        $posY := nextY$ ||
        $battery := battery$ - 2 ||
        $visZones := visZones \cup$ { $zoneOf(nextX,nextY)$ } ||
        $obstacle\_distance := 2$
      END
    ELSE
      ANY $nextX$, $nextY$ **WHERE**
        $nextX \in 0 \,..\, maxX$
        $\wedge$ $nextY \in 0 \,..\, maxY$
        $\wedge$ (($nextX = posX \wedge nextY = posY + 1 \wedge posY < maxY$)
          $\vee$ ($nextX = posX \wedge nextY = posY$ - $1 \wedge posY > 0$)
          $\vee$ ($nextX = posX + 1 \wedge nextY = posY \wedge posX < maxX$)
          $\vee$ ($nextX = posX$ - $1 \wedge nextY = posY \wedge posX > 0$))
      THEN
        $posX := nextX$ ||
        $posY := nextY$ ||
        $battery := battery$ - 2 ||
        $visZones := visZones \cup$ { $zoneOf(nextX,nextY)$ } ||
        IF $nextX$, $nextY \in Danger$ **THEN**
          $obstacle\_distance := 0$
        ELSE
          $obstacle\_distance := 2$
        END
      END
    END
  END;

Exemple illustratif. Considérons une situation dans laquelle le drone se trouve dans une cellule ayant quatre voisines atteignables. Parmi ces cellules voisines, deux appartiennent à l'ensemble

`Danger` et deux sont considérées comme sûres. Si l'agent choisit l'action `move_step`, le déplacement est effectué vers l'une des cellules voisines sans distinction préalable entre cellules sûres et dangereuses. Le drone peut donc se retrouver dans une cellule dangereuse.

En revanche, si l'agent choisit l'action `avoid_maneuver`, l'environnement privilégie un déplacement vers une cellule voisine non dangereuse. Dans ce cas, le drone est garanti de se déplacer vers une cellule sûre, au prix d'un coût énergétique plus élevé. Considérons maintenant une situation où toutes les cellules voisines sont dangereuses. L'action `avoid_maneuver` ne bloque pas le drone : un déplacement est tout de même effectué vers l'une des cellules voisines. Dans ce cas, le risque de crash demeure possible.

2.4 Opération `return_home`

L'opération `return_home` modélise un déplacement dont l'objectif est de **rapprocher le drone de sa base**. Elle est exécutable uniquement si le drone n'est pas crashé, est en vol (`airborne = 1`), n'est pas arrimé (`docked = 0`), et dispose d'une batterie strictement positive. Le déplacement est défini de manière **non déterministe**, mais il est **contraint** : la position suivante $(nextX, nextY)$ doit être choisie de façon à réduire (ou au moins ne pas augmenter) l'écart à la base sur chacun des axes. Plus précisément :

- si $posX < baseX$, alors la coordonnée $nextX$ est imposée à $posX + 1$;
- si $posX > baseX$, alors $nextX$ est imposée à $posX - 1$;
- si $posX = baseX$, alors $nextX$ reste égale à $posX$;

et de même pour l'axe vertical :

- si $posY < baseY$, alors $nextY = posY + 1$;
- si $posY > baseY$, alors $nextY = posY - 1$;
- si $posY = baseY$, alors $nextY = posY$.

Ainsi, l'opération réalise un pas de retour vers la base en s'approchant simultanément sur l'axe horizontal et/ou vertical (en fonction de la position relative à la base). Après le choix de $(nextX, nextY)$, l'environnement met à jour l'état :

- la position devient $(posX, posY) := (nextX, nextY)$;
- la batterie diminue d'une unité (`battery := battery - 1`) ;
- la zone correspondant à la cellule atteinte est ajoutée aux zones couvertes :

$$\texttt{visZones} := \texttt{visZones} \cup \{\texttt{zoneOf}(nextX, nextY)\}.$$

Enfin, l'information de danger `obstacle_distance` est mise à jour en fonction de la cellule atteinte : si $(nextX, nextY) \in \texttt{Danger}$, alors `obstacle_distance` est fixé à 0, sinon il est fixé à 2.

return_home =
  PRE
    $crashed = 0 \wedge airborne = 1 \wedge docked = 0 \wedge battery > 0$
  THEN
    ANY $nextX$, $nextY$ **WHERE**
      $nextX \in 0 \,..\, maxX$
      $\wedge\ nextY \in 0 \,..\, maxY$
      $\wedge\ (\ (posX < baseX \wedge nextX = posX + 1)$
        $\vee\ (posX > baseX \wedge nextX = posX - 1)$
        $\vee\ (posX = baseX \wedge nextX = posX)\ )$
      $\wedge\ (\ (posY < baseY \wedge nextY = posY + 1)$
        $\vee\ (posY > baseY \wedge nextY = posY - 1)$
        $\vee\ (posY = baseY \wedge nextY = posY)\ )$
    THEN
      $posX := nextX \ ||$
      $posY := nextY \ ||$
      $battery := battery - 1 \ ||$
      $visZones := visZones \cup \{\ zoneOf(nextX,nextY)\ \} \ ||$
      IF $nextX$, $nextY \in Danger$ **THEN**
        $obstacle\_distance := 0$
      ELSE
        $obstacle\_distance := 2$
      END
    END
  END;

2.5 Opérations `dock_cmd` et `undock_cmd`

Les opérations `dock_cmd` et `undock_cmd` modélisent respectivement l'arrimage du drone à sa base et son redécollage après une phase au sol. Elles permettent de représenter explicitement les phases de recharge et de reprise de mission.

Opération `dock_cmd`. L'opération `dock_cmd` est exécutable uniquement lorsque le drone est en vol, non crashé, non arrimé, et se trouve exactement à la position de la base. Ces préconditions garantissent que l'arrimage ne peut avoir lieu qu'au bon endroit et dans un état cohérent. Lorsque l'opération est exécutée, le drone passe à l'état arrimé (`docked := 1`) et quitte l'état de vol (`airborne := 0`). Cette opération ne modifie ni la position du drone, ni le niveau de batterie, ni les zones déjà couvertes. Elle marque simplement l'entrée du drone dans une phase stationnaire à la base.

Opération `undock_cmd`. L'opération `undock_cmd` modélise le redécollage du drone depuis la base. Elle n'est exécutable que si le drone n'est pas crashé, est actuellement arrimé, et dispose d'une batterie strictement positive. Lors de l'exécution, le drone quitte l'état arrimé (`docked := 0`) et repasse en vol (`airborne := 1`). La position du drone reste inchangée et correspond toujours à la base. La zone contenant la base est ajoutée à l'ensemble des zones couvertes, ce qui garantit que la reprise de mission tient compte de la position de départ.

2.6 Opération `charge_step`

L'opération `charge_step` modélise la recharge progressive de la batterie lorsque le drone est arrimé à sa base. Elle n'est exécutable que si le drone n'est pas crashé et se trouve dans l'état arrimé (`docked = 1`). Lors de l'exécution de cette opération, le niveau de batterie du drone augmente de manière incrémentale. Si la batterie est inférieure ou égale à 95, sa valeur est augmentée de 5 unités. Dans le cas contraire, la batterie est directement fixée à sa valeur maximale, soit 100.

Cette opération n'entraîne aucun déplacement du drone, ni modification de son état de vol ou de son état de crash. Elle modélise une phase stationnaire au cours de laquelle le drone reste à la base et récupère progressivement de l'énergie avant de pouvoir reprendre une mission de patrouille.

charge_step =
  PRE
    $crashed = 0 \wedge docked = 1$
  THEN
    IF $battery \leq 95$ **THEN**
      $battery := battery + 5$
    ELSE
      $battery := 100$
    END
  END;

2.7 Opération `emergency_stop`

L'opération `emergency_stop` modélise une situation de défaillance critique du drone, correspondant à un crash ou à un arrêt d'urgence. Contrairement aux autres opérations, elle ne possède pas de précondition explicite : elle peut être déclenchée à tout moment par l'environnement lorsque la situation l'exige. Lors de l'exécution de cette opération, l'état du drone est mis à jour de manière irréversible :

- le drone passe dans l'état crashé (`crashed := 1`) ;
- il quitte l'état de vol (`airborne := 0`) ;
- il n'est plus considéré comme arrimé à la base (`docked := 0`) ;
- le danger est fixée à une valeur indiquant un danger immédiat (`obstacle_distance := 0`).

Cette opération représente un état terminal du point de vue de la mission de patrouille. Une fois déclenchée, aucune action normale de déplacement, de décollage ou de recharge n'est plus possible. Elle permet de modéliser explicitement l'échec de la mission dans l'environnement d'apprentissage.

emergency_stop =
  BEGIN
    $crashed := 1$ ||
    $airborne := 0$ ||
    $docked := 0$ ||
    $obstacle\_distance := 0$
  END;

3 Objectifs de l'agent et apprentissage

L'objectif de l'apprentissage par renforcement dans ce projet est d'apprendre un comportement autonome de patrouille efficace pour le drone. L'agent ne contrôle pas directement les variables internes de l'environnement, mais choisit à chaque étape une action parmi celles autorisées par l'état courant (déplacement, évitement, retour à la base, recharge, etc.). L'environnement se charge ensuite d'appliquer la dynamique définie par la spécification.

3.1 Objectifs de patrouille

Le but principal du drone est de maximiser la couverture de la zone surveillée. Plus précisément, l'agent doit apprendre à visiter un maximum de zones de patrouille distinctes au cours d'un épisode.

Revisiter plusieurs fois des cellules appartenant à une même zone n'apporte aucun bénéfice supplémentaire du point de vue de la couverture. Cet objectif incite l'agent à explorer l'environnement de manière efficace, en évitant de rester confiné dans une région limitée de la grille.

3.2 Gestion de l'énergie

Le drone dispose d'une batterie limitée qui se décharge lors des déplacements et des manœuvres d'évitement. L'agent doit apprendre à gérer cette ressource de manière autonome, en décidant quand poursuivre la patrouille et quand initier un retour vers la base pour se recharger. Un retour trop tardif peut conduire à une situation dangereuse ou à l'impossibilité de poursuivre la mission, tandis qu'un retour trop précoce limite la couverture de zones. L'agent doit donc apprendre un compromis entre exploration et préservation de l'énergie.

3.3 Gestion du risque et sécurité

Certaines cellules de la grille sont dangereuses et exposent le drone à un risque de crash. L'agent dispose d'actions lui permettant soit de se déplacer librement, soit de privilégier des manœuvres d'évitement plus coûteuses en énergie. Il doit apprendre à utiliser ces actions de manière pertinente, en fonction de la situation. Un crash met fin à l'épisode et correspond à un échec de la mission. L'agent doit donc apprendre à limiter les situations à risque tout en poursuivant ses objectifs de patrouille.

3.4 Compromis et comportement global

Le problème étudié dans ce projet repose sur un compromis entre plusieurs objectifs parfois antagonistes :

- couvrir un maximum de zones de patrouille ;
- éviter les zones dangereuses et limiter le risque de crash ;
- gérer efficacement la batterie et les phases de recharge ;
- limiter les déplacements inutiles.

L'apprentissage par renforcement permet à l'agent de découvrir progressivement une stratégie équilibrée, adaptée à la structure de l'environnement et aux contraintes imposées par la dynamique du drone.

4 Objectifs du projet : implémentation de l'environnement

Dans ce projet, l'objectif principal n'est pas d'implémenter un algorithme d'apprentissage par renforcement, mais de réaliser un **environnement de simulation** conforme à la spécification fournie. Cet environnement servira ensuite de support expérimental pour des agents (humains ou automatiques) et pourra être utilisé, dans un second temps, pour de l'apprentissage par renforcement.

L'environnement doit permettre de simuler l'exécution des opérations métier du drone (décollage, déplacements, évitement, retour à la base, arrimage, recharge, arrêt d'urgence), tout en garantissant le respect des invariants et des contraintes définies dans la spécification.

4.1 Modes d'exécution attendus

Votre implémentation devra proposer trois modes d'utilisation complémentaires.

4.1.1 Mode interactif (pilotage manuel)

Dans ce mode, l'utilisateur pilote le drone manuellement, en choisissant à chaque étape une action parmi celles autorisées par l'état courant (en respectant les préconditions). L'environnement applique l'action, met à jour l'état et affiche le nouvel état. Ce mode doit permettre de :

- vérifier facilement la cohérence de l'implémentation ;
- comprendre l'effet des différentes opérations sur l'état ;
- tester manuellement des scénarios (retour à la base, évitement, etc.).

4.1.2 Mode automatique (exécution d'une séquence d'actions)

Dans ce mode, l'environnement exécute automatiquement une **séquence d'actions** donnée en entrée (par exemple sous forme d'un fichier csv). L'environnement applique successivement les actions de la séquence tant qu'elles sont exécutables, et s'arrête si une action est invalide (précondition non satisfaite) ou si un état terminal est atteint (par exemple crash). Ce mode doit permettre de :

- rejouer des scénarios de test reproductibles ;
- comparer plusieurs stratégies codées manuellement ;
- faciliter l'évaluation et le débogage.

4.1.3 Mode exhaustif (exploration de l'espace d'états)

Dans ce mode, l'environnement explore de manière exhaustive l'espace des états atteignables, en appliquant systématiquement toutes les actions possibles depuis chaque état. L'objectif est de construire un graphe de transition (ou une représentation équivalente) permettant d'analyser :

- les états atteignables depuis l'état initial ;
- les transitions possibles et les éventuels blocages ;
- la couverture de l'espace d'états en fonction des paramètres (taille de grille, danger, zones).

L'exploration exhaustive doit gérer correctement le non-déterminisme de certaines opérations (par exemple `move_step` ou `avoid_maneuver`) en considérant l'ensemble des successeurs possibles. Ce mode constitue une étape importante pour valider l'environnement : il permet de vérifier systématiquement la conformité de l'implémentation à la spécification, et prépare l'utilisation ultérieure de cet environnement dans un cadre d'apprentissage par renforcement.