SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Devoir de Programmation PC2R 2019
Arènes Vectorielles Synchrones

Version du 01/04: legeres corrections du protocole et précisions sur la compatibilité.

Description du devoir

But du devoir

Le but du devoir est de réaliser une application clients-serveur permettant à des utilisateurs de jouer à un jeu de déplacement vectoriel en arène : plusieurs joueurs dirigent **simultanément** des véhicules dans une arène **torique** en contrôlant indirectement leur **vitesse**. Les joueurs s'affrontent dans des sessions en cherchant à être le premier à effectuer une tâche simple (passer à un endroit particulier de l'arène, appelé **objectif**).

Exigences techniques

L'application doit être réalisée suivant une architecture client / serveur.
La partie client et la partie serveur doivent être chacune écrite dans un langage différent, les deux langages appartenants à l'ensemble {`C`, `Java`, `OCaml`}.
L'utilisation d'autres langages est possible mais soumise à autorisation de la part de l'équipe pédagogique.
Le client et le serveur doivent pouvoir fonctionner indépendamment l'un de l'autre (et pouvoir fonctionner avec le client et le serveur des autres étudiants). Ils doivent respecter le(s) protocole(s) décrit(s) plus loin.

Attention. Les règles du jeu sont découpées en trois parties (A, B, C) formant une séquence ordonnée pour l'inclusion, l'objectif est de produire un projet qui correspond à la partie la plus compliquée (la partie C), mais il est conseillé de compléter entièrement chaque partie dans l'ordre.

Partie A: Jeu simple

Le jeu se déroule en sessions (parties) successives. A la connexion d'un premier joueur, le serveur lance un compte-à-rebours interne (par exemple 20s). A la fin de ce compte-à-rebours, il commence une session avec tous les joueurs connectés. A la fin d'une session, le serveur attend un délai fixé à l'avance (par exemple 20s) puis relance une nouvelle session avec tous les joueurs connectés. Ainsi, un joueur reste connecté jusqu'à ce qu'il décide de quitter explicitement le jeu.

Au moins 1 joueur participe à une session de jeu. Il n'y a, a priori, pas de nombre maximum de joueurs. Un joueur peut rejoindre ou quitter une session à tout moment.

Arène et Véhicules

Arène. Le jeu se déroule au sein d'une arène torique à coordonnées décimales. C'est à dire que l'arène est définie par un centre (de coordonnées $0, 0$), une demi-hauteur de valeur h et une demi-largeur de valeur l.

L'arène est torique ce qui veut dire que, pour tout x, le point x, h est identifié au point $x, -h$ et que, pour tout y, le point l, y est identifié au point $-l, y$. Intuitivement on peut imaginer l'arène comme une carte vue du dessus où quand on sort de la carte par l'extrême-est (resp. ouest) on se retrouve à l'extrême-ouest (resp. ouest) et quand on sort de la carte par l'extrême-nord (resp. sud) on se retrouve à l'extrême-sud (resp. nord).

L'arène est décimale, ce qui veut dire que les coordonnées sont (virtuellement) **continues**, un objet se trouvant en (x, y) est théoriquement discernable d'un objet se trouvant en $(x + 0.0001, y)$

Véhicule. Les joueurs sont représentés au sein du jeu par un **véhicule**, qui, consiste en un unique point. Ainsi un véhicule est défini par trois caractéristiques: (i) sa position (x, y) (coordonnées dans l'arène), (ii) sa direction θ (angle par rapport à l'axe $[O, x)$, et (iii) son vecteur vitesse (vx, vy) (il est important de réaliser que la direction du véhicule peut différer de la "direction" de son vecteur vitesse).

Déplacement. Le déplacement du véhicule est géré ainsi: un paramètre `refresh_tickrate` du client donne la fréquence de rafraichissement; toutes les $\frac{1}{\texttt{refresh\_tickrate}}$ secondes le client calcule la position du vaisseau en modifiant sa position en accord avec son vecteur vitesse (sa position devient $x + vx, y + vy$).

Contrôle. Le contrôle du véhicule est géré par 3 commandes: `clock` permet de tourner la direction du véhicule dans le sens horaire, `anticlock` dans le sens antihoraire, et `thrust` applique une impulsion au véhicule dans sa direction actuelle.

Le client dispose de constantes `turnit` et `thrustit` (à régler pour que le jeu soit "jouable") telles que la commande `clock` (resp. `anticlock`) change l'angle θ en $\theta - \texttt{turnit}$ (resp. $\theta + \texttt{turnit}$), la commande `thrust` change la vitesse vx, vy en $vx + \texttt{turnit} * cos(\theta), vy + \texttt{turnit} * sin(\theta)$

Début de session Au début d'une session, les véhicules sont placés aléatoirement dans l'arène et leur vitesse est nulle $(0, 0)$.

Un **objectif** est choisi par le serveur à des coordonnées aléatoires. Le but d'un joueur est d'être le premier à faire passer son véhicule à une distance inférieure à `obj_radius` de l'objectif.

Déroulement. Quand un véhicule passe à proximité de l'objectif, celui-ci disparait, le contrôleur du véhicule gagne un point, et un nouvel objectif est généré.

Fin de session. Quand un joueur totalise `win_cap` points, il est déclaré vainqueur et la session s'arrête.

Synchronisation. Le paramètre `server_tickrate` indique la fréquence de rafraichissement réseau (on supposera qu'il est bien inférieur à `refresh_tickrate`). Toutes les $\frac{1}{\texttt{server\_tickrate}}$ secondes, le serveur envoie à tous les clients la position de tous les véhicules, ensuite, chaque client envoie la position de son véhicule au serveur.

Protocole (Partie A)

Clients et Serveur échangent selon un protocole texte. Une commande est composée de chaînes ASCII terminées par des `/`. La commande elle-même est terminée par un `\n`. En cas de doute sur la non-ambiguité ou la complètude du protocole, écrire à l'équipe pédagogique.

Connexion

```
CONNECT/user/
(C -> S) Nouvelle connexion d'un client nomme 'user'
WELCOME/phase/scores/coord/
(S -> C) Validation de la connexion, indication de la phase (attente ou jeu),
des joueurs connectés et des scores et des coordonnées de l'objectif éventuel
DENIED/
(S -> C) Refus de la connexion (par exemple parce qu'un client avec le meme nom est deja connecte).
NEWPLAYER/user/
(S -> C) Signalement de la connexion de 'user' aux autres clients.
```

Déconnexion

```
EXIT/user/
(C -> S) Déconnexion de 'user'.
PLAYERLEFT/user/
(S -> C) Signalement de la deconnexion de 'user' aux autres clients.
```

Début d'une session

```
SESSION/coords/coord/
(S -> C) Début d'une nouvelle session, coordonnées de départ des véhicules, coordonnées de l'objectif.
WINNER/scores/
(S -> C) Fin de la session courante, scores finaux de la session.
```

Jeu

```
NEWPOS/coord/
(C -> S) Envoi des nouvelles coordonnées au serveur.
TICK/coords/
(S -> C) Propagation des nouvelles coordonnées.
NEWOBJ/coord/scores
(S -> C) Nouvel objectif, rappel des scores.
```

Figure 1: Protocole. Partie A.

Chaînes

- `user` est une chaîne de lettres identifiant un joueur, en lettres romaines minuscules, par exemple `riri`.
- `phase` est une chaîne de lettres identifiant la phase: soit `attente` soit `jeu`.
- `coord` est une chaîne représentant des coordonnées dans l'arène, l'abscisse est précédée de `X` et l'ordonnée de `Y`, par exemple `X0.000125Y-2.5`.
- `coords` est une chaîne représentant les coordonnées de tous les véhicules dans l'arène, il s'agit de plusieurs chaines `coord` précédé par le nom du joueur et un `:`, séparées par des `|`, par exemple `riri:X0.000125Y-2.5|fifi:X0Y0|loulou:X-0.03333333Y0.27`.
- `scores` est une chaîne donnée indiquant les scores des participants, précédés par le nom et `:` et séparés par des `|`. Par exemple: `riri:5|fifi:0|loulou:1`

Partie B: Calcul côté serveur

La triche est tres facile dans une session de jeu simple: un client peut être modifié pour envoyer directement au serveur la position de l'objectif. Il peut sembler judicieux de faire calculer les déplacements par le serveur.

Dans la partie B, les clients communiquent, tous les `server_tickrate`, les commandes effectuées et recoivent les positions et les vecteurs vitesse de tous les véhicules.

Le serveur met à jour, localement, la position de tous les clients selon un `server_refresh_tickrate` (supérieur au `server_tickrate`) selon les dernière commandes reçues.

Les clients mettent à jour, localement, à chaque `refresh_tickrate` la position de tous les véhicules (y compris du leur) en se basant sur les dernières information reçues: cela a pour effet de créer du *lag* (le véhicule ne répond pas directement aux commandes, et semble se "téléporter"), pour des valeurs de `server_tickrate` faibles.

Si plusieurs commandes ont lieu durant un *tick*, le client les compile (additionnant les angles d'une rotation et le nombre de poussées).

Jeu

```
NEWCOM/comms/
(C -> S) Envoi des commandes au serveur.
TICK/vcoords/
(S -> C) Propagation des nouvelles coordonnées, vitesses et directions.
NEWOBJ/coord/scores
(S -> C) Nouvel objectif, rappel des scores.
```

Figure 2: Modification du protocole. Partie B.

Chaînes

- `comms` est une chaîne résumant les commandes effectuées sur le client depuis le dernier *tick*: elle contient une valeur d'angle (en radian) et une valeur de poussée (un entier) précédé par (respectivement) `A` et `T`, par exemple `A-1.57T2` (rotation de $-\frac{\pi}{2}$, deux commandes de poussée).
- `vcoords` est une chaîne représentant les coordonnées et les vecteurs vitesse de tous les véhicules dans l'arène, il s'agit de chaines `coord` précédées par le nom du joueur et un `:` et suivies par une chaine similaire pour la vitesse utilisant `VX` et `VY` à la place de `X` et `Y` et une chaîne pour l'angle prédécée de `T`, séparées par des `|`, par exemple `riri:X0.000125Y-2.5VX-0.002VY0T1.57|fifi:X0Y0VX0VY0T0|loulou:X-0.03333333Y0.27VX0.01VY-0.01T-3.14`.

Partie C: Collision

On désire maintenant ajouter des obstacles (initialement des points de l'arène) qui repoussent les véhicules et gérer les collisions entre véhicules.

Les véhicules et les obstacles ont un rayon de collision (`ve_radius` et `ob_radius`). Si les coordonnées d'un véhicule se trouvent à l'intérieur du cercle de rayon `ve_radius` (resp. `ob_radius`) d'un véhicule (resp. d'un obstacle), il est repoussé: dans un premier temps, il on supposera que sa vitesse est inversée (elle devient $-vx, -vy$).

La prédiction du client doit prendre en compte les collisions: c'est-à-dire que si une collision a lieu entre deux mises à jour des positions+vitesse par le serveur, le client doit modifier le comportement du véhicule selon les règles de collision.

Connexion

```
WELCOME/phase/scores/coord/ocoords
(S -> C) Validation de la connexion, indication de la phase (attente ou jeu),
des joueurs connectés et des scores, coordonnées de l'objectif éventuel et
des obstacles éventuels.
```

Début d'une session

```
SESSION/coords/coord/ocoords/
(S -> C) Début d'une nouvelle session, coordonnées de départ des véhicules,
coordonnées de l'objectif, coordonnées des obstacles.
```

Figure 3: Modification du protocole. Partie C.

- `ocoords` est une chaîne représentant les coordonnées de tous les obstacles l'arène, il s'agit de plusieurs chaines `coord`, séparées par des `|`, par exemple `X0.000125Y-2.5|X0Y0|X-0.03333333Y0.27`.

Extensions

Il est probable que la réalisation d'extension(s) implique un enrichissement du protocole. Il est important que, au maximum, clients et serveurs bénéficiant d'extensions soient compatibles avec ceux n'en bénéficiant pas.

Par exemple:

- Votre client et votre serveur doivent pouvoir ignorer les commandes inconnues du protocole.
- Vous pouvez placer des informations supplémentaires à la fin des commandes, par exemple `CONNECT/user/color` à la place de `CONNECT/user` et le client et le serveur ignorent la fin des commandes qu'ils ne savent pas interprêter.

Pour obtenir la note maximale, le projet doit implanter des extensions supplémentaire parmi les suivantes (ou d'autres extensions non prévues dans ce sujet) (l'implémentation d'extension ambitieuse sera plus valorisée qu'un simple ajout d'un chat):

- **Jeu de Combat**: les véhicules sont capables d'actions aggressives envers les autres véhicules: tir de projectiles ou de "laser" dans la direction à laquelle ils font face, ou pose de "bombes" sur le sol. Les véhicules touchés sont stoppés (à vitesse nulle) ou incapacités (stoppés pendant un certain temps).

- **Jeu de Course**: les véhicules doivent être les premiers à passer à travers plusieurs objectifs, dans un certain ordre. Les objectifs ne disparaissent pas quand ils sont atteints, le premier joueur a avoir atteint tous les objectifs gagne la course.

- **Calcul côté client et vérification**: Le calcul de position est effectué aussi côté client et la nouvelle position client est envoyée en même temps que les commandes. Apres avoir calculé la nouvelle position serveur à partir des commandes reçues, le serveur la compare à la position client reçue: si elle n'est "pas trop loin", il donne la priorité à la position client.

  Cela permet une meilleure granularité des déplacements: par exemple, "tourner de 45 degrés, effectuer une poussée, tourner de -45 degrés, effectuer une poussée", est compilé en "tourner de 0 degrés, effectuer 2 poussées" par les règles de la partie B, même si cela donne un résultat différent.

- **Sûreté des collisions**:

- **Choc élastiques**: les collisions entre véhicules et obstacles sont résolues par la physique chocs élastiques à deux dimensions (cf. Wikipedia, ou un cours de Mécanique).

- **Chat:** Le client et les serveurs gerent un minisalle de clavardage qui permet aux clients de communiquer entre eux à tout moment (quelque soit la phase de jeu)

```
ENVOI/message/
(C -> S) Envoi (public) d'une chaine de caractere "message" à tous les joueurs.
PENVOI/user/message/
(C -> S) Envoi (privé) d'une chaine de caractere "message" au joueur "user" uniquement.
RECEPTION/message/
(S -> C) Réception d'un message public.
PRECEPTION/message/user/
(S -> C) Réception d'un message prive de l'utilisateur "user".
```

Points d'intérêt

L'évaluation d'un projet s'intéressera (entre autres) aux points suivants:

- **Qualité de la gestion de la concurrence**: le serveur doit gérer plusieurs threads clients qui accedent de manière **sûre** et **efficace** à des ressources partagées.

- **Qualité de la gestion de la synchronie**: les mécanismes mis en place pour assurer une expérience "simultanée" (par exemple, la prédiction) doivent être efficaces.

  - Le jeu doit pouvoir fonctionner avec un `server_tickrate` faible.

- **Respect du protocole** et compatibilité avec les clients/serveurs des autres groupes.

- **Qualité du Rapport**: description et justifcation des choix d'implémentation et manuel du jeu.

- **Jouabilité**: ergonomie de l'interface client, et qualité des entrées/sorties (paramètres, journal) du serveur.

  - Il est conseillé d'utiliser une représentation graphique des véhicules permettant d'identifier facilement leur direction (par un exemple un triangle isocèle non-équilatéral, ou une flèche, ou un sprite).