

Sujet de Projet PAF 2024: *Sim City*

PAF 2022 - Master STL S2

Version du 04/03/2024



Ce document présente le sujet du projet de l'UE PAF 2024 et contient les attendus, la description du projet, une liste des extensions possibles et des consignes pour le rapport. Ce document complète le sujet d'Examen Réparti 1 du 04 mars.

1 Objectifs et Rendus

1.1 Objectifs

L'objectif principal de ce projet est le développement d'un **jeu** vidéo **sûr**, dans le langage **Haskell**. L'évaluation portera sur ces trois points:

1. programmation **sûre**: les types et les fonctions associées du projet doivent toutes être accompagnées de propositions d'**invariants**, de **pré-conditions**, et de **post-conditions**. Les opérations des entités doivent toutes être testées en utilisant, de façon raisonnablement complète, les **tests unitaires** (*HSpec*) et les **tests basés sur les propriétés** (*Quickcheck*).
2. programmation **fonctionnelle**: le langage Haskell est imposé, l'application doit être écrite dans un style fonctionnel, une attention doit être donnée à la séparation de la logique (évolution de l'état du jeu) et les effets de bords (boucle de jeu, entrée/sortie, affichage). L'utilisation de spécificités du langage Haskell est encouragée: types définis comme sommes d'états remarquables, instanciation de classes de types, utilisation des structures algébriques (**Functor**, **Applicative**, **Monad**).
3. programmation **graphique**: le jeu doit utiliser la bibliothèque SDL2 afin de proposer une expérience visuelle, intuitive et fluide pour le joueur. Le jeu doit avoir un intérêt ludique non-négligeable.

Le projet sera évalué sur ces trois critères, ce qui veut dire, entre autres, que le projet peut s'éloigner des caractéristiques proposées dans ce document, tant qu'il satisfait les deux critères. Aucune définition (en particulier, les définitions de l'ER1) n'est imposée.

1.2 Rendu

Le projet est à faire en binôme (monôme possible avec autorisation de l'équipe enseignante, trinôme non-autorisé). Le rendu consiste en un projet GitLab contenant un rapport. Les enseignants doivent y être ajoutés comme *maintainer*.

La deadline est le ???.

2 Description du Jeu

(repris du partiel 2024:)

L'objectif est de réaliser un jeu vidéo de type *city builder* sur une grille, similaire à *Sim City* (Maxis, 1989).

Le jeu se déroule sur une carte vue du dessus, le haut de l'écran représente le Nord. Le joueur place sur la carte des routes, des bâtiments administratifs et des zones (résidentielles, commerciales, industrielles) afin de construire et faire évoluer une *ville*.

Une *simulation* gère en temps réel la vie des citoyens de la ville *individuellement*, qui dorment, travaillent et font leurs achats dans différents bâtiments au sein des zones de la ville.

Les citoyens se déplacent en temps réel dans la ville, ce qui génère du trafic routier, qui peut conduire à des embouteillages et force le joueur à planifier à l'avance l'urbanisation de sa ville.

Des systèmes de contraintes et d'évolution rendent le jeu ludique, par exemple, une économie (les citoyens reçoivent un salaire, le dépense en loyer, nourriture et impôts, qui sont utilisés par le joueur pour construire de nouvelles infrastructures), des contraintes de services (des réseaux d'électricité et d'eau courante doivent être mis en place, des policiers patrouillent dans la ville), une notion de qualité des zones (la "valeur" d'une zone peut évoluer en fonction de différents facteurs locaux, ce qui a un impact sur les loyers et les salaires de cette zone)

On pourra chercher sur le Web des documents plus ou moins théoriques décrivant ce style de jeux vidéo. Par exemple :

- *Wikipédia*: https://en.wikipedia.org/wiki/City-building_game
- *MicropolisJS* : <https://www.graememcc.co.uk/micropolisJS/>

2.1 Fonctionnalités

- l'état du jeu (carte, bâtiments, citoyens à l'extérieur des bâtiments, animations) doit être affiché à l'aide d'une interface graphique (en SDL2), qui représente la zone de jeu, "vue de dessus",
- les éléments de simulation décrits dans l'ER1 doivent être implémentés.
- les utilisateurs peuvent construire des bâtiments administratifs et des routes, et délimiter des zones R/C/I à la souris et au clavier..
- les extensions suivantes doivent être implémentée :
 - **Economie** : Des impôts (réglables) sur les sociétés et les citoyens rapportent de l'argent au joueur. Le joueur utilise cet argent pour construire des infrastructures.
 - **Services** : (au moins) une centrale électrique doit être construite par le joueur, et l'électricité doit être acheminée (par contact entre zone, ou par des constructions spéciales) aux différentes zones pour qu'elles se développent. Des commissariats envoient régulièrement des véhicules de patrouille qui se déplacent aléatoirement (ou non) dans la ville.
 - **Qualité de vie** : des caractéristiques doivent être ajoutées aux bâtiments des zones en fonction de paramètres locaux (par exemple pour un bâtiment résidentiel, le nombre d'habitant, leur richesse, la sécurité de la zone, la proximité avec des usines polluantes, ...). Ces caractéristiques permettent

aux zones d'évoluer positivement ("cabane" devient "maisonette" et le changement s'accompagne d'une hausse des loyers et de la qualité de vie) ou négativement.

- **Pathfinding** : Les citoyens savent comment se déplacer efficacement d'un point à un autre, en évitant les bouchons, si possible.
- **Immigration et Emigration** : Des places (logements, emplois) disponibles dans une ville attractive attirent les immigrants. A l'inverse, les citoyens trop fatigués, affamés ou sans argent quittent la ville. Une ou plusieurs zone *d'interface* peuvent être créés pour représenter les points de départ/arrivée des immigrants/émigrants.

3 Contenu et Extensions

3.1 Consignes

Pour obtenir une **note satisfaisante** à l'évaluation, il faut que le projet respecte les contraintes suivantes:

- implémentation des **fonctionnalités** (listées ci-dessus) en Haskell + SDL2 (ou une modification du jeu de contenu équivalent),
- rédaction de **propriétés** pre/post/inv pour les types et les opérations du jeu,
- test systématique des propriétés pre/post/inv incluant **une utilisation pertinente de tests basés sur les propriétés** (*Quickcheck*).
- quelques extensions basiques, afin de proposer un contenu ludique minimal.

Pour obtenir la **note maximale**, le projet doit inclure de multiples extensions parmi celles proposées ou non (c'est encore mieux si elles sont originales), une utilisation pertinente **du test basés sur les propriétés** et **des classes algébriques**.

La motivation des extensions est importante: il faudra montrer ce qu'elles apportent au jeu, mais surtout ce qu'elles apportent au projet: complexité de la rédaction de propositions, élégance du code fonctionnel utilisé, défi pour obtenir des tests complets, ... Une extension qui ne contiendrait pas de contenu "logiciel sûr" (propositions et tests) ou de contenu "programmation fonctionnelle" (utilisation de construction propres à la programmation fonctionnelle ou à Haskell pour la mettre en oeuvre) n'ajouterait rien à l'évaluation.

3.2 Tests Basés sur la propriété

Le projet doit contenir une utilisation pertinente de *QuickCheck*, conforme à celles proposées en TME. Un soin particulier sera apporté à l'efficacité des générateurs (cf. le sujet de TME sur l'utilisation de **suchThat**).

La rédaction et l'utilisation de générateurs complexes pour des types non-triviaux est fortement encouragée. Dans tous les cas, les tests doivent être décrits dans le rapport et la méthode de rédaction des générateurs doit y être explicitée.

3.3 Classes de types

Les projets doivent rendre explicite la maîtrise des classes de types :

- manipulation correcte, élégante et efficace de **Maybe**, **Either**, **IO**, ... ,
- instanciation de classes algébriques par des types définis dans le projet (par exemple **instance Monad Machin where ...**) et utilisation des bénéfices apportés par l'instanciation (primitives, *do-notations*, compositions, ...)

La programmation "avancée" avec les algébriques (rédaction de *transformers*, ...) est attendue pour une évaluation maximale.

3.4 Extensions

Voici quelques extensions possibles renforçant l'intérêt du projet en tant que "jeu vidéo" (on rappelle que les extensions originales sont encouragées) :

Transports. Une gestion poussée des transports est proposée, avec plusieurs modes pour les citoyens (à pied, en voiture, en bus, en train, en avion, ...) et une gestion fine des encombrements des voies, des intersections, des potentiels accidents, ...

Services supplémentaires. Des établissements scolaires (de différents types) permettent d'éduquer les citoyens et de les faire accéder à des emplois plus rémunérateurs. Les citoyens peuvent tomber malade ou mourir et le joueur peut construire des hôpitaux et des cimetières. Des déchets sont produits par les citoyens, qui doivent ensuite être traités par des services spécialisés. Tous ces services génèrent du trafic sur les routes.

Commerce et Industrie. Les biens produits par les industries locales, ou importés sont vendus dans les commerces locaux, ou exportés. Les citoyens ont des besoins spécifiques en biens de consommation, liés à leur niveau de vie.

Désastres. Des incendies (et/ou des tornades, des tsunamis, des ouragans) se produisent de manière aléatoire (ou non), forçant le joueur à agir de manière proactive (construction de casernes de pompiers) et réactive (reconstruction).

Environnement local comonadiques. La "qualité" de vie d'un bâtiment dépend de plusieurs facteurs dans son entourage. Cette approche suggère une utilisation des **Comonad** pour traiter ces transformations à haut niveau.

Inspiration On pourra s'inspirer de jeux existants pour ajouter des fonctionnalités au projet :

- *SimCity* (Maxi, 1989): une série de jeux de référence pour le *city building*. Le premier est disponible dans un navigateur dans une version libre (appelée *MicropolisJS*). Attention, à l'exception du dernier jeu de la série (*SimCity 2013*), ces jeux ne simulent pas individuellement le comportement de chaque citoyen (et le dernier jeu de la série le fait d'une manière "originale").
- *Caesar II* (*Impressions Games*, 1995) (et tous les jeux de la série : *Zeus Pharaoh*, *Emperor*): des *city-builders* historiques, qui ne sont pas des simulations totales, mais qui ont introduit la distribution de service par des entités qui "patrouillent" sur les routes, et affectent les bâtiments adjacents à leur trajet.
- *Cities: Skylines* (*Colossal Order*, 2015): deux *city-builders* récents qui proposent une simulation totale.
- ...

4 Rapport

Le rapport de projet se compose de quatre parties:

- un **manuel** d'utilisation succinct expliquant comment tester le jeu depuis le projet *stack*.
- une liste exhaustive des **propositions** (invariants/pré-conditions/post-conditions) implémentées dans le projet, classées par types / opérations et contenant une brève description de ce que vérifie la propriété.
- une description des **tests** implémentés par le **Spec.hs** du projet.
- un **rapport** proprement dit, qui décrit le projet, les extensions choisies et leur implémentation et qui, en plus, explicite les points importants de l'implémentation, en mettant en évidence du code, des propositions, des tests pertinents (ou des bugs mis découvertes grâce à la méthode de développement sûr).

De manière générale, il est fortement conseillé de consigner dans le rapport (sous la forme d'un paragraphe) toutes les difficultés rencontrées dans le développement du projet et tous les *points forts* (tests basés sur la propriété, utilisation d'algèbres, invariants de type complexe, ...) du projet afin de les mettre en valeur pour l'évaluation.

5 Barème Prévisionnel

Cette suggestion de barème n'est pas opposable, il s'agit d'un exemple possible pour la notation du projet, destiné à faire comprendre les priorités d'évaluation:

- 4 points pour **le rapport**: le rapport doit être clair, lisible, et contenir les informations nécessaires (manuel, propositions, tests). Deplus, le rapport doit présenter suffisamment de points intéressants du développement du projet.
- 4 points pour l'implémentation correcte d'un **jeu minimum** en SDL2.
- 2 points pour **les propositions**: la rédaction des invariants/pre-conditions/post-conditions est systématique, pertinente, et commentée.
- 2 points pour **les tests**: les tests sont complets, correctement structurés, utilisent la puissance du *property based testing* et commentés.
- 1 point pour l'utilisation pertinente **du test basé sur la propriété**.
- 1 points pour l'utilisation pertinente **des classes algébriques**.
- 6 points pour **les extensions**: un nombre suffisant d'extensions ont été choisies, implémentées dans un style fonctionnel et (autant que possible) "haskellien", augmentées de propositions et de tests adéquats (on rappelle qu'une extension qui n'est pas accompagnée de inv/pre/post et de tests n'est pas évaluée). La pertinence, la difficulté et l'originalité des extensions choisies sont prises en compte.