



---

## LINFO1104 : LethOz Company

---



Année académique 2023 - 2024

**Enseignant :** PETER Van Roy

**Cours :** LINFO1104

**Assistants :**

Crochet Christophe,

Clément Delzotti,

Evrard Colin

## Table des matières

<b>1</b>	<b>Overview</b>	<b>1</b>
1.1	Support multijoueur . . . . .	1
<b>2</b>	<b>Contexte du jeu</b>	<b>1</b>
<b>3</b>	<b>Fonctionnalités de base</b>	<b>1</b>
<b>4</b>	<b>Instructions</b>	<b>1</b>
4.1	Objectifs de base . . . . .	2
4.2	Extensions . . . . .	5
<b>5</b>	<b>Directives de soumission</b>	<b>7</b>
5.1	Critères d'évaluation . . . . .	7
5.2	Deadline . . . . .	7
<b>6</b>	<b>Notes supplémentaires</b>	<b>8</b>

## 1 Overview

“LethOz Company” est un jeu stratégique multijoueur dans lequel les joueurs dirigent des vaisseaux spatiaux dans un champ d’astéroïdes, récupérant des ressources pour le compte de la grande compagnie minière intergalactique *Large Ethereum Organization of Zadristan* mieux connue sous le diminutif *LethOz*. L’objectif est de développer la compagnie afin de collecter des déchets, d’éviter les dangers et d’utiliser stratégiquement les bonus pour surpasser les adversaires.

Ce projet utilise la nature multi-paradigme du langage de programmation Oz, en utilisant uniquement la programmation déclarative, la gestion de l’état du jeu et l’application d’effets dynamiques.

### 1.1 Support multijoueur

Le jeu supporte plusieurs joueurs, chacun contrôlant son vaisseau avec des stratégies et des objectifs distincts.

## 2 Contexte du jeu

- Le jeu se déroule dans un environnement spatial dynamique et hostile, chaque cellule de la grille représentant un espace rempli de déchets (*scraps*), d’astéroïdes ou de bonus spéciaux.
- Les joueurs contrôlent des vaisseaux pour collecter des déchets pour la Compagnie, éviter ou placer stratégiquement des charges sismiques pour perturber les adversaires, et collecter des bonus pour améliorer leurs capacités.
- En tant que joueur, l’un de vos objectifs est de collecter autant de déchets que possible. Mais cela a un coût, car chaque rebut prend de la place. Ces déchets s’ajouteront à la fin de la cargaison de votre spaceship, d’une manière très similaire aux trains de marchandises et aux wagons de l’époque moderne. Par conséquent, votre spaceship devra éviter d’entrer en collision avec sa propre cargaison et, bien sûr, avec d’autres vaisseaux !

## 3 Fonctionnalités de base

1. **Mouvement des vaisseaux et décodage des stratégies** : Implémenter la logique de navigation des vaisseaux dans l’environnement spatial hostile, y compris un mécanisme de décodage de stratégie permettant aux joueurs de prédéfinir leurs tactiques de navigation.
2. **Application d’effets dynamiques** : Concevoir une variété d’effets tels que des boosts de vitesse, des boucliers et des capacités de distorsion qui peuvent être appliqués aux vaisseaux pour affecter le gameplay.
3. **Gestion de l’état du jeu** : Développer la logique du backend pour gérer la progression du jeu, y compris le suivi des ressources extraites, la gestion des charges sismiques et la résolution des interactions entre les joueurs.

## 4 Instructions

Le projet doit être réalisé en binôme, chaque équipe devant mettre en œuvre le décodage de la stratégie et les trois effets supplémentaires essentiels décrits ci-dessous.

Une fois que vous aurez maîtrisé les aspects fondamentaux, vous serez encouragés à améliorer le jeu en y incorporant un **minimum de deux fonctionnalités supplémentaires**.

Bien qu'il n'y ait pas de limite au nombre d'extensions que vous pouvez introduire, *la priorité doit être donnée au respect des exigences du projet de base en premier lieu.*

N'oubliez pas que l'ajout de nombreuses extensions ne peut pas compenser un projet fondamentalement défectueux ou mal exécuté.

Votre évaluation sera basée sur la qualité de votre code et de votre rapport. Nous attendons de votre code qu'il soit correct, bien organisé (décomposé en sous-problèmes plus petits et bien commentés) et aussi efficace que possible (pensez à utiliser la récursivité de queue). Nous vous conseillons de vous en tenir aux caractéristiques déclaratives du langage, **pas de Cellule ou de Tableau**. Vous pouvez utiliser la bibliothèque standard (mais elle doit être déclarative).

Si vous vous écartez de ces directives, veuillez expliquer et justifier vos décisions dans le rapport d'accompagnement avec votre code et votre scénario de test.

Le partage de code entre différentes équipes est strictement interdit, sauf pour les scénarios. Le scénario de test que vous soumettez doit être unique et ne pas reproduire celui d'autres équipes. Toutefois, la discussion et l'échange d'idées sont autorisés et encouragés.

## 4.1 Objectifs de base

Le jeu se déroule sur une grille mesurant 24 carrés de largeur sur 24 carrés de hauteur.

- Chaque case est identifiée par les coordonnées  $(x, y)$  telles que  $(1 \leq x \leq 24)$  et  $(1 \leq y \leq 24)$ .
- La coordonnée  $(1, 1)$  marque le coin supérieur gauche, ou le nord-ouest, de l'aire de jeu.

Chaque case de la grille peut être occupée par des éléments d'un spaceship, des astéroïdes (servant d'obstacles), des bonus ou des combinaisons de ces éléments. Vous trouverez ci-dessous un exemple de ce à quoi le jeu peut ressembler à un moment donné :

Le jeu se déroule en plusieurs étapes, au cours desquelles les actions suivantes se produisent :

1. Les vaisseaux spatiaux déploient des charges sismiques comme prévu.
2. les minuteurs des charges sismiques sur la grille diminuent, et ceux qui atteignent zéro explosent.
3. **Les vaisseaux spatiaux se mettent à jour en fonction de leurs effets actuels et de l'instruction programmée pour ce tour** (c'est votre point d'intervention).
4. Les bonus sont appliqués aux astronefs et les astronefs détruits sont identifiés. Un astronef est détruit si
  - Il entre en collision avec un autre astronef, lui-même, un astéroïde ou un mur ;
  - Il est pris dans l'explosion d'une charge sismique ;
  - Il sort de la zone de jeu.

À ce stade, le moteur de jeu détermine également si une équipe peut être déclarée gagnante, ce qui met fin au jeu et proclame la victoire. Un joueur gagne lorsqu'il n'y a plus d'autre joueur, ou qu'il a récupéré 10 scraps/déchets.

Le *scenario* de chaque partie définit les positions de départ des astronefs sur la grille, ainsi que l'emplacement des bonus et des éventuelles charges sismiques placées dès le départ. Chaque *spaceship* occupe plusieurs positions à la fois, chacune ayant une orientation par rapport à l'une des directions cardinales : **north**, **south**, **west**, ou **east**. La première position indiquée est l'avant du vaisseau (la tête), et la dernière est l'arrière (la fin de la queue).

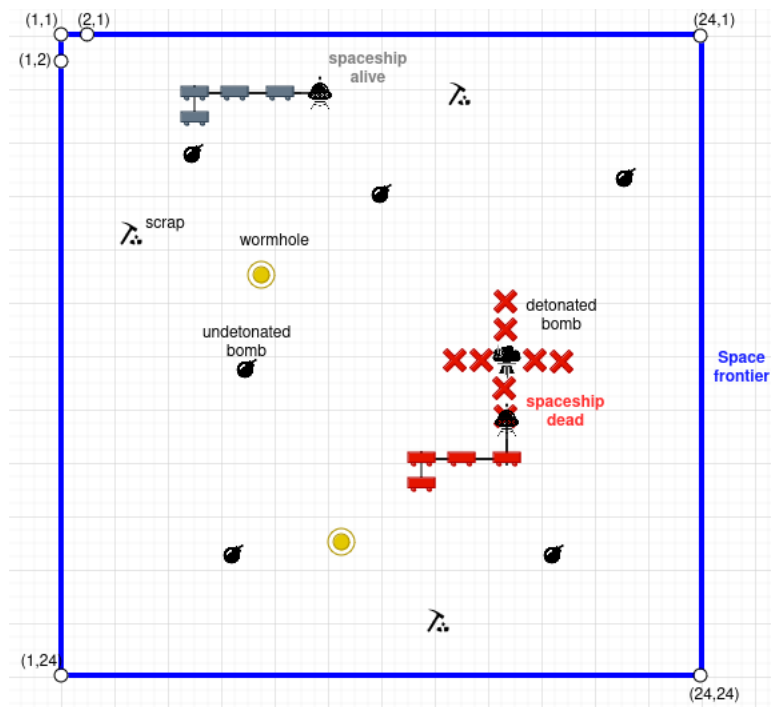


FIGURE 1 – Exemple à un moment donné

```
P      ::= <integer such as 1 <= x <= 24>
direction ::= north | south | west | east
```

```
spaceship ::= spaceship(
  positions: [
    pos(x:<P> y:<P> to:<direction>) % Head
    ...
    pos(x:<P> y:<P> to:<direction>) % Tail
  ]
  effects: nil
)
```

Chaque *spaceship* suit une stratégie prédéfinie à chaque étape, qui l'oblige à exécuter une instruction : `forward`, `turn(left)`, ou `turn(right)`. L'exécution de cette instruction consiste à réorienter l'avant du vaisseau dans la direction souhaitée et à le faire avancer d'une case dans cette direction. C'est le but de la fonction `Next`.

```
relative_direction ::= left | right
instruction          ::= forward | turn(<relative_direction>)
```

Le moteur de jeu, pour chaque vaisseau, attend une stratégie décodée sous la forme d'une **simple liste de fonctions**. Chaque fonction prend un seul argument, un *spaceship*, et renvoie un nouveau *spaceship* avec des attributs mis à jour. La fonction  $(n^{\{th\}})$  de la stratégie décodée de chaque *spaceship* est appliquée à l'étape  $(n^{\{th\}})$  du jeu. Vous devez développer la fonction `DecodeStrategy` pour convertir les stratégies de la forme suivante

```
strategy ::= <instruction> '|' <strategy>
           | repeat(<strategy> times:<integer>) '|' <strategy>
           | nil
```

en une liste de fonctions. Chaque fonction doit utiliser votre fonction `Next` pour mettre à jour le vaisseau spatial selon les instructions de l'étape. Cette approche, y compris la syntaxe de répétition des sous-stratégies avec `repeat(...)`, nécessite une réflexion approfondie.

**Résumé** Commencez par implémenter deux fonctions dans `Code.oz` :

- `fun {Next Spaceship Instruction} ... end`, appliquant `Instruction` sur le vaisseau spatial `Spaceship` et renvoyant le vaisseau spatial mis à jour. Le vaisseau retourné devrait avoir son front réorienté, et se déplacer selon l'instruction.

Par exemple, pour les arguments

- `spaceship(positions:[pos(x:4 y:2 to:east) pos(x:3 y:2 to:east) pos(x:2 y:2 to:east)] effects:nil)`
- `turn(right)`

le résultat devrait être `spaceship(positions:[pos(x:4 y:3 to:south) pos(x:4 y:2 to:south) pos(x:3 y:2 to:east)] effects:nil)`.

Les détails sur les effets seront présentés dans la section suivante.

- `fun {DecodeStrategy Strategy} ... end`, avec `Strategy` défini comme précédemment. `DecodeStrategy` transforme la stratégie en une liste de fonctions de type `fun {$ Spaceship} ... spaceship(...) end`. Chaque fonction met à jour le vaisseau pour l'instruction à ce pas de jeu.

Par exemple, `{DecodeStrategy [repeat([turn(right)] times:2) forward]}` produit une liste de 3 fonctions `fun {$ Spaceship} ... spaceship(...) end`, les deux premières appliquant `turn(right)` au vaisseau et la troisième appliquant `forward`. Ces fonctions doivent appeler `Next`.

**Effets de la Compagnie** Afin de vous aider dans votre tâche de collecte de ressources, la compagnie vous assiste en dispersant différents bonus, appelés *effets*, sur la grille de jeu. Ces effets sont ajoutés à la liste des *effects* d'un vaisseau spatial par le moteur du jeu lorsque son front atteint un bonus. Il est de votre responsabilité d'appliquer ces effets et de les retirer une fois prises en compte par votre fonction `Next`.

Le moteur de jeu ne fait qu'ajouter des éléments à cette liste, et les effets **doivent être appliqués dans l'ordre**. Si plusieurs effets du même type sont présents, seul le premier doit s'appliquer ; les autres doivent être ignorés et supprimés.

1. **scrap** : L'astronave s'étend d'une case à la fin de son mouvement. L'avant se déplace comme d'habitude.




Steps											
1				2				3			
											

FIGURE 2 – Tableau : Evolution d'un vaisseau spatial vers l'avant affecté par l'effet **scrap** à l'étape 2

2. **revert** : Le vaisseau spatial s'inverse avant d'exécuter l'instruction de déplacement, transformant son arrière en avant et vice versa, y compris l'inversion de direction.
3. **wormhole(x:X y:Y)** : L'avant du vaisseau, lorsqu'il atteint un bonus, est téléporté à la position (X, Y) avant d'exécuter l'instruction de déplacement, en conservant son orientation.




Steps		
1	2	3
		

FIGURE 3 – Tableau : Evolution d'un vaisseau spatial vers l'avant, affecté par l'effet revert à l'étape 2.




Steps		
1	2	3
		

FIGURE 4 – Tableau : Evolution d'un vaisseau spatial vers l'avant, affecté par l'effet wormhole(...) à l'étape 2..

**Tips** Cette base du projet suggère de commencer par des instructions de base pour s'assurer que vos *spaceships* répondent correctement au scénario `scenario_test_moves.oz`, qui ne contient pas de bonus. Ensuite, attaquez les effets `scrap`, suivies de `wormhole(...)` et `revert`, en testant avec leurs scénarios respectifs.

Toutefois, la réussite de ces tests ne garantit pas l'exactitude des résultats. Expérimentez en écrivant vos scénarios et en testant le jeu de manière interactive pour explorer d'autres possibilités de jeu.

Nous vous encourageons à partager vos scénarios sur le forum Moodle, mais *en aucun cas celui que vous avez l'intention de présenter*.

## 4.2 Extensions

Conformément aux lignes directrices du projet, vous devez développer au moins deux fonctionnalités supplémentaires. Même si c'est votre créativité qui fixe les limites, nous vous proposons quelques suggestions pour vous inspirer ou vous orienter vers d'autres innovations.

Tout le code qui vous est fourni fait partie du projet. C'est à dire qu'on attend de vous que vous compreniez comment le projet fonctionne, quel fichier est responsable de quelle fonctionnalité et comment ces derniers interagissent au niveau du code. En particulier, pour tester votre code et implémenter de nouvelles fonctionnalités, vous devrez probablement lire le contenu du fichier `project_library.oz`, qui contient une grande partie des fonctions utiles au projet.

Pour évaluer efficacement vos extensions, vous devrez élaborer un nouveau scénario ou modifier un scénario existant, en incorporant ou en changeant les bonus si nécessaire.

Ces bonus comprennent :

- les attributs tels que la position, la `color` (pour la distinction visuelle),
- les effets appliquées aux vaisseaux spatiaux,
- et l'attribut `target` qui détermine quel astronef est affecté par le contact.

Communément, `catcher` est assigné comme valeur `target` dans de nombreux scénarios, appliquant le bonus à l'astronef qui le déclenche. Les alternatives incluent `others`, `allies`, `opponents`, et `all`, élargissant les stratégies potentielles.

```
color ::= <any colors recognized by Tk, see https://www.tcl.tk/man/tcl8.4/TkCmd/colors.htm>
bonus ::= bonus(position: pos(x:<P> y:<P>)
               color: <color>
               effect: boost | wormhole(x:<P> y:<P>) | reverse | ...
               target: catcher | others | allies | opponents | all
            )
```

**Implémentation d'une grille enveloppante** Envisagez de concevoir des scénarios sans barrières périphériques (attribut `walls` fixé à `false` dans la configuration de votre `scenario` (voir par exemple `scenario_test_moves.oz`) où les vaisseaux spatiaux passent de manière transparente à travers les bords de la grille, réapparaissant de l'autre côté.

Ce mécanisme empêche les vaisseaux d'être détruits lorsqu'ils atteignent la limite, un peu comme l'effet du bonus de téléportation.

**L'effet `shrink`** Bien qu'il existe des effets de scrap, l'introduction d'un effet de `shrink` pourrait ajouter un nouveau niveau de difficulté, les vaisseaux spatiaux risquant de devenir trop petits pour rester viables.

**L'effet `shield(N)`** Il procure une invincibilité temporaire contre les charges sismiques et les collisions.

**L'effet `emb(N)` (ElectroMagnetic bomb)** Cet effet immobilise le vaisseau spatial pendant `N steps`, sans tenir compte des instructions données et en annulant potentiellement d'autres effets actifs.

**L'effet `malware(N)`** L'inversion des instructions `turn(left)` et `turn(right)` pour `N steps` pourrait introduire des éléments humoristiques et chaotiques, en particulier dans les scénarios multijoueurs.

**L'effet `time_warp`** Un bonus rare qui permet à un joueur de remonter le temps d'un tour, en annulant les actions de tous les joueurs. Cet effet complexe nécessiterait de suivre l'état de la partie d'un tour à l'autre, ce qui permettrait de revenir à l'état précédent. Son utilisation pourrait être limitée pour maintenir l'équilibre.

**Arènes multi-grilles** Développez le gameplay au-delà d'une seule grille en introduisant plusieurs grilles ou niveaux interconnectés entre lesquels les joueurs peuvent se déplacer. Les vaisseaux spatiaux peuvent trouver des portails qui les transportent vers différentes grilles, chacune offrant des défis et des bonus uniques.

**Élargir l'horizon** Pour exploiter pleinement le potentiel du jeu, nous vous encourageons à vous inspirer de scénarios existants ou à concevoir des effets, des stratégies ou des éléments de jeu entièrement nouveaux. Les vaisseaux spatiaux peuvent avoir un identifiant d'équipe (`team`, n'importe quelle couleur reconnue par Tk) et une stratégie de charge sismique (`seismicCharge`, une liste de booléens consommée progressivement pour déterminer le déploiement des bombes).

Votre créativité peut s'étendre à la modification des attributs des vaisseaux via la fonction `Next`, de la même manière que pour les `positions` et les `effets`. Par exemple, un vaisseau doté d'un effet `shield` à la fin d'un tour peut survivre à des situations fatales, alors qu'un vaisseau affligé d'un effet `death` sera éliminé pour toute la durée du jeu.

N'hésitez pas à inventer de nouvelles instructions ou structures syntaxiques pour les stratégies, en veillant à ce qu'elles complètent les mécanismes du jeu de base. Ces innovations pourraient transformer le jeu en une



nouvelle expérience rappelant *TRON* ou d'autres jeux emblématiques, à condition qu'elles n'interfèrent pas avec les stratégies et les effets de base.

## 5 Directives de soumission

Soumettez votre projet sous la forme d'une archive zip contenant les fichiers suivants :

1. **Code.oz** : Votre code Oz implémentant la logique du jeu. Vous avez à votre disposition deux exemples d'agent `Crazy.oz` et `Normal.oz`.
2. **Rapport.pdf** : Un rapport détaillé discutant de vos choix de conception, des difficultés rencontrées, de l'analyse de la complexité de vos fonctions, et des descriptions de vos extensions implémentées. Le rapport ne doit pas dépasser 5 pages.
3. **Scenario.oz** : Un scénario de jeu entièrement automatisé démontrant toutes les fonctionnalités et extensions que vous avez implémentées.

Veillez à ce que votre scénario mette en valeur les aspects uniques de votre projet.

### 5.1 Critères d'évaluation

- **Qualité du code** : Votre code sera évalué en fonction de son efficacité et de son respect des principes de programmation déclarative.
- **Lisibilité et clarté du code** : Votre code ne sera pas évalué de façon automatique. Cela implique que nous lirons votre code en plus de le lancer. De la même façon qu'un professeur de droit attend une grammaire impeccable et une écriture lisible dans une dissertation, nous attendons de vous que vous teniez compte du fait que votre code doit être facile à comprendre **du premier coup d'oeil** par un tiers. Un code illisible et/ou incompréhensible sera donc pénalisé. Voici quelques conseils pour vous aider à produire du code clair.
  1. **Indentez et espacez votre code** : Idéalement, chaque structure de contrôle ajoute un niveau d'indentation (semblable au python). Groupez les lignes de code qui vont ensemble, séparez les groupes par un ou plusieurs sauts de ligne.
  2. **Commentez votre code** : Chaque fois que le lecteur de votre code pourrait se poser des questions sur l'utilité d'une partie d'un code, écrivez un commentaire succinct décrivant ce que vous essayez d'accomplir. De même, chaque fonction/procédure devrait être munie d'une description des inputs qu'elle attend, et des éventuels outputs attendus. Veillez cependant à rester concis, nous n'attendons pas de vous que vous écriviez le prochain opus de *Game of Thrones*.
  3. **Séparez votre code en sous-problèmes** : Peut-être êtes-vous familiers avec le concept de *code smell* ? Il s'agit de caractéristiques du code indicatrices que votre approche est mauvaise. Un de ces indicateurs est la présence de *fonctions gigantesques* possédant un nombre absurde d'arguments. Une bonne heuristique pour éviter cela est de toujours essayer de décomposer votre fonction en sous-problèmes qui eux-mêmes seront résolus par leur propre fonction. Le code résultant ne devrait pas présenter de fonctions de plus d'une vingtaine de lignes. Il peut évidemment y avoir des exceptions comme les instructions `case of` qui prennent vite beaucoup de lignes.
- **Qualité du rapport** : Votre rapport doit présenter en détail la conception de votre projet, les défis qu'il pose et l'approche que vous avez adoptée pour les surmonter.
- **Innovation et complexité** : Les implémentations qui vont au-delà des exigences de base et qui font preuve d'une pensée innovante seront très appréciées.

### 5.2 Deadline

Votre projet doit être soumis au plus tard le **29 avril 2024** à 23h55. Les soumissions tardives seront possibles mais pénalisées.

## 6 Notes supplémentaires

- Bien que la collaboration sur les idées soit encouragée, le partage de code entre les groupes est strictement interdit.
- N'hésitez pas à utiliser le forum du cours pour les discussions et les questions. Cependant, évitez de partager des implémentations de code spécifiques.

Bonne chance, et nous attendons avec impatience de voir vos stratégies innovantes d'exploitation minière de l'espace et vos implémentations de jeux !