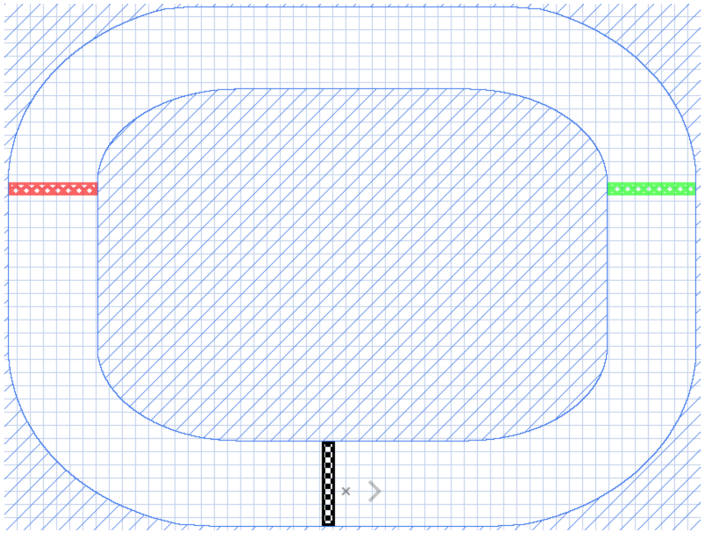
Lycée Réaumur - Laval (53)
Lycée de l'Hyrôme - Chemillé-en-Anjou (49)
Lycée Europe R. Schuman - Cholet (49)

24h du code 2025 : Vectorun

---

1. Principe du jeu

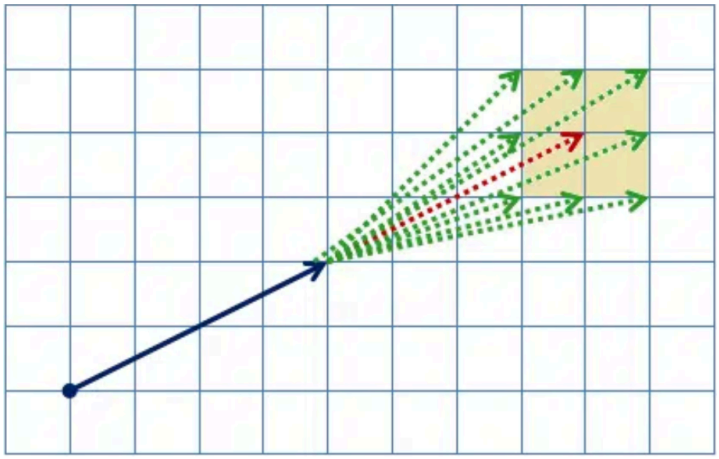
Vectorun est un jeu de course basé sur un système de vecteurs. Les joueurs doivent piloter leur véhicule en modifiant leur vitesse (taille) et direction à chaque tour, en respectant les lois de la physique. Le but est de parcourir le circuit en un minimum de coups et sans sortir de la piste.



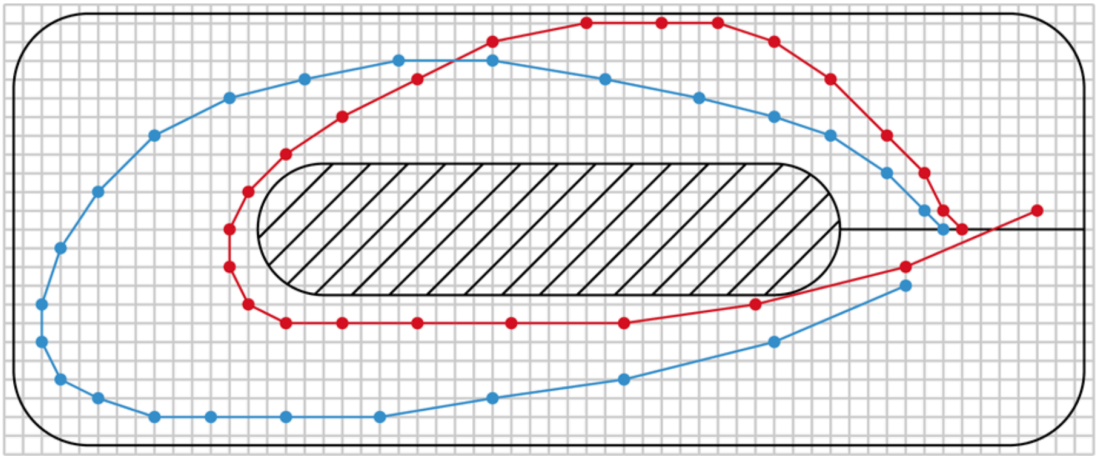
- Le joueur commence avec une vitesse nulle
- À chaque tour, il peut modifier son vecteur vitesse
- Le but est d'atteindre la ligne d'arrivée en restant sur la piste et en passant tous les checkpoints (ici en vert et rouge)

Tous les déplacements se feront d'un point de la grille à un autre point de la grille. Chaque point de grille a huit points de grille voisins : haut, bas, gauche, droite et les quatre directions diagonales. Les joueurs se relaient pour déplacer leurs voitures selon quelques règles simples. Chaque coup est marqué en traçant une ligne depuis le point de départ de ce coup jusqu'à un nouveau point.

Ci-dessous est représenté un exemple de mouvement dans le jeu de course. Le coup précédent du joueur est la flèche en bleu; son prochain mouvement peut être vers le point principal marqué en rouge, ou vers l'une de ses cases environnantes, indiquées en vert :



Un exemple de l'état d'une grille à la fin d'une partie avec deux joueurs est présentée ci-dessous :



2. Fonctionnement du jeu

Le jeu Vectorun sur lequel vous allez travailler est basé sur une architecture client-serveur. Le serveur de jeu est déjà développé et disponible dans votre starter pack sur :
https://24hducode.forge.apps.education.fr.

2.1. Format des messages

Tous les messages envoyés ou reçus sont sous format JSON et de la forme suivante :

```
{
  "type": "TYPE_DU_MESSAGE",
  "payload": "CONTENU_DU_MESSAGE"
}
```

- Le champ `type` correspond au type de message. Les messages peuvent être de type `YOPYOP`, `GO`, `PLAY`, `YOUR_TURN`, `GET_MAP`, `GET_DEPARR`, `GET_CHECK`, `END_GAME` ou `ERR`
- Le champ `payload` correspond au contenu du message. Il peut être variable en fonction du type de message.

Vous trouverez en annexe, une description détaillée des messages que vous pouvez envoyer ou recevoir avec le serveur.

2.2. Scénario d'échanges détaillé

Comme dans tout protocole de communication, il y a des étapes à respecter pour que le client et le serveur puissent se comprendre correctement.

Étape 1

Lorsque l'on commence à discuter, c'est bien de dire bonjour. Dans cette première étape, le client envoie au serveur un message de type `YOPYOP`. Dans ce jeu, le client peut choisir d'être soit joueur (le client peut jouer), soit spectateur (le client voit l'état du jeu mais ne peut pas jouer). Ainsi, le contenu de ce premier message sera le rôle que le client veut prendre.

Si tout se passe correctement, le serveur devrait vous renvoyer lui aussi un message de type `YOPYOP` dans lequel il vous précise votre identifiant unique de client. A partir de ce moment, vous êtes connu et le serveur vous ajoute aux clients en attente du lancement de la partie.

Étape 2

Une fois que tous les clients se sont connectés au serveur et ont envoyés leur message `YOPYOP`, n'importe quel joueur peut décider de lancer la partie. Il faudra alors envoyer un message de type `GO` qui lancera la partie pour tous les clients connectés et en attente. Ce type de message n'est donc à envoyer qu'une seule fois par un seul client.

Le serveur répondra alors avec un message `GO` envoyé à tous les clients pour signifier que la partie commence.

Étape 3

A chaque tour, c'est le rôle du serveur de vous dire si c'est à vous de jouer. Ainsi, vous pouvez recevoir à n'importe quel moment un message de type `YOUR_TURN` de la part du serveur qui signifie que c'est à votre tour de jouer.

Vous répondez alors avec un message de type `PLAY` et dont le contenu est une liste de deux éléments `[x,y]` de l'évolution de votre vecteur vitesse. x et y sont alors compris entre -1 et 1.

A chaque fois qu'un joueur joue, tous les clients reçoivent l'état de tous les vecteurs grace à un message `PLAY` envoyé par le serveur.

Étape 4

A la fin, lorsque la partie se termine, le serveur envoie un message de type `END_GAME` en vous précisant dans le contenu si vous avez gagné ou perdu.

2.3. Autres échanges

En parallèle des échanges obligatoires présentés précédemment, vous pouvez également, a votre initiative, demander des informations supplémentaires au serveur :

- un message de type `GET_MAP` vous renverra la carte sous forme d'un tableau à deux dimensions composé de `0` et de `1`
- un message de type `GET_DEPARR` vous renverra les coordonnées du point de départ sous forme d'un tableau à deux dimensions

- un message de type `GET_CHECK` vous renverra les coordonnées des différents checkpoints sous forme d'un dictionnaire (les numéros des checkpoints en clé, la liste des cases allouées en valeur)

3. Vos missions

Vous retrouverez ci-dessous les différentes missions à accomplir avec le nombre de points pour chacune d'entres-elles. Le nombre total de points par équipe est de 115.

Pour tous les clients suivants, la connexion au serveur devra être automatique et ils devront lancer les commandes de récupération de la carte, du départ et des checkpoints. Ceci vous rapportera 5 points d'office.

3.1. Client texte (25 points)

Le but de cette mission est de permettre à l'utilisateur de jouer à Vectorun depuis une interface en ligne de commandes. Elle devra répondre aux exigences suivantes :

- demande les changements de vecteurs (5 pts)
- demande les changements de vecteurs de manière efficace (5 pts)
- le client arrive à jouer une partie complète sans bug (5 pts)
- affiche la carte avec des couleurs (5 pts)

3.2. Client graphique (25 points)

Permettre à l'utilisateur de contrôler le serveur et jouer à Vectorun depuis une interface graphique (web ou autre). Il faudrait avoir un affichage en temps réel de la carte et de proposer au joueur de modifier son vecteur vitesse à chaque tour.

- demande les changements de vecteurs (5 pts)
- demande les changements de vecteurs de manière efficace (5 pts)
- le client arrive à jouer une partie complète sans bug (5 pts)
- affiche la carte avec des images (5 pts)

3.3. NSIN Deck OLED (25 points)

Pour cette édition 2025 des 24 heures du code, une manette a été conçue et est à votre disposition. Le but de cette mission est de pouvoir contrôler le serveur et jouer à Vectorun depuis la NSIN Deck OLED. Elle dispose de :

- carte XIAO ESP32C3 : sert de micro-contrôleur (équivalent d'une carte Arduino, programmable avec l'IDE Arduino)
- 5 boutons poussoirs de couleurs différentes
- 1 joystick
- 1 afficheur OLED Grove
- 1 batterie LI-PO connectée à la carte XIAO

Il faudrait pouvoir prévisualiser l'état du vecteur au prochain coup sur l'écran OLED avant de valider et d'envoyer l'information au serveur.

- utilisation pertinente des boutons (5 pts)
- utilisation pertinente du Joystick (5 pts)
- utilisation pertinente de l'écran (5 pts)
- le client arrive à jouer une partie complète sans bug (5 pts)

3.4. Automatisation (25 points)

La phase automatisation est décomposée en plusieurs parties qui rapportent plus ou moins de points :

- réalise un tour complet sur une des cartes fournies sans se planter dans le décor ou sortir de la carte (5 pts)
- analyse par un algorithme d'une carte fournie et affichage des coups successifs (7 pts)
- réalise un tour complet d'une carte de la finale sans se planter dans le décor ou en sortir (8 pts)
- gagne le tournoi d'IA (bonus 15 pts supplémentaires)

Annexe 1 : messages détaillés

YOPYOP

- Client –>Serveur : Rôle que le client souhaite prendre sous forme d'un entier. 0 : joueur; 1 : spectateur
  Exemple :

```
{
  "type": "YOPYOP",
  "payload": 0
}

```

- Serveur –>Client : Identifiant unique du client sur le serveur sous forme d'une chaine de caractère
  Exemple :

```
{
  "type": "YOPYOP",
  "payload": "127.0.0.152645"
}

```

GO

Pas de contenu particulier, que ce soit du client vers le serveur ou inversement. Permet de lancer le jeu.

PLAY

- Client –>Serveur : évolution du vecteur position sous forme d'une liste avec pour premier élément la valeur en `x` et pour deuxième la valeur en `y`. Les deux doivent être comprises dans l'interval `[-1; 1]` Exemple :

```
{
  "type": "PLAY",
  "payload": [-1, 0]
}

```

- Serveur –>Client : liste des vecteurs de tous les joueurs sous forme d'un dictionnaire dont les clés sont les identifiants des joueurs et les valeurs sont les vecteurs associés au joueur. Exemple :

```
{
  "type": "PLAY",
  "payload": {"127.0.0.152645": [[23, 45], [-1, 0]], "127.0.0.152565": [[32, 54], [0,
   1]]}
}

```

YOUR_TURN

Message uniquement du serveur vers le client pour informer au client que c'est à lui de jouer. Exemple :

```
{
  "type": "YOUR_TURN",
  "payload": {'pos': [29, 56], 'vit': [0, 0]}
}
```

GET_MAP

- Client –>Serveur : pas de contenu
- Serveur –>Client : carte du jeu sous la forme d'une liste à deux dimensions contenant des 0 (la piste) et des 1 (les murs) Exemple :

```
{
  "type": "GET_MAP",
  "payload": [[1, 1, 0, 0, 1], [1, 1, 0, 0, 1], [1, 1, 0, 0, 1]]
}

```

GET_DEPARR

- Client –>Serveur : pas de contenu
- Serveur –>Client : ligne de départ et d'arrivée sous la forme d'une liste à deux dimensions Exemple :

```
{
  "type": "GET_DEPARR",
  "payload": [[29, 53], [29, 54], [29, 55], [29, 56], [29, 57], [29, 58], [29, 59]]
}

```

GET_CHECK

- Client –>Serveur : pas de contenu
- Serveur –>Client : coordonnées des différents checkpoints sous la forme d'un dictionnaire Exemple :

```
{
  "type": "GET_CHECK",
  "payload": {'1': [[27, 4], [27, 5], [27, 6], [27, 7], [27, 8]], '0': [[57, 36],
  [58, 36], [59, 36], [60, 36]], '2': [[4, 37], [5, 37], [6, 37], [7, 37], [8, 37],
  [9, 37], [10, 37]]}
}

```

END_GAME

Message du serveur vers le client pour informer de la fin du jeu. Le contenu est soit `WON_BY_YOU` (vous avez gagné), soit `WON_BY_ID_JOUEUR` (vous avez perdu et c'est le joueur spécifié dans le message qui a gagné).

Exemple :

```
{
  "type": "END_GAME",
  "payload": "WON_BY_127.0.0.152645"
}
```

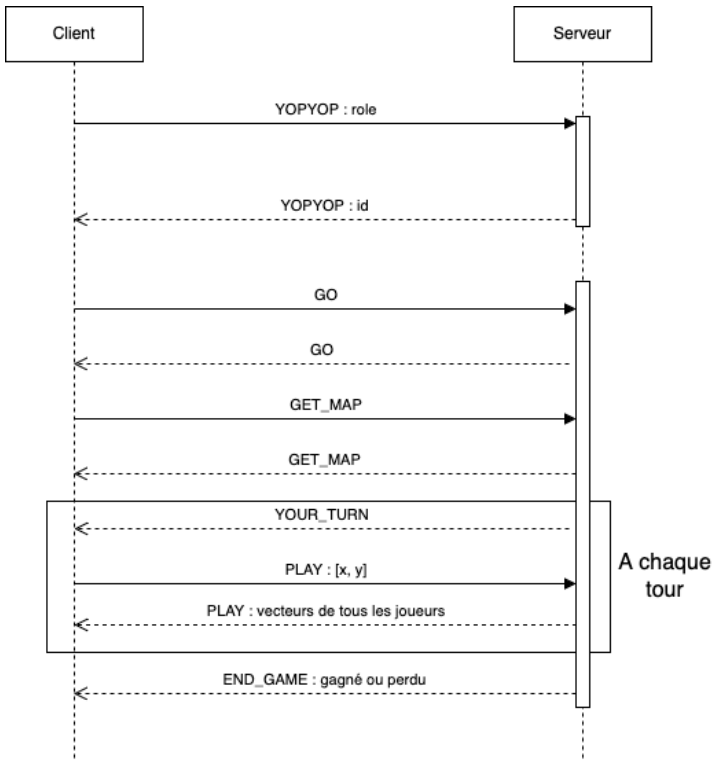
ERR

Message du serveur vers le client pour informer d'une erreur. Les codes d'erreurs varient en fonction du contexte et sont décrits dans l'annexe 2.

Exemple :

```
{
  "type": "ERR",
  "payload": "NOT_YOUR_TURN"
}
```

Un exemple type des échanges entre le client et le serveur est donné ci-dessous par un diagramme de séquence :



Annexe 2 : liste des messages d'erreurs

Code d'erreur	Signification
NOT_JSON	Votre message envoyé n'est pas dans le bon format et est incompréhensible par le serveur
BAD_INPUT	Le contenu du message ne correspond pas à ce qui était attendu
ALREADY_STARTED	Renvoyé à la suite d'un message `GO` alors que le jeu a déjà démarré
ROLES_NOT_DEFINED	Tout le monde n'a pas encore envoyé un message `YOPYOP` et les rôles ne sont donc pas tous définis
VECTOR_NOT_VALID	Le vecteur position que vous avez envoyé n'est pas valide
NOT_YOUR_TURN	Ce n'est pas encore votre tour de jouer
COMMAND_UNKNOWN	Le type de message que vous avez spécifié est inconnu