# C-Tron
# (Cloud Gaming Edition)

---

Ce projet peut être faire seul ou en binôme. Il est à rendre sur Moodle le **11 Décembre 2022** au plus tard.

---

## Avant de commencer

Ce projet doit être rendu avec un rapport au format pdf. N'oubliez pas d'indiquer vos noms dans le rapport. La qualité de ce dernier sera prise en compte dans la notation.

Votre code doit être commenté, indenté, et doit compiler (n'oubliez pas de fournir un makefile). Le code doit être fait dans le langage C. Veillez à suivre les consignes concernant les structures de données et le prototype de chaque fonction à implémenter.

Le rapport (format pdf) servira à commenter/expliquer vos choix d'implémentation lorsque nécessaire. Ce dernier contiendra aussi les réponses aux questions posées (Section **??**). Attention : un rapport n'est pas un `README`, et inversement.

Un non-respect de ces consignes entraînera un retrait de point. Le plagiat sera sanctionné. Si du code a été obtenu via Internet ou via un outil de génération, sa source doit être clairement indiquée en commentaire dans votre code. La présence de code copié sans source attachée sera sanctionnée.

Une archive au format .zip dont le nom contiendra les noms des membres du binôme sera à déposer sur Moodle. Elle doit contenir votre rapport (en pdf), votre code et un makefile.

Le fichier `client_template.c` disponible sur Moodle permet de faciliter votre utilisation des librairies sur lesquelles vous devrez vous reposer. Des informations supplémentaires sont disponibles en annexe de ce sujet.

Un fichier `common.h` est également disponible et fourni les structures de données qui *doivent* être utilisées.

## 1 Description rapide

(a) Light Cycles dans Tron

(b) Light Cycles avec `ncurses`

Figure 1 – Light cycles

Light cycles est un jeu que l'on peut notamment apercevoir dans le film Tron, sorti (originellement) en 1982. Dans ce jeu, illustré sur la figure **??**, des joueurs contrôlent des *light cycles*, sorte de motos laissant un mur de lumière solide dans leur sillage. Le but du jeu est de faire en sorte que l'adversaire entre en collision avec l'un des murs ou les bordures du jeu.

| Touche clavier (1j/2j) | Effet |
|---|---|
| z/i | Oriente le light cycle vers le haut |
| q/j | Oriente le light cycle vers la gauche |
| s/k | Oriente le light cycle vers le bas |
| d/l | Oriente le light cycle vers la droite |
| espace/m | Permet de choisir si le light cycle doit laisser un mur de lumière ou non |

FIGURE 2 – Commandes disponibles pour les joueurs.

Le but de ce projet est de re-créer ce jeu dans le terminal, à l'aide de la libraire `ncurses` [**?**] (voir Fig. **??**). Une partie opposera deux joueurs, qui communiqueront via des sockets par l'intermédiaire d'un serveur qui fait tourner le jeu. Une aide d'utilisation à `ncurses` est disponible en annexe, et un squelette de programme montrant sont utilisation est disponible avec le sujet (`client_template.c`)

L'architecture du jeu suivra un modèle s'approchant d'une forme de cloud gaming : le serveur est en charge des calculs (des positions des joueurs, des collisions, etc), et les clients ne font qu'afficher le jeu à l'écran.

# 2 Implémentation

## 2.1 Gameplay

Les *light cycles* des joueurs ne peuvent pas être arrêtés. De manière similaire au jeu SNAKE, ces-derniers sont tout le temps en train d'avancer, et un joueur peut uniquement changer la direction de son light cycle. Par défaut, un light cycle laisse un mur de lumière sur la case qu'il vient de quitter, mais le joueur peut désactiver cette aptitude lorsqu'il le désire.

Nous autoriserons 2 joueurs au maximum : soit les 2 joueurs sont sur le même ordinateur, soit chaque joueur sera sur un ordinateur différent.

Chaque joueur peut contrôler son light cycle via les touches du clavier. Si les deux joueurs sont sur le même clavier, ils utiliseront des touches différentes. Les contrôles disponibles sont décrits dans la figure **??**.

La partie peut prendre fin de plusieurs manières différentes :

— Un joueur entre en collision avec un mur de lumière. Ce joueur perd.
— Un joueur entre en collision avec les murs du jeu. Ce joueur perd.
— Les Light cycles des deux joueurs entrent en collision. Les deux joueurs ont perdu.

Si les deux joueurs perdent, la partie termine sur une égalité. Sinon, le joueur qui n'a pas perdu est déclaré gagnant.

## 2.2 Architecture client-serveur

Les joueurs désirant jouer doivent établir une connexion avec un serveur. **Les connexions clients-serveur reposent sur le protocole TCP.** Une fois que le serveur détecte que deux joueurs sont présents, la partie commencent.

**L'architecture utilisée est similaire à celle utilisée pour du cloud gaming** (bien qu'extrêmement simplifiée) **:** le serveur est en charge de tous les calculs, et envoie uniquement des informations d'affichage aux clients qui affichent chaque frame du jeu à l'écran à chaque réception. Cette architecture est illustrée sur la Fig. **??**.

### 2.2.1 Programme Client

Une fois lancés, un client tente de se connecter au serveur de jeu, dont l'adresse et le port sont passés en argument. Une fois connecté, le client doit envoyer le nombre de joueurs actuellement en train de jouer sur ce client. Ensuite, le client surveille l'arrivée d'inputs du serveur ou de l'entrée standard afin de les traiter.

Lors d'une partie, les clients, tournant sur les ordinateurs des joueurs, doivent surveiller l'entrée standard (via laquelle les joueurs contrôlent leur personnage). **Dès qu'une nouvelle commande est perçue, cette dernière est envoyée au serveur. Les clients ne calculent pas les positions des joueurs. Ce calcul est effectué par le serveur**.

Le programme client doit être en mesure de supporter jusqu'à **deux joueurs sur une même machine**. Le premier joueur bougera avec les commandes *z,q,s,d,space*, tandis que le second (si un second joueur utilise le même clavier) utilisera *i,j,k,l,m*.

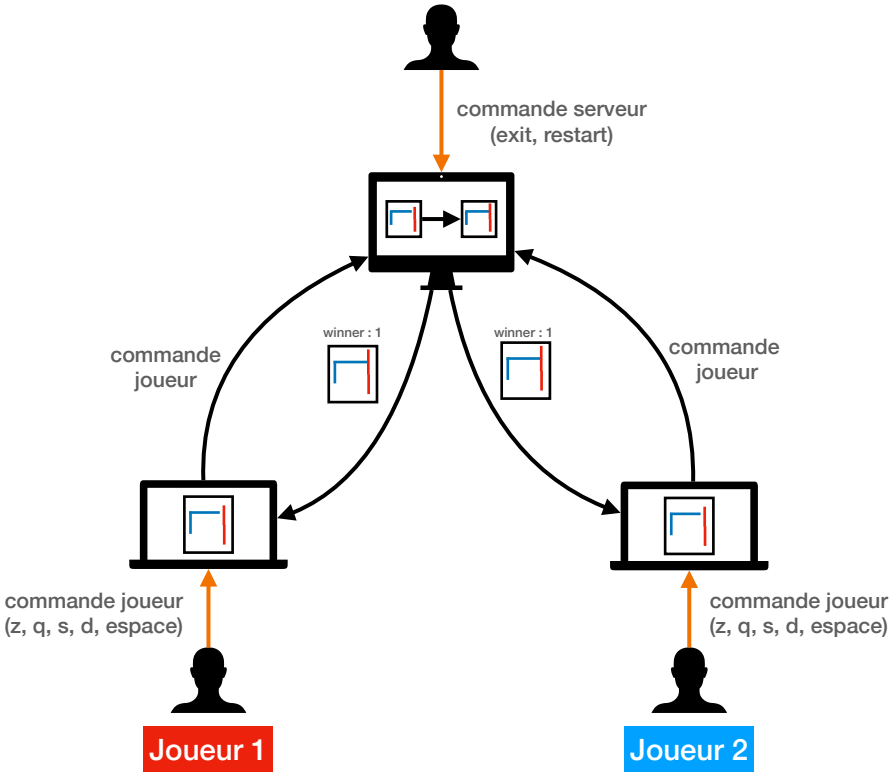



FIGURE 3 – Architecture client-serveur utilisée pour C-Tron.

**En plus de l'entrée standard, le programme client doit surveiller son canal ouvert avec le serveur**, qui lui envoie périodiquement des informations concernant la nouvelle position des joueurs, ainsi que des informations concernant l'éventuel gagnant.

Le programme client doit se lancer via la commande suivante :

```
./client [IP_serveur] [port_serveur] [nb_joueurs]
```

Où `nb_joueurs` est le nombre de joueurs jouant sur ce client (deux maximum).

#### 2.2.2 Programme Serveur

Une fois lancé, le serveur attend que deux joueurs soient connectés (un client avec deux joueurs, ou deux joueurs séparés). Le serveur écoute sur ses IP locales, au port renseigné en argument. Dès qu'un client se connecte, il attend de recevoir (de ce client spécifique) combien de joueurs se trouvent derrière le client. Dès que deux *joueurs* sont présents, la partie commence.

**Le serveur maintient l'état actuel du jeu et est chargé de calculer et mettre à jour les nouvelles positions des joueurs** en fonction de la direction des light cycles (qui doit être modifiée si une commande émise par le joueur a été reçue). **C'est également lui qui met à jour la position des murs de lumière, et calcule si une collision a eu lieu.** Ces mises à jour doivent être faites périodiquement : environ toutes les 100ms par défaut, mais paramétrable par au moment de l'exécution. Plus précisément, le serveur doit se lancer via la commande :

```
./serveur [port_serveur] [refresh_rate]
```

Où `refresh_rate` est la périodicité entre chaque nouvelle frame du jeu. Attention : bien que le serveur calcule / met à jour les nouvelles positions et envoie la nouvelle frame seulement toutes les `refresh_rate` ms, il lit et traite les messages des joueurs dès leur réception.

Une fois les positions mises à jour et les collisions éventuelles calculées, **le serveur envoient ces informations aux clients, qui se contentent de les afficher** à l'écran.

**Le serveur accepte également certaines commandes via l'entrée standard.** En particulier, entrer `restart` permet de redémarrer la partie, et `quit` permet de quitter le jeu.

### 2.3 Détails d'implémentation à respecter

Idéalement, votre programme client devrait être en mesure de communiquer avec *notre* programme serveur, et *notre* programme client devrait être en mesure de communiquer avec votre serveur [1].

Comme toute communication sur l'Internet, afin d'assurer une telle forme de cross-compatibilité, la manière dont les données sont envoyées (leur format) doit être standardisé.

Un fichier `common.h` vous est fourni, et contient les structures de données qui devront **impérativement** être utilisé pour la communication entre le client et le serveur.

La structure `client_init` contient les informations nécessaires pour indiquer le nombre de joueurs derrière un même client. Elle doit être envoyée dès que la tentative de connexion d'un client a été acceptée.

La structure `client_input` contient les informations envoyées par le client lorsqu'une commande est passée. Plus précisément, la pression d'une touche de clavier (z,q,s,d,space,i,j,k,l,m) se traduit par l'envoi d'une telle structure au serveur, lui indiquant qu'un joueur (identifié par un id) a changé de direction ou a activé/désactivé son mur de lumière.

Pour finir, la structure `display_info` est envoyée par le serveur, et contient toutes les informations actuelles du plateau de jeu. Plus précisément, cette structure contient un champ `winner`, qui est à -1 si personne n'a gagné, et est changé à l'identifiant du vainqueur lorsqu'un des joueurs gagne. Il contient également un champ `board`, de taille `XMAX` et `YMAX`, qui contient tout le plateau du jeu.

La structure `board` est un simple tableau de `char`, où la valeur d'une case représente un élement se trouvant à cette position dans le jeu. Les valeurs sont également standardisées. Une valeur de 111 signifie la présence d'un mur du jeu. Les valeurs de 0,1,2,3,4 sont réservées à des identifiants de joueurs et signifient la présence d'un joueur sur cette case. Les valeurs de 50,51,52,53,54 sont réservées pour des murs de lumière laissés par des joueurs. Plus précisément, un mur de lumière laissé par le joueur $x$ sera symbolisé par $x + 50$.

Les couleurs déclarées dans `client_template.c` sont rattachés aux valeurs en question. Autrement dit, la valeur $x$ d'une case doit être affichée après activation de la couleur numéro $x$ (davantage d'informations sont disponibles en annexe du sujet).

La Fig. **??** montre un example de `board` tel que défini par le serveur et tel qu'affiché par le client après réception. Le client lit chaque case du tableau, et affiche un caractère (au choix) selon la couleur rattachée à cette valeur. Par exemple, la couleur 50 est la couleur "cyan

1. Nous mettrons à disposition un programme client & serveur codé l'équipe pédagogique prochainement

sur cyan", permettant de colorer une case bleue, et la couleur 51 est la couleur "jaune sur jaune", permettant de colorer une case en jaune. Notez que ces couleurs sont déjà déclarées (et rattachées aux valeurs correspondantes) dans le fichier `client_template.c`.

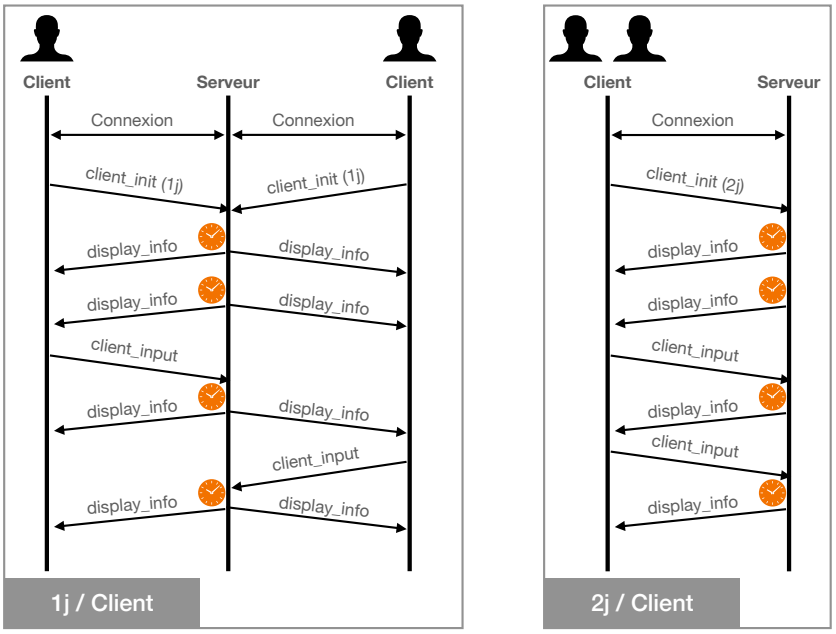



FIGURE 4 – Echange de messages client-serveur (avec structures de données).

Enfin, la Fig. **??** montre un exemple d'échange de message entre client(s) et serveur, avec deux clients (à gauche) et un seul client (à droite) en exhibant les structures de données à utiliser.

## 3 Questions

En plus du code C fourni, vous répondrez également aux questions suivantes dans votre rapport.

1. Discutez des avantages et inconvénients de ce genre d'architecture. En particulier, examinez l'impact des caractéristiques physiques de la connexion (délai, gigue, taux de pertes...).
2. L'utilisation du protocole TCP est-elle souhaitable pour les jeux en ligne ? Qu'en est-il du Cloud Gaming ? Idéalement, quel(s) protocoles utiliseriez-vous dans le cas présent ?

## 4 Modalité d'évaluation

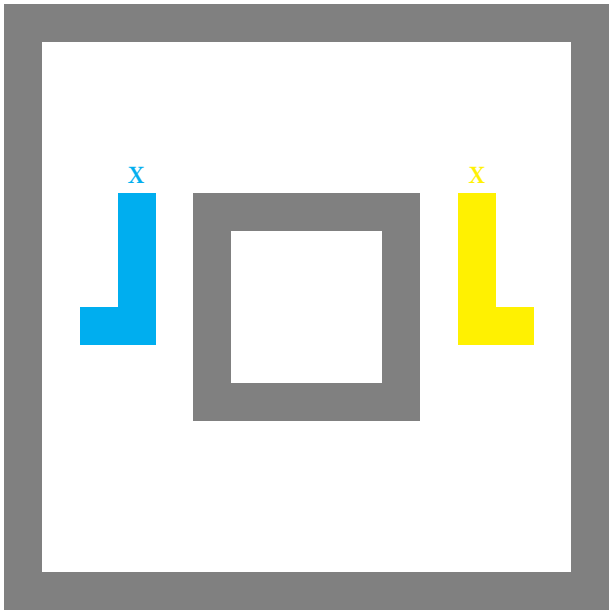
Afin de vérifier si votre client et serveur respectent les consignes de l'énoncé, ces-derniers seront testés à l'aide de nos propres programmes clients et serveurs. Plus précisément, votre client devra idéalement être en mesure d'initier une partie avec notre serveur, et inversement.

Afin de tester vos programmes au fur et à mesure de leur conception, nous mettrons à votre disposition ces deux exécutables `client` et `serveur` sur la page Moodle de l'UE. Pensez à régulièrement vérifier que vos codes respectent les pré-requis demandés à l'aide de ces exécutables.

Votre note dépendra non seulement de votre code (indenté et commenté), mais aussi de votre rapport, qui justifiera vos choix d'implémentation et contiendra les réponses aux questions posées.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 111 | 111 | 111 | 111 | 111 | 111 | 111 | 111 | 111 | 111 | 111 | 111 | 111 | 111 | 111 | 111 |
| 111 | | | | | | | | | | | | | | | 111 |
| 111 | | | | | | | | | | | | | | | 111 |
| 111 | | | | | | | | | | | | | | | 111 |
| 111 | | | 0 | | | | | | | | | 1 | | | 111 |
| 111 | | | 50 | | 111 | 111 | 111 | 111 | 111 | 111 | | 51 | | | 111 |
| 111 | | | 50 | | 111 | | | | | 111 | | 51 | | | 111 |
| 111 | | | 50 | | 111 | | | | | 111 | | 51 | | | 111 |
| 111 | | 50 | 50 | | 111 | | | | | 111 | | 51 | 51 | | 111 |
| 111 | | | | | 111 | | | | | 111 | | | | | 111 |
| 111 | | | | | 111 | 111 | 111 | 111 | 111 | 111 | | | | | 111 |
| 111 | | | | | | | | | | | | | | | 111 |
| 111 | | | | | | | | | | | | | | | 111 |
| 111 | | | | | | | | | | | | | | | 111 |
| 111 | | | | | | | | | | | | | | | 111 |
| 111 | 111 | 111 | 111 | 111 | 111 | 111 | 111 | 111 | 111 | 111 | 111 | 111 | 111 | 111 | 111 |

(a) Variable `board` représentant une partie avec des murs sur les bordures et au centre du plateau. On y voit aussi deux joueurs, 0 et 1, ayant laissés deux traînées lumineuse en forme de "L".




(b) Affichage opéré par le client une fois que le `board` illustré en Fig. **??** a été reçu. Les joueurs et les murs de lumière sont affichés dans leur couleurs respectives.

FIGURE 5 – Exemple d'utilisation de la variable `board` contenu dans `display_info`.

## 5 Idées d'améliorations

Plusieurs améliorations peuvent être envisagées pour poursuivre ce projet.

- C-Tron en P2P. Une fois les joueurs connectés au serveur, un des deux joueurs est choisi pour "héberger" la partie. La connexion par le serveur se transforme alors en connexion directe entre les deux joueurs, permettant de libérer le serveur.
- Changer le protocole utilisé pour UDP au lieu de TCP.
- Permettre des parties à plus de deux joueurs (par équipe ou en free-for-all) avec un lobby de connexion.
- N'importe quelle autre idée !

Attention : bien que vous soyez libres d'apporter les améliorations que vous souhaitez, **vous veillerez à fournir également une version de votre code respectant les consignes originelles.** Cette version initiale sera testée à l'aide de nos propres programmes client et serveur. Le programme "amélioré" sera donc fourni dans un dossier à part.

## Annexes

Ces annexes détaillent les fonctions fournies dans le fichier `client_template.c`, vous permettant de vous familiariser avec l'utilisation de la librairie `ncurses` et de l'utilisation interactive du terminal. Notez que ces fonctions sont uniquement nécessaires pour le client, car le serveur n'affiche rien à l'écran.

## A Entrée standard non-bufferisée

Par défaut, l'entrée standard est bufferisée : l'entrée clavier d'un utilisateur n'est prise en compte qu'une fois la touche "entrée" appuyée.

Pour qu'un joueur puisse contrôler son light cycle sans avoir à valider chaque changement de direction en apuyant sur la touche "entrée", il est nécessaire de modifier le terminal via le code client.

Les modifications à apporter peuvent être vues dans la fonction `tune_terminal()`, fournie dans le fichier `client_template.c`. Cette fonction doit être appelée au début de votre fonction `main()`.

# B Utiliser `ncurses`

## B.1 Compilation & initialisation

Afin d'utiliser `ncurses`, cette librairie doit être `include`. De plus, **l'option `-lncurses` doit être utilisée à la compilation**.

Plusieurs fonctions doivent être appelées afin d'initialiser `ncurses`. Ces appels sont rassemblés dans la fonction `init_graphics()` (présente dans le fichier `client_template.c`), qui initialise également les différentes couleurs pouvant être utilisées. Cette fonction doit être appelée au début de la fonction `main()`.

## B.2 Affichage & couleur

Les couleurs dans `ncurses` sont instanciées par pair, définissant à la fois la couleur du caractère affiché ainsi que la couleur de l'arrière-plan derrière ce caractère. Ce couple de couleur est identifié par un entier. Par exemple, `init_pair(1, COLOR_BLUE, COLOR_BLACK);` est un couple de couleur identifié par l'entier $1$, permettant d'utiliser du bleu sur fond noir.

Dans ce projet, il sera nécessaire d'utiliser des couleurs différentes afin de distinguer les joueurs. Certains couples de couleurs ont déjà été instanciés pour vous dans la fonction `init_graphics`. Des constantes ont été définies afin de simplifier leur identification. Par exemple, la couleur permettant d'afficher un caractère rouge sur fond noir est identifié par la constante `RED_ON_BLACK`.

Afin de faciliter l'utilisation de ces couleurs, l'entier identifiant une couleur rouge sur fond rouge est l'identifiant de la couleur rouge sur fond rouge auquel `TRAIL_INDEX_SHIFT` a été rajouté. Par exemple, `RED_ON_BLACK` permet d'afficher un caractère rouge sur fond noir, et `RED_ON_BLACK + TRAIL_INDEX_SHIFT` permet d'afficher un caractère rouge sur fond rouge (`RED_ON_RED` peut également être utilisé).

Notez qu'il est normalement suffisant d'utiliser en tant que `color` la valeur de la case que l'on s'apprête à afficher.

Pour afficher un caractère à l'écran, la fonction `display_character(int color, int y, int x, char caractere)` est mise à votre disposition. Cette fonction affiche un caractère à la position $(x, y)$, en utilisant le couple de couleur identifié par l'entier `color`. Par exemple, afficher un "X" bleu sur fond noir en haut à gauche de l'écran peut être fait en utilisant `display_character(BLUE_ON_BLACK, 0, 0, X)`.

Afin d'afficher du texte, p. ex, pour indiquer la victoire d'un joueur, il suffit d'un simple appel à la fonction `mvaddstr(y, x, char* str);`. Par exemple, `mvaddstr(15, 15, "TEST");` affichera `TEST` aux coordonnées $(15, 15)$.

**Il est important de noter que pour afficher un texte ou un caractère rajouté à l'écran à l'aide de ces fonctions, il est nécessaire d'appeler la fonction `refresh()`. Pour rafraîchir l'écran dans sa totalité et supprimer le contenu précédent, on peut appeler la fonction `clear()`.**

Un exemple d'utilisation, affichant des "X" avec des couleurs aléatoire ainsi que du texte dans un cadre peut être étudié dans le fichier fourni sur Moodle.