# Coding War

## Description

L’objectif est de créer une intelligence artificielle qui participera à une partie de poker. Des équipes, d’un maximum de 3 personnes, seront constituées et disposeront de 4 mois pour réaliser l’IA  « parfaite ». Les équipes joueront ensuite les unes contre les autres afin de déterminer un vainqueur.

### Inscriptions

Les inscriptions commenceront à partir de la semaine du 28 mai (à partir de la réception de la communication) et se termineront le 7 septembre. Elles se feront en envoyant un mail à « kevin.bonnoron@cgi.com »ou « remi.sarraute@cgi.com » avec dans le corps du message, le nom de l’équipe et le nom des participants.

### Conditions de victoire

Pour remporter la compétition il faut :

* Soit battre tous ses adversaires (plus de jeton)
* Avoir le plus de jetons au bout de 150 mains

### Informations complémentaires

Le non-respect d’une des règles indiquées dans ce document provoquera la disqualification immédiate de l’équipe à son origine.

Ce document peut être modifié au cours des 4 mois du développement pour modifier / ajuster certaines règles. Si cela survient les équipes seront notifiées par mail des changements. Par ailleurs aucune modification ne sera effectuée après la date de fin des inscriptions.

## Règles

Les règles du poker « Texas Hold’em no limit » seront utilisées avec quelques variantes afin de les adapter pour notre besoin.

Il est interdit de :

* Utiliser le code d’une ia existante sur le web (soyez fair-play, le but étant que tout le monde s’amuse)
* Utiliser n’importe quelle technique de hacking sur le serveur
* D’entrer les actions que l’IA devra faire de manière humaine. Une fois l’IA connecté sur le serveur, aucune manipulation humaine / externe à l’IA devra être faite.
* S’inscrire plusieurs fois dans différentes équipes (il reste néanmoins possible de changer d’équipe en cours de route)

Voici les règles supplémentaires :

* Aucune restriction sur le langage de programmation (dès l’instant qu’il est capable de communiquer par Socket TCP)
* Les IA joueront tour à tour et transmettront le coup qu’elles feront,
* Chaque IA disposera au maximum de 15 secondes pour répondre,
* Si une IA tente une action interdite, elle sera ignorée par le serveur et le serveur continuera d’attendre que l’IA envoie un coup valide jusqu’à épuisement de son temps de réponse.
* Si une IA envoie un message mal formaté, le message ne sera pas pris en compte et le serveur continuera d’attendre que l’IA envoie un message valide jusqu’à épuisement de son temps de réponse.
* Si une IA tente une action interdite, elle sera considérée comme « fold » pour la main en cours. Par exemple :
  + jouer en dehors de son tour,
  + miser plus de jetons que ce qu’elle possède,
  + miser moins que 0,
  + miser moins que la dernière mise ou que la grosse blinde (exception pour le « all in »),
* Si une IA met trop de temps à répondre le serveur considérera :
  + check si aucune mise n’est présente,
  + fold si une mise est présente
* Aucune recave
* Le tapis de départ est de 1500
* Les blindes augmentent de la manière suivante :
  + 10/20 pour les 15 premières mains,
  + 20/40 pour les 15 mains suivantes,
  + 30/60 pour les 20 mains suivantes,
  + 40/80 pour les 30 mains suivantes,
  + 50/100 pour les 40 mains suivantes,
  + 75/150 pour les 15 mains suivantes,
  + 100/200 pour les 15 dernières mains

## API

Voici la définition des structures et des messages utilisant ces structures. La définition est sous forme json.

### Réseau

L’intelligence artificielle se connectera au serveur sur le port 1300 en utilisant le protocole TCP. Elle transmettra et recevra des chaînes de caractères au format JSON.

### Structures

Voici la définition des structures qui seront utilisées dans les dialogues entre les intelligences artificielles et le serveur. Le format JSON sera utilisé pour les échanges.

#### Message

Structure représentant le message transmis entre le serveur et les joueurs.

{

"id": String,

"data": Any

}

* id : identifiant du message
* data : contenu du message (diffère selon le message)

#### Player

Structure représentant un joueur (= ia). Cela peut représenter le joueur courant comme n’importe quel autre.

{

"id": Integer,

"name": String,

"chips": Integer,

"state": Enum(ACTIVE, FOLDED, ELIMINATED)

"dealer": boolean

}

* id : utilisé pour identifier de manière unique un joueur
* name : nom du joueur (valorisé avec le nom de l’équipe)
* chips : nombre de jetons (non misés)
* state : indicateur de l’état du joueur
  + ACTIVE : joueur encore en lice (non éliminé), non couché et non all-in,
  + FOLDED : joueur qui s’est couché,
  + ELIMINATED : joueur éliminé
* dealer : indicateur si le joueur est le dealer pour la main en cours

#### Card

{

"kind": Enum(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, JACK, QUEEN, KING)

"color": Enum(SPADE, HEART, DIAMOND, CLUB)

}

* kind : type de la carte
* color : couleur de la carte
  + SPADE : pique
  + HEART : cœur
  + DIAMOND : carreau
  + CLUB : trèfle

#### Action

Structure représentant un coup que le joueur peut faire

{

"value": Integer

}

* value : montant de la mise
  + Considéré comme CHECK, si « value » vaut 0 et qu’aucun joueur n’a déjà misé,
  + Considéré comme FOLD, si « value » vaut 0 et qu’au moins un joueur a déjà misé,
  + Considéré comme CALL, si « value » vaut la dernière somme misée,
  + Considéré comme RAISE, si « value » vaut plus que la dernière somme misée

### Messages Serveurs

A noter que les références vers les structures sont précédées d’un #

#### server.lobby.join.success

Transmis lors de l’enregistrement avec succès de l’équipe au lobby

{

"id": "server.lobby.join.success"

}

#### server.lobby.join.failure

Transmis lors de l’enregistrement en erreur de l’équipe au lobby

{

"id": "server.lobby.join.failure"

"data": {

"reason": String

}

}

* reason : Raison du rejet de l’enregistrement de l’équipe.

#### server.game.start

Transmis une seule fois au début de la partie pour indiquer au joueur que la partie commence.

{

"id": "server.game.start",

"data": {

"info": [#Player](#_Player),

"count": Number

}

}

* info : les informations du joueur courant
* count : le nombre de participants

#### server.game.cards

Transmis quand le joueur reçoit ses cartes

{

"id": "server.game.cards",

"data": {

"cards": [[#Card](#_Card)]

}

}

#### server.game.hand.start

Transmis quand une nouvelle main commence

{

"id": "server.game.hand.start",

"data": {

"players": [[#Player](#_Player)]

}

}

* players : liste des joueurs de la partie (même les joueurs éliminés qui seront mis à la fin du tableau) triés dans l’ordre dans lequel ils joueront

#### server.game.play

Transmis quand le serveur demande au joueur d’indiquer son coup

{

"id": "server.game.play"

}

#### server.game.play.timeout

Transmis quand le joueur a mis trop de temps avant de répondre

{

"id": "server.game.play.timeout"

}

#### server.game.play.success

Transmis quand le coup d’un joueur est valide et à bien été pris en compte par le serveur

{

"id": "server.game.play.success"

}

#### server.game.play.failure

Transmis quand le coup d’un joueur est non valide et n’a pas été pris en compte par le serveur

{

"id": "server.game.play.failure"

}

#### server.player.action

Transmis par le serveur à tous les joueurs lorsqu’une IA à jouer un coup valide (sauf à l’ia qui a joué le coup)

{

"id": "server.player.action",

"data": {

"id": Integer

"action": [#Action](#_Action)

}

}

* id : identifiant du joueur qui a fait l’action
* action : action effectué par le joueur

#### server.game.board.cards

Transmis par le serveur quand des cartes sont ajoutés sur le tapis

{

"id": "server.game.board.cards",

"data": {

"cards": [[#Card](#_Card)]

}

}

#### server.game.hand.end

Transmis quand une main se termine

{

"id": "server.game.hand.end",

"data": {

"winners": [[#Player](#_Player)]

}

}

#### server.game.end

Transmis une seule fois en fin de partie pour indiquer aux joueurs que la partie est terminée.

{

"id": "server.game.end",

"data": {

"winner": [#Player](#_Player),

}

}

* winner : le joueur gagnant

### Messages clients

#### client.lobby.join

{

"id": "client.lobby.join",

"data": {

"name": String

}

}

* name: Nom de l’équipe

#### client.game.action

{

"id": "client.game.action",

"data": {

"action": [#Action](#_Action)

}

}

* action : l’action que l’ia désire faire