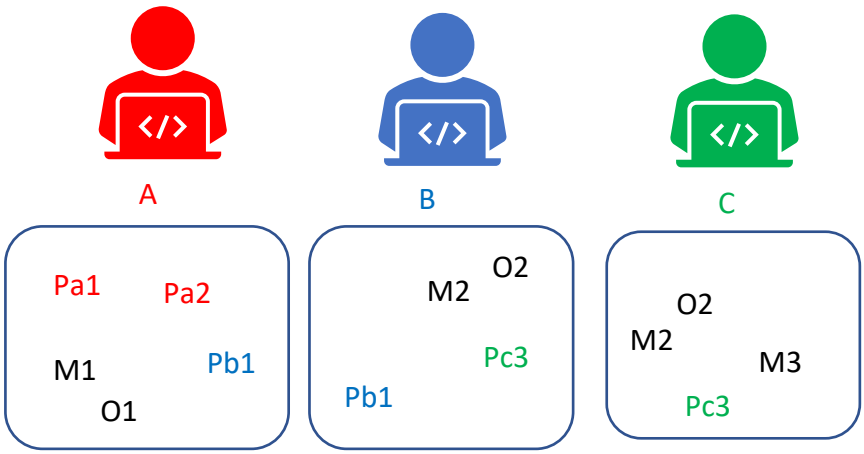
Répartition cohérente du jeu Donjon et Dragons

Christian Toinard

1. Introduction

L'objectif est d'étendre le projet de programmation précédant en permettant à différents joueurs d'entrer en compétition sur la base d'un système réparti à large échelle.
Pour cela vous allez travailler en groupe et choisissez d'étendre l'un des projets Python réalisés précédemment, vous pourrez également reprendre un projet 3A formation initiale de l'an dernier en citant le code repris et les modifications apportées.
La finalité est de permettre de répartir une partie de jeu entre les différents participants. Il ne faut **pas** introduire **le moindre serveur** ni de façon permanente ni de façon transitoire. Chaque joueur a une copie locale du jeu qui lui permet de visualiser la scène de jeu, d'interagir avec des objets, de combattre des monstres et éventuellement les personnages des autres joueurs.
Un joueur interagit localement avec son logiciel de jeu via son clavier et sa souris. Les actions d'un joueur produisent des changements d'état de la partie qui sont transmis aux copies distantes sans passer par un serveur. Afin de garantir une jouabilité satisfaisante, il faut que les **changements d'état** soient **observés de façon cohérente** en transmettant aux copies distantes les changements effectués localement.



2. Principe de répartition

L'objectif est de réaliser un vrai jeu en réseau sur une base entièrement répartie. La solution doit offrir un niveau de jouabilité importante en garantissant que chaque participant respecte les règles du jeu communes tout en disposant d'une jouabilité et d'une fluidité satisfaisantes.
On s'intéresse en priorité à la **cohérence du système réparti**.

a) Un joueur A dispose d'une entité du logiciel de jeu qui s'exécute sur son poste personnel. Il peut interagir via son clavier et sa souris avec les éléments (objet O1, personnages Pa1, Pa2, Pb1 et monstre M1) qui sont dans son champ de vision.
b) Deux joueurs B et C peuvent partager une même partie de la scène (M2, O2, Pc3). Dans ce cas, les personnages Pb1, Pc3 qui voient cette partie commune peuvent agir en **concurrence sur les éléments partagés** (M2, O2). Ainsi, Pb1 et Pc3 peuvent interagir en concurrence avec le monstre M2 ou l'objet O2.
c) Lorsqu'un personnage Pb1 interagit en concurrence avec Pc3 sur le même monstre M2 ou le même objet O2, le personnage Pb1 peut impacter une capacité M2c1 du monstre

M2 ou récupérer un composant O2c1 de l'objet O2 tandis que Pc3 impacte une autre capacité M2c2 de M2 ou récupère un autre composant O2c2 de O2. Cette concurrence est compatible avec la notion de tour pour les combats puisqu'il s'agit de deux combats [Pb1, M2] et [Pc3, M2] concurrents.

d) Deux **interactions concurrentes garantissent la cohérence** de l'élément. Par exemple, Pc3 ne peut impacter que la capacité M2c2 qui n'a pas été impactée par Pb1. Ainsi, l'élément M2 reste globalement intègre puisque Pc3 n'a pu impacter que la capacité M2c2 non impactée par Pb1. Si une partie de l'élément reste non modifiée alors celle-ci est disponible pour d'autres interactions. Par exemple, si le montre M2 dispose d'une troisième capacité M2c3 alors celle-ci reste disponible.
e) Un joueur observe rapidement les changements d'état pour la partie du monde qui est dans son champ de vision. Ainsi, B et C voient tous les deux le changement d'état du monstre M2 puisqu'ils ont chacun affecté une partie de ses capacités. B observe les modifications apportées par C à M2 et réciproquement.
f) La notion de rapidité est faite sur un mode moindre effort (best-effort). C'est-à-dire que lorsque deux joueurs observent une même partie du monde, il est acceptable qu'un joueur voit avec un certain retard les changements observés par l'autre joueur. Par exemple, le joueur C peut observer après B que la capacité M2c2 a été impactée par le personnage Pb1 appartenant à B. On accepte donc des décalages temporels dans la vision du monde des deux joueurs B et C.
g) Un joueur B qui observe des personnages d'un autre joueur C peut prendre connaissance des caractéristiques actuelles de ce joueur : compétences, inventaire, résistances, dégâts, état (points de vie, énergie, …), etc. Les caractéristiques de C peuvent s'afficher en temps réel ou faire l'objet d'une interaction du joueur B pour apparaître.
h) Pour garantir la **cohérence, chaque élément** (personnage, monstre, objet) ou partie de cet élément (exemple : épée contenue dans un coffre) **dispose d'un attribut de propriété** qui est **cessible**. Lorsqu'un joueur n'en dispose pas, il doit **demander la propriété**. La **propriété est transmise par son propriétaire en même temps que l'état** de l'élément ou partie d'élément. Ce mécanisme de transmission de la propriété et de l'état doit permettre à la fois la concurrence et la cohérence pour l'élément ou une partie de celui-ci. Pour cela, le **propriétaire** doit être **unique** à un **instant donné.** Le propriétaire dispose donc de l'état cohérent de cet élément ou de cette partie. Il peut donc modifier l'élément ou en céder la propriété avec un état cohérent à un joueur distant. Cependant, vous allez devoir réfléchir aux **contraintes** en termes **d'interface utilisateur**. **En** effet si un **joueur interagit** avec un élément dont il n'est pas propriétaire, il devra **attendre l'état cohérent avant** de décider ou non **de poursuivre son interaction**.

3. Jouabilité

Une des attentes essentielles est d'offrir une bonne jouabilité. Pour cela vous allez réfléchir aux fonctionnalités de votre jeu et aux évolutions de son interface. Vous devez justifier l'intérêt pour les joueurs et l'attractivité que cela donne à votre jeu multi-joueurs.

Bien que nous ne fixions pas de limite dans les fonctionnalités et qualités proposées, nous vous invitons notamment à proposer une réflexion et une mise en œuvre pour les points suivants :

- Définition des règles communes de jeu. Ces règles reposent sur celles existantes dans le jeu repris. Elles devront être partagées par tous les joueurs. Cependant, certaines règles

pourront évoluer ou être ajoutées. Par exemple, vous souhaitez permettre des combats entre les personnages de deux joueurs.

- Respect des règles communes. Votre logiciel doit appliquer le même ensemble de règles et ce sur l'ensemble des joueurs. Cela peut conduire par exemple à échanger entre les entités d'application (les différents joueurs) les règles partagées avant de commencer une partie.
- Participation à une partie. Vous permettez de créer une partie entre différents joueurs. Cela peut passer par exemple par un mécanisme d'invitation comme le principe d'une session zoom. Vous évitez le plus possible d'avoir à attendre que tous les joueurs soient présents pour commencer une partie. Ce qui signifie que vous autorisez de jouer tout en attendant l'arrivée des autres joueurs.
- Gestion des mondes et des donjons. Par exemple, vous permettez la création de mondes illimités. Ceux-ci évoluent alors et se créent dynamiquement en fonction des déplacements des personnages dans le monde infini. On peut envisager qu'une entité calcule localement une partie de ce monde tout en garantissant sa cohérence via des constantes (graines aléatoires) définies à la création de la partie. Il peut s'agir de graines associées aux positions x et y permettant de calculer les obstacles et les donjons. Dans le cas d'un monde infini, le calcul local du monde est à privilégier puisque transmettre les résultats des calculs aux autres entités serait à l'inverse complexe et coûteux.
- Gestion des monstres et des objets. Vous pouvez par exemple proposer que chaque participant soit propriétaire d'un certain nombre de monstres et d'objets. Cela permettrait à chaque entité d'application de créer dynamiquement de nouveaux monstres et objets afin de rendre attractif le jeu. Cette propriété doit être dans une certaine mesure cessible.
- Intelligence artificielle répartie. Les monstres et les objets peuvent présenter une intelligence artificielle. Par exemple, un monstre repère les personnages dans son champ d'interaction et tente de poursuivre et d'attaquer les plus faibles. Le pilotage du monstre/objet peut éventuellement passer d'une entité à l'autre en fonction de la zone dans laquelle se trouve le monstre/objet.

Vous devez non seulement justifier les avantages proposés par vos fonctionnalités et interfaces mais aussi leurs inconvénients. Autrement dit vous **justifier le respect de la cohérence** et **donner les limites** notamment en termes **de jouabilité**.

4. Travail du point de vue réseau

Vous définissez les **protocoles**, notamment pour la cohérence, qui supportent vos fonctionnalités. Vous les **concevez et justifiez** et en donnant les limites.

Pour chaque protocole, vous **expliquez la mise en œuvre**. Donc vous devez justifier que votre code réalise de façon satisfaisante ce protocole. En effet, on peut tout à fait proposer un bon protocole mais dont la réalisation est défaillante. Par exemple, vous ne calculez pas les limites des messages lorsque vous transmettez ces messages via TCP donc votre réalisation ne fonctionne pas c'est-à-dire marche de façon tout à fait erratique.

5. Travail du point de vue système

Vous allez devoir définir l'architecture système, c'est-à-dire les processus et activités ainsi que les moyens de communication et de synchronisation.

Vous devez justifier votre architecture système vis-à-vis du traitement de la concurrence. En effet, votre protocole peut tenter de garantir une cohérence mais votre architecture système peut être incapable de garantir la cohérence des données et des opérations sur ces données.

Il vous est aussi demandé de justifier l'efficacité de votre architecture. Ainsi, une architecture utilisant des activités peut être rendue inefficace en raison des mécanismes de synchronisation choisis (par exemple, chaque activité s'exécute en exclusion mutuelle ce qui interdit tout parallélisme ou gain de performance).

Vous justifiez que la mise en œuvre est correcte. Vous montrez notamment que vous traitez bien les cas d'erreurs et les exceptions système. En autre terme, vous devez justifier le comportement de votre code et ses limites.

6. Passage à l'échelle

Votre jeu doit pouvoir passer le plus possible à l'échelle. Cet aspect est secondaire mais il offre des bonus supplémentaires en termes d'évaluation puisque le jeu sera plus ou moins largement et efficacement multi-joueurs.

Le passage à l'échelle se caractérise par différents aspects :

a) Le nombre de joueurs simultanés qu'autorise votre jeu. Celui-ci doit supporter au moins deux joueurs dans une partie. Le nombre de joueurs peut être limité pour des raisons d'interface utilisateur, d'architecture système ou de mécanisme réseau.
b) Vous devrez donner une mesure des performances de votre jeu qui se base sur des expérimentations réelles. Ainsi, il n'est pas utile d'annoncer un nombre illimité de joueurs si vous n'avez testé votre jeu qu'entre quelques joueurs.
c) Vos mesures de performances peuvent notamment porter sur

- la capacité de l'interface utilisateur à supporter un grand nombre de joueurs
- l'architecture système qui autorise une dégradation lente des performances en fonction du nombre de joueurs
- les protocoles réseau que vous avez définis pour minimiser la surcharge réseau et machine
- la fluidité du jeu en fonction du nombre de joueurs et du volume d'interaction.

Sur tous ces points, il est inutile de dire ma solution est efficace si vous n'avez pas fait de réelles quantifications portant sur différentes expérimentations et volumétries. Vous devez donc être capable de définir les limites de vos expériences et les points non mesurés.

7. Objectifs de sécurité

Votre jeu doit éventuellement proposer des propriétés et des mécanismes de sécurité.
Cet aspect est aussi secondaire.
Les objectifs de sécurité peuvent porter notamment sur :

a) l'identification des participants. Ainsi, si vous utilisez des identifiants pour chaque participant, ceux-ci doivent être le plus possible confidentiels et intègres. La confidentialité peut par exemple reposer sur une connexion de type SSL qui chiffre les données transmises entre les différents joueurs. L'intégrité peut consister en un mécanisme de signature de l'identifiant afin d'éviter que quelqu'un ne forge un identifiant à la place d'un autre joueur.

b) la difficulté de rejouer un identifiant pour usurper une identité. Pour cela, vous pouvez par exemple ajouter une durée de vie à l'identifiant afin que celui-ci soit recalculé régulièrement. Couplé à un mécanisme de signature on peut alors limiter les attaques par rejeu.
c) la difficulté de forger un message malveillant. Par exemple, vous utilisez une cryptographie avec des clés asymétriques afin d'ajouter une somme de contrôle signée dans les messages. Ainsi, le récepteur peut vérifier l'intégrité et l'authenticité du message.
d) la difficulté de conduire des attaques système ou d'injecter des données dans le jeu. Une attaque système peut notamment correspondre à faire exécuter à l'interface Python un script malveillant qui écrit dans la mémoire partagée lue par le composant réseau.

Dans tous les cas, vous devrez tenter d'identifier les limites de votre solution en termes de sécurité. Ainsi, vous pourrez décrire les failles résiduelles ou les attaques non couvertes voir dérouler des preuves de concept pour ces attaques.

8. Organisation de l'équipe

Vous allez structurer une équipe d'étudiants avec un maximum de 3 étudiants.
Un étudiant sera choisi comme coordinateur pour l'équipe afin de rendre compte du travail au sein de l'équipe, mettre en place des processus de décision et de coopération entre les membres de l'équipe. Le coordinateur sera chargé des échanges avec l'enseignant qui encadre le projet.

Vous êtes entièrement libre de vous organiser au sein de l'équipe. Cependant, vous devez, avant de vous engager dans une solution ou une organisation, l'exposer précisément à l'encadrant. La présence aux séances de suivi est obligatoire pour toute l'équipe et doit contribuer à la coordination et aux échanges.
Une auto-évaluation vous est demandée. Vous devez le plus possible prendre en charge un problème ou une fonctionnalité du début à la fin (c'est-à-dire de sa définition, conception de la solution, mise en œuvre, test et intégration, justification). Si le travail à plusieurs, sur un même problème ou fonctionnalité, est toléré vous devez par ailleurs avoir proposé une contribution individuelle. Vous devez décrire et justifier vos contributions. Vous vous donnez une note et une position dans le groupe et vous les justifiez.
Une évaluation par les pairs est demandée. Pour cela les différents membres devront donner un avis sur le coordinateur. Celui-ci attribuera une note à chacun des membres. Il justifiera les notes données au regard des contributions de chacun des membres.

9. Contraintes de développement

Vous développez la mise en réseau en C avec les mécanismes système adaptés (API socket, processus, thread, IPC locaux, mécanismes de synchronisation, …).

De préférence, nous vous conseillons de produire la partie réseau dans un ou plusieurs processus indépendants de la partie Python. Il vous faudra cependant utiliser des mécanismes de communication et de synchronisation entre les processus Python et C. Cela vous demande donc de faire de la programmation système à la fois en Python et en C. Par contre, la partie réseau doit être développée entièrement en C au moyen de l'interface socket.

Nous vous laissons libre de développer sous Unix ou sous Windows. Le fait de fournir une solution sous Windows vous donnera une plus-value dans la mesure où c'est un OS sur lequel

vous n'avez pas de cours de programmation système ni réseau. Un OS Windows actuel est très proche d'un Unix et la programmation en C sur ces deux plateformes présente de nombreuses similarités. Cependant, il existe des spécificités à Windows que vous devrez étudier afin de maîtriser la programmation système sur cet OS. C'est pourquoi nous évaluerons positivement votre capacité à proposer une solution sous Windows.

Bien entendu vous pouvez proposer une solution multi-plateformes supportant à la fois Unix/Linux et Windows.

10. Travail attendu

Votre travail doit permettre de proposer très vite une première version du jeu que vous présentez durant les séances de suivi. Ainsi, plus vous fournissez rapidement une première version qui permet à deux joueurs d'interagir concurremment sur un objet partagé en garantissant sa cohérence. La première version doit supporter une partie avec deux joueurs.
Sur la base de cette première version vous proposerez différentes améliorations pour les fonctionnalités, l'interface, le réseau, l'architecture système, le passage à l'échelle et la sécurité. Pour chacune de ces améliorations vous présenterez les limites.
La livraison finale consiste en 4 diapositives qui résument la solution, les avantages, les inconvénients ainsi que les évaluations et mesures faites. La soutenance repose essentiellement en une démonstration qui illustre le fonctionnement, les avantages et inconvénients. Votre travail sera évalué sur la base 1) de votre capacité à présenter et justifier votre solution durant les séances, 2) du rapport d'auto-évaluation/évaluation des pairs et 3) de la soutenance finale.
Une attention particulière sera portée à votre autonomie, votre capacité à communiquer et justifier vos contributions.