Rapport technique du projet BD & RÉSEAU

Salle d'arcade

Réalisation d'une architecture associant réseau, base de donnée et site web d'une salle d'arcade

Classe	L3 Groupe D-3
Établissement	CY Université
Professeurs	M. Lemaire & Mme Dang Ngoc

Version du document	5
Date de modification	30/11/25

Réalisé par ZURKHANG Tenzin, YANG Kaiwei et COSTA Mathéo

Sommaire

Historique des versions

Versions apportées

Version 5 (30 Novembre 2025)

Réalisation du serveur web et des pages web pour les clients et administrateurs.
Complétion des codes du serveur et du client avec la connexion à la base de donnée.
Quelques modifications apporté au jeu de donnée et suppression du score par temps et du fournisseur de jeu.
Ajout des résultats des requêtes SQL
Explication plus détaillé de la partie réseau
Ajout des codes importants dans l'annexe

Version 4 (9 Novembre 2025)

Le code SQL a été déplacé dans l'annexe et le jeu de données a été séparé dans la partie BD.
L'implémentation du code du serveur et du client a été effectuée dans l'annexe, avec leurs tests respectifs.
L'implémentation en scripts SQL des 10 questions en français.
Quelques changements ont été apportés au jeu de données et à notre modélisation BD.

Version 3 (18 Octobre 2025)

Séparation en 2 parties : Base de Données et Réseau
Modélisation complète de la BD avec son jeu de données, les 10 questions vers la bd.
L'implémentation de la partie SQL avec le DDL et le DML.
Les nouveaux visuels des diagrammes applicatifs.

Version 2 (05 Octobre 2025)

Révision du rapport en entier avec la rectification du dictionnaire de donnée, des échanges réseaux et du planning prévisionnel
Ajout de la page de garde, du MLD, du MCD, du sommaire et des versions du documents.

Version 1 (21 Septembre 2025)

Création du compte rendu pour l’ébauche des données réseaux, les dictionnaire de donnée, le MLD et le MCD. Réalisation du planning prévisionnel.

Présentation du contexte

Nous avons choisi la gestion d'une salle d'arcade car ce milieu, à la fois emblématique et en pleine transformation, offre un cas d'étude riche mêlant techniques diverses et enjeux concrets. Malgré la concurrence des jeux en ligne et des consoles personnelles, les salles d'arcade restent des espaces privilégiés de socialisation et de compétition locale, bien que souvent gérés avec des systèmes peu flexibles ou déconnectés.

Notre projet vise à moderniser cette gestion en proposant une base de données centralisée pilotant joueurs, jetons, bornes et scores. Cette centralisation garantit une meilleure traçabilité des consommations et joue un rôle clé pour améliorer l'expérience utilisateur, par exemple en permettant à un joueur de retrouver son compte dans plusieurs salles et de profiter de systèmes de récompenses automatisés.

L'intégration d'un protocole réseau client-serveur, attendu par le module, s'inscrit naturellement dans cette logique : il simule la communication entre bornes et serveurs, assurant la synchronisation en temps réel des données et renforçant ainsi la cohérence globale du système. Cela ouvre également la porte à de futures évolutions, comme des classements dynamiques ou des événements multicentriques.

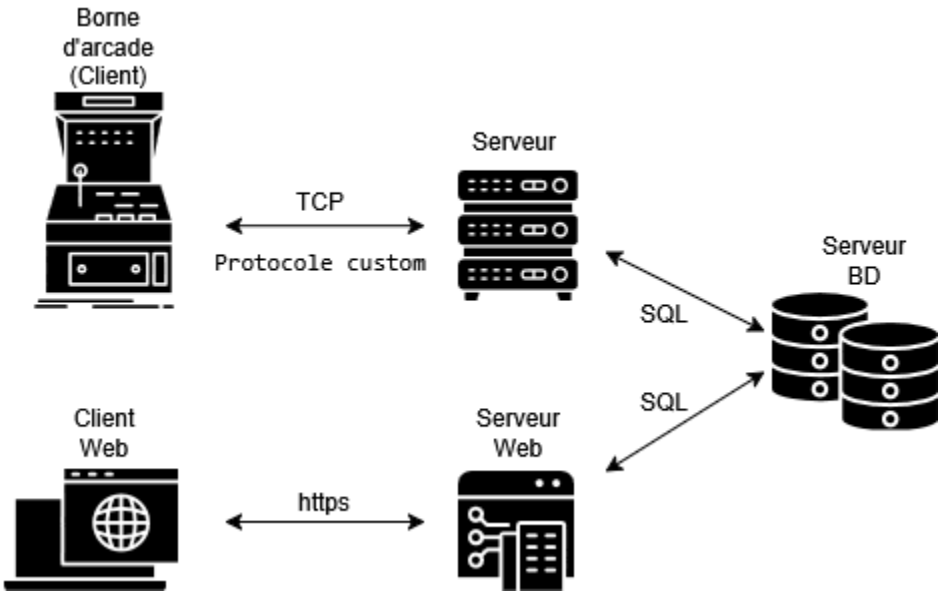

Figure 1: Architecture générale du projet

Nos objectifs clés

- Moderniser un univers traditionnel avec une gestion automatisée simplifiant les interactions pour joueurs et exploitants.
- Assurer la fluidité et la fiabilité des échanges réseau pour une expérience utilisateur sans rupture.
- Favoriser la compétitivité et la fidélisation grâce à un système de récompenses basé sur des données précises.
- Offrir une architecture évolutive permettant d’ajouter aisément des fonctionnalités futures.

Planning prévisionnel

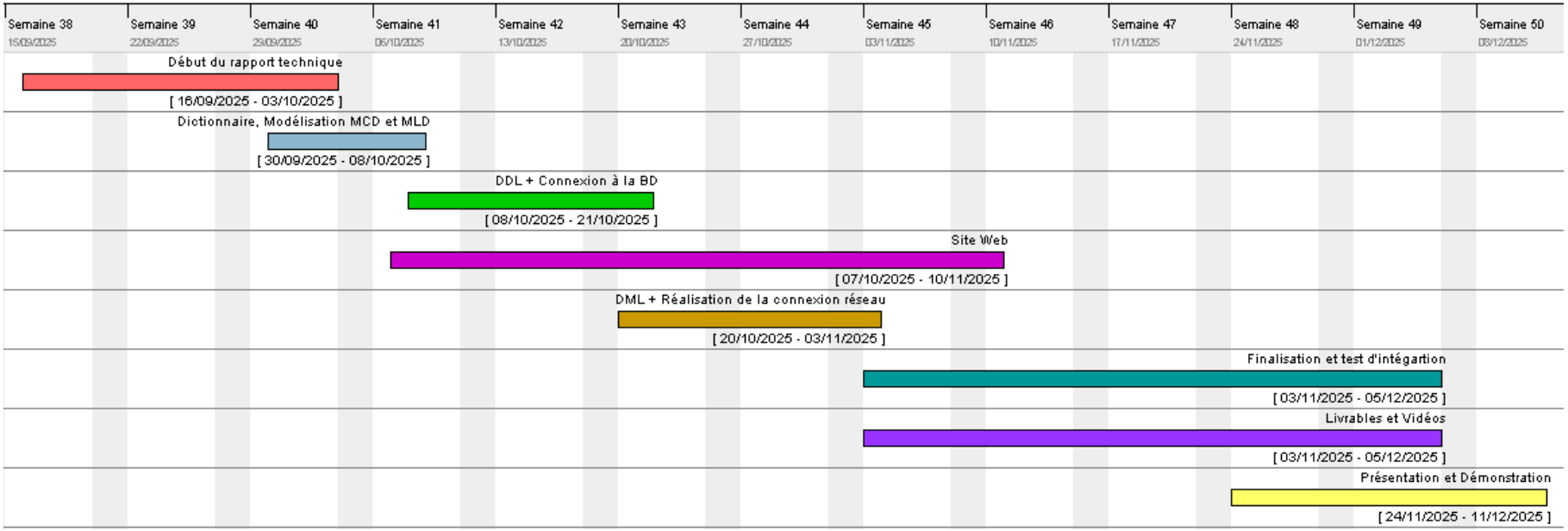

Figure 2: Diagramme de Gantt

Base de données

Modélisation

Dictionnaire de données

- Personne :

	Type	Domaine	Autres Contrainte	Exemple	N / NN
id_perso	int		PK, auto-increment	1	NN
nom	varchar(25)			Titi	NN
prenom	varchar(25)			Tata	NN
sexe	ENUM	(“homme”, ”femme”)		homme	NN
date_naissance	DATE	aaaa-mm-dd		2000-12-31	NN
mail	varchar(40)		UNIQUE	tititata@exemple.com	NN
tel	char(10)	numérique (0-9)	UNIQUE	0123456789	Null
login	varchar(25)		UNIQUE	Steph	NN
mot_de_passe				$2y$10$EixZaYVK1fsbw1ZfbX3OXePaWxn96p36M.otCqfdYjfE2zTvkgqH2	NN

➢ Client :

	Type	Domaine	Autres Contrainte	Exemple	N / NN
id_client	int		PK, auto-increment	1	NN
solde_jetons	int	>= 0	Par défaut : 0	10	NN
date_inscription	DATE	aaaa-mm-dd	Par défaut : now()	2025-01-24	NN
pseudo	varchar(25)				NN

➢ Admin :

	Type	Domaine	Autres Contrainte	Exemple	N /NN
id_admin	int		PK, auto-increment	1	NN
date_anciennete_employe	DATE	aaaa-mm-dd	Par défaut : now()	2020-06-03	NN
poste_employe	ENUM	(“gerant”, “nettoyage”, “personnel”)		gerant	NN
status_employe	ENUM	(“travail”, “en_pause”, “repos”)	Par défaut : travail	travail	NN

➢ Carte :

	Type	Domaine	Autres Contrainte	Exemple	N / NN
id_carte	int		PK,auto-increment	2	NN
status_carte	ENUM	(“Perdu”, “Liée”)	par défaut : Liée	Perdu	NN
date_carte	DATE	aaaa-mm-dd	par défaut : now()	2025-01-24	NN
version_carte	int	> 0		1	NN

➢ Recharge :

	Type	Domaine	Autres Contrainte	Exemple	N / NN
id_recharge	int		PK, auto-increment	10	NN
montant_jeton	int	>= 0		15	NN
mode_paiement	ENUM	(“CB”, “espèce”)		CB	NN
date_recharge	DATE	aaaa-mm-dd		2025-10-01	NN

➢ Fournisseur :

	Type	Domaine	Autres Contrainte	Exemple	N / NN
id_four	int		PK, auto-increment	22	NN
nom_four	varchar(20)			6	NN
mail_four	varchar(25)		UNIQUE	tata@exemple.com	NN
tel_four	char(10)	numérique (0-9)	UNIQUE	0987654321	NN
adresse_four	varchar(30)			35 R. de Turbigo, 75003 Paris	NN

➢ Jeux :

	Type	Domaine	Autres Contrainte	Exemple	N / NN
id_jeux	int		PK, auto-increment	6	NN
nom	varchar(50)		UNIQUE	Pac-man	NN

➢ Jeux_Solo :

	Type	Domaine	Autres Contrainte	Exemple	N / NN
id_jeux_solo	int		PK, auto-increment	6	NN
difficulte	enum	(“Facile”, “Normal”, “Difficile”)		Normal	NN

➢ Jeux_Multi :

	Type	Domaine	Autres Contrainte	Exemple	N / NN
id_jeux_multi	int		PK, auto-increment	7	NN
type_equipe	ENUM	(“Coop”, “vs”)		Coop	NN
nb_joueur_min	DATE	>= 2 & <=nb_joueur_max		4	NN
nb_joueur_max	ENUM	>= nb_joueur_min & <=4		4	NN

➢ Borne :

	Type	Domain	Autres Contrainte	Exemple	N / NN
id_borne	int		PK, auto-increment	1	NN
etat_borne	ENUM	(“Disponible”, “En Maintenance”, “HS”)	par défaut : Disponible	En Maintenance	NN
date_achat	DATE	aaaa-mm-dd		2025-10-01	
prix_jeton	int		>0	1	NN

➢ Partie :

	Type	Domain	Autres Contrainte	Exemple	N / NN
id_partie	int		PK, auto-increment	8	NN
score	int	>= 0		200	Null
date_partie	DATE	aaaa-mm-dd		2025-10-01	NN
récompense	ENUM	(“5”, “2”, “1”,”0”)	Par défaut 0	0	NN
status_partie	ENUM	(“En cours”, “Termine”, “Recompense donnee ”)	par défaut : En cours	Termine	NN

MCD

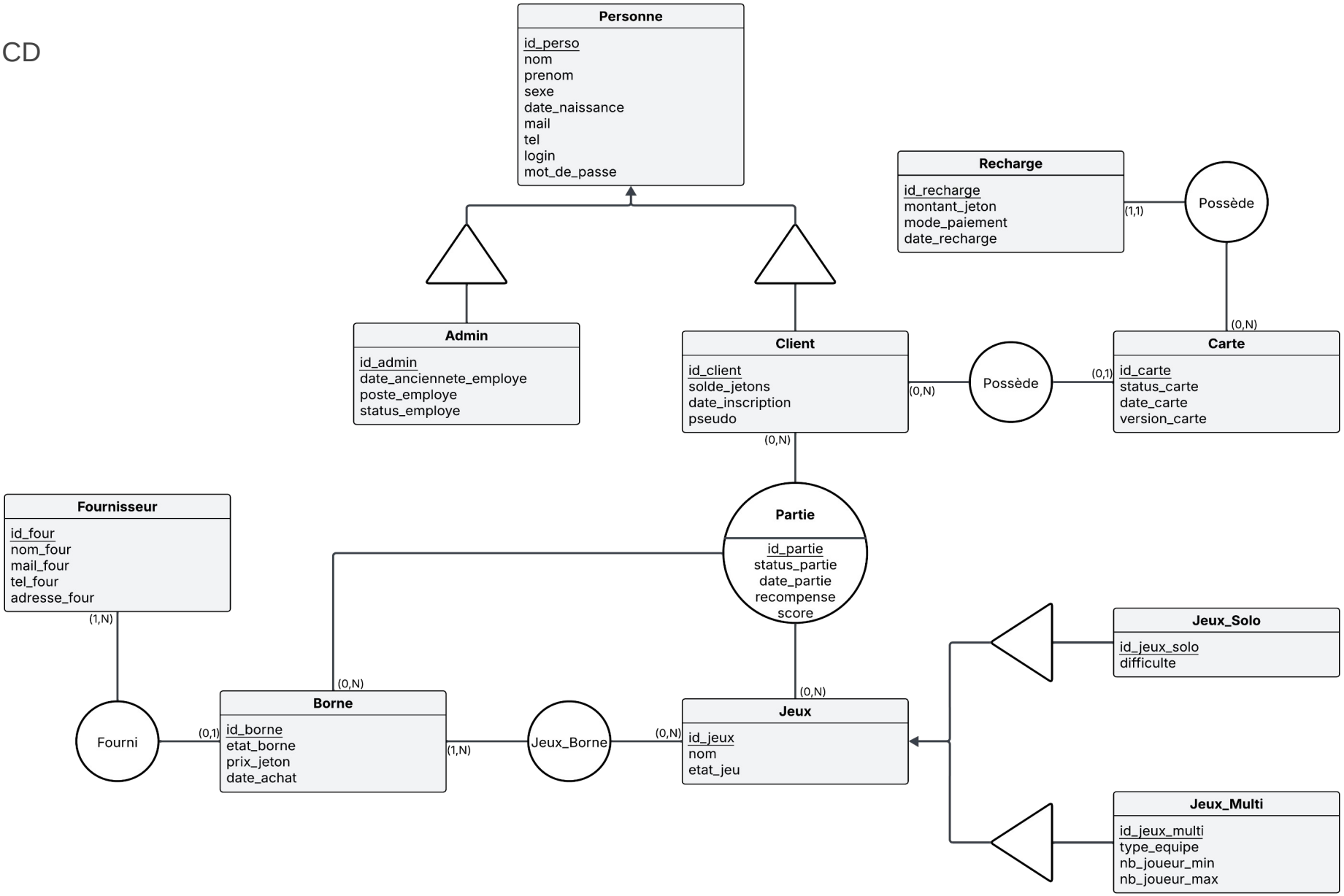

Figure 3: MCD

MLD

- Personne (id_perso, nom, prenom, sexe, date_naissance, mail, tel, login, mot_de_passe)
- Client (#id_client, solde_jetons, date_inscription, pseudo)
- Admin (#id_admin, date_anciennete_employe, poste_employe, status_employe)
- Carte (id_carte, status_carte, date_carte, version_carte, #id_client)
- Recharge (id_recharge, montant_jeton, mode_paiement, date_recharge, #id_carte)
- Fournisseur (id_four, nom_four, mail_four, tel_four, adresse_four)
- Jeux (id_jeux, nom, etat_jeu)
- Jeux_Solo (#id_jeux_solo, difficulte)
- Jeux_Multi (#id_jeux_multi, type_equipe, nb_joueur_min, nb_joueur_max)
- Borne (id_borne, etat_borne, prix_jeton, date_achat, #id_four)
- Jeux_Borne(#id_Jeux, #id_Borne)
- Partie (id_partie, score, date_partie, recompense, status_partie, #id_client, #id_jeux, #id_borne)

Jeu de donnée

- Personne :

id_perso	nom	prenom	sexe	date_naissance	mail	tel	login	mot_de_passe
1	Dupon	Alice	femme	1995-02-15	alice.dupont@mail.com	0612345671	user01	$2y$10$5ozcep67XkxrvBztGBOtZepHS6DMcoDK/9g0DxFz3hKQfFF810c9u
2	Martin	Bob	homme	1990-06-20	bob.martin@mail.com	0612345672	bob_	$2y$10$Gu/WLzQZh1CVYYXRUlj6fu3UMz.mFGl90CAnlK.aDqLF5Cpkk9Jl2
3	Durand	Caroline	femme	1988-11-05	caroline.durand@mail.com	0612345673	Caro	$2y$10$ZaWNAk9dr3SHxBGPWd2bee7TwgiQhgAImMa7/PXwH6OibjMCOFfB6
4	Lefevre	David	homme	1992-03-30	david.lefevre@mail.com	0612345674	DAVID	$2y$10$NayLrL1dtZ0aG4uYPIdVyuCmGDfreyOb3TLZ/OKK1u5kjzb6jlrs.
5	Moreau	Emma	femme	1997-08-12	emma.moreau@mail.com	0612345675	Ma	$2y$10$3ZXeNSYAEPySHB9SmSfEt.BXGgWCJlWcXR7mihzoWQrTX4ypYEb02
6	Petit	Franck	homme	1985-05-10	franck.petit@mail.com	0612345676	Franc	$2y$10$1t7ME9ZTIEqPaF.He7m8FejbLd0zH/OvSROgtJCeb4LLGSRutJyIy
7	Rousseau	Gina	femme	1991-07-22	gina.rousseau@mail.com	0612345677	Gina	$2y$10$56WrwMzFld5nWzCT3Les6.VUC84.6UnrHCDTQDfBh8pNf0lfePPTK
8	Blanc	Hugo	homme	1993-12-11	hugo.blanc@mail.com	0612345678	white	$2y$10$65w5XV2PslXmSj7z7nR3CeJvqTpSXAPTyff7uXuUw3cqVqj23pr9G
9	Morel	Isabelle	femme	1989-09-01	isabelle.morel@mail.com	0612345679	ISA	$2y$10$C5bgpDXUJPLzt5PB2SQgfe2YugPR352qLZzlMcDCn2.guzGicnfRi
10	Girard	Jacques	homme	1994-04-18	jacques.girard@mail.com	0612345680	PasDebol	$2y$10$INBblDp34vs9EkXC3AOyT.WTXloiCorJYMy.ohmi6nYV/SHx07/Ga

- Client :

id_client	pseudo	solde_jetons	date_inscription
1	Alice9	100	2023-01-10
2	BobLeBricoleur5	50	2023-02-15
3	88_caro	200	2023-03-20
4	DAVID	75	2023-04-25
5	Emma__	150	2023-05-30

- Admin :

id_admin	date_anciennete_employe	poste_employe	status_employe
6	2020-01-01	gerant	travail
7	2021-06-15	nettoyage	en_pause
8	2019-09-10	personnel	travail
9	2022-03-20	gerant	repos
10	2020-11-05	personnel	travail

- Carte :

id_carte	status_carte	date_carte	version_carte	id_client
1	Liee	2023-01-11	1	1
2	Liee	2023-02-16	1	2
3	Perdu	2023-03-21	2	4
4	Liee	2023-04-26	2	4
5	Liee	2023-05-31	1	5

- Recharge :

id_recharge	montant_jeton	mode_paiemen	date_recharge	id_carte
1	50	CB	2023-01-15	1
2	100	espece	2023-02-20	2
3	75	CB	2023-03-25	3
4	150	CB	2023-04-30	4
5	200	espece	2023-06-05	5

- Fournisseur :

id_four	nom_four	mail_four	tel_four	adresse_four
1	Nintendo	contact@nintendo.com	0611122233	12 rue du Jeu, Tokyo
2	Capcom	support@capcom.com	0622233344	45 avenue du Fun, Osaka
3	Sega	info@sega.com	0633344455	78 boulevard Play, Tokyo
4	Konami	sales@konami.com	0644455566	9 rue Arcade, Tokyo
5	Atari	contact@atari.com	0655566677	22 rue Jeton, Paris

- Jeux

id_jeux	nom	etat_jeu
1	Pacman	Disponible
2	Super Mario	Disponible
3	Tetris	En Maintenance
4	Zelda	Disponible
5	Donkey Kong	HS

- Jeux_Solo :

id_jeux_solo	difficulte
1	Facile
2	Normal
3	Difficile

- Jeux_Multi :

id_jeux_multi	type_equipe	nb_joueur_min	nb_joueur_max
4	vs	2	4
5	Coop	4	4

- Borne :

id_borne	etat_borne	prix_jeton	date_achat	id_four
1	Disponible	5	2022-01-01	1
2	En Maintenance	10	2022-06-15	2
3	Disponible	4	2022-09-10	3
4	HS	5	2023-03-20	4
5	Disponible	8	2023-11-05	5

- Jeux_Borne

id_jeux	id_borne
1	1
2	1
1	3
3	3
4	3
1	5
2	5
3	5
4	5
5	5

- Partie

id_partie	score	date_partie	recompense	status_partie	id_client	id_jeux	id_borne
1	100	2023-07-01	0	Termine	5	1	1
2	85000	2023-07-02	2	Recompense donne	5	2	5
3	50	2023-07-03	1	Recompense donnee	1	1	3
4	120	2023-07-04	5	Recompense donnee	3	4	5
5	NULL	2025-11-0	0	En cours	4	2	5

10 requêtes SQL

Quel est le solde de chaque client?

```
SELECT pseudo, solde_jetons
FROM Client;
```

	pseudo character varying (25)	solde_jetons integer
1	Alice95	100
2	Bob90	50
3	Caroline88	200
4	David92	75
5	Emma97	150

Figure 4: Résultat SQL - Solde par client

Combien de parties chaque client a-t-il jouées?

```
SELECT c.pseudo, COUNT(p.id_partie) AS nb_parties
FROM Client c
LEFT JOIN Partie p ON c.id_client = p.id_client
GROUP BY c.pseudo;
```

	pseudo character varying (25)	nb_parties bigint
1	Caroline88	1
2	David92	1
3	Alice95	1
4	Emma97	2
5	Bob90	0

Figure 5: Résultat SQL - Nombres de parties par clients

Quel est le plus ancien employé ?

```
SELECT p.nom, p.prenom, a.date_anciennete_employe
FROM Admin a
JOIN Personne p ON a.id_admin = p.id_perso
ORDER BY a.date_anciennete_employe ASC
LIMIT 1;
```

	nom character varying (25)	prenom character varying (25)	date_anciennete_employe date
1	Blanc	Hugo	2019-09-10

Figure 6: Résultat SQL - Le plus ancien employé

Combien de personnes ont joué à chaque jeu?

```
SELECT j.nom AS jeu, COUNT(DISTINCT p.id_client) AS nb_joueurs
FROM Jeux j
LEFT JOIN Partie p ON j.id_jeux = p.id_jeux
GROUP BY j.nom
ORDER BY nb_joueurs DESC;
```

	jeu character varying (50)	nb_joueurs bigint
1	Pacman	2
2	Super Mario	2
3	Zelda	1
4	Donkey Kong	0
5	Tetris	0

Figure 7: Résultat SQL - Total des joueurs par jeux

Combien de jeux chaque borne propose ?

```
SELECT b.id_borne, COUNT(jb.id_jeux) AS nb_jeux_installes
FROM Borne b
LEFT JOIN Jeux_Borne jb ON b.id_borne = jb.id_borne
GROUP BY b.id_borne
ORDER BY b.id_borne;
```

	id_borne [PK] integer	nb_jeux_installes bigint
1	1	2
2	2	1
3	3	3
4	4	0
5	5	5

Figure 8: Résultat SQL - Les jeux par bornes

Quel est le fournisseur de chaque borne?

```
SELECT b.id_borne, f.nom_four AS fournisseur
FROM Borne b
JOIN Fournisseur f ON b.id_four = f.id_four;
```

	id_borne integer	fournisseur character varying (20)
1	1	Nintendo
2	2	Capcom
3	3	Sega
4	4	Konami
5	5	Atari

Figure 9: Résultat SQL - Les fournisseurs des bornes

Combien de recharges chaque client a-t-il fait?

```
SELECT c.pseudo, COUNT(r.id_recharge) AS nb_recharges
FROM Client c
LEFT JOIN Carte ca ON c.id_client = ca.id_client
LEFT JOIN Recharge r ON ca.id_carte = r.id_carte
GROUP BY c.pseudo;
```

	pseudo character varying (25)	nb_recharges bigint
1	Caroline88	0
2	David92	2
3	Alice95	1
4	Emma97	1
5	Bob90	1

Figure 10: Résultat SQL - Les recharges par clients

Quand chaque client a-t-il joué pour la dernière fois?

```
SELECT c.pseudo, MAX(p.date_partie) AS derniere_partie
FROM Client c
LEFT JOIN Partie p ON c.id_client = p.id_client
GROUP BY c.pseudo;
```

	pseudo character varying (25)	derniere_partie date
1	Caroline88	2023-07-04
2	David92	2025-11-09
3	Alice95	2023-07-03
4	Emma97	2023-07-02
5	Bob90	[null]

Figure 11: Résultat SQL - Date de dernière partie

Quelle est le classement pour chaque jeu ?

```
SELECT Jeux.nom AS jeu, Client.pseudo, MAX(Partie.score) AS meilleur_score
FROM Partie
JOIN Jeux ON Partie.id_jeux = Jeux.id_jeux
JOIN Client ON Partie.id_client = Client.id_client
WHERE Partie.score IS NOT NULL
GROUP BY Jeux.nom, Client.pseudo
ORDER BY Jeux.nom, meilleur_score DESC;
```

	jeu character varying (50)	pseudo character varying (25)	meilleur_score integer
1	Pacman	Emma97	100
2	Pacman	Alice95	50
3	Super Mario	Emma97	85000
4	Zelda	Caroline88	120

Figure 12: Résultat SQL - Classement par jeux

Combien de récompenses chaque joueur a-t-il reçues?

```
SELECT Client.pseudo, SUM(Partie.recompense) AS total_recompenses
FROM Client
LEFT JOIN Partie ON Client.id_client = Partie.id_client
GROUP BY Client.pseudo
ORDER BY total_recompenses DESC;
```

	pseudo character varying (25)	total_recompenses bigint
1	Bob90	[null]
2	Caroline88	5
3	Emma97	2
4	Alice95	1
5	David92	0

Figure 13: Résultat SQL - Total des récompenses

Réseau

Données échangées via le réseau

Tous les échanges entre la borne d'arcade (client) et le serveur reposent sur le protocole TCP, garantissant une communication fiable et ordonnée. Chaque échange suit un format structuré avec des commandes claires et des paramètres précis.

Les diagrammes applicatifs utilisent une représentation visuelle standardisée : les flèches bleues représentent les messages envoyés du client vers le serveur, tandis que les flèches rouges représentent les réponses du serveur vers le client. Les encadrés à droite du serveur expliquent le traitement effectué pour chaque requête reçue, et les post-its jaunes décrivent l'objectif global du protocole illustré.

Les commandes envoyées par le client respectent le format suivant :

- Nom de la commande : toujours en MAJUSCULES (ex: PSEUDO, BALANCE, START_GAME)
- Paramètres : identifiants numériques ou chaînes de caractères séparés par des espaces
  - <id_carte>, <id_jeux>, <id_borne> : identifiants d'entités
  - <score>, <montant_jeton> : valeurs numériques

Note importante : Les chevrons (< >) présents dans les diagrammes ne font pas partie du protocole réel. Ils servent uniquement à la lisibilité de la documentation pour indiquer qu'il faut remplacer <id_carte> par une valeur concrète (ex: 1, 42, etc.).

Chaque réponse du serveur suit une structure uniforme : `OK <données>` en cas de succès ou `ERROR <code_erreur>` en cas d'échec. Toutes les réponses se terminent par un caractère de retour à la ligne (`\n`). Lorsque le serveur renvoie plusieurs données, elles sont séparées par des virgules suivies d'un espace (par exemple : `OK 1 Emma97 100, 2 Alice95 50, 3 Bob90 30`).

Chaque protocole nécessite une nouvelle connexion TCP. Le client se connecte, exécute le protocole complet (une ou plusieurs commandes liées), puis se déconnecte. En cas d'erreur, le serveur ferme immédiatement la connexion, empêchant ainsi les états incohérents. Cette approche garantit la cohérence des échanges à chaque interaction.

Diagrammes applicatifs

Échanges avant la partie

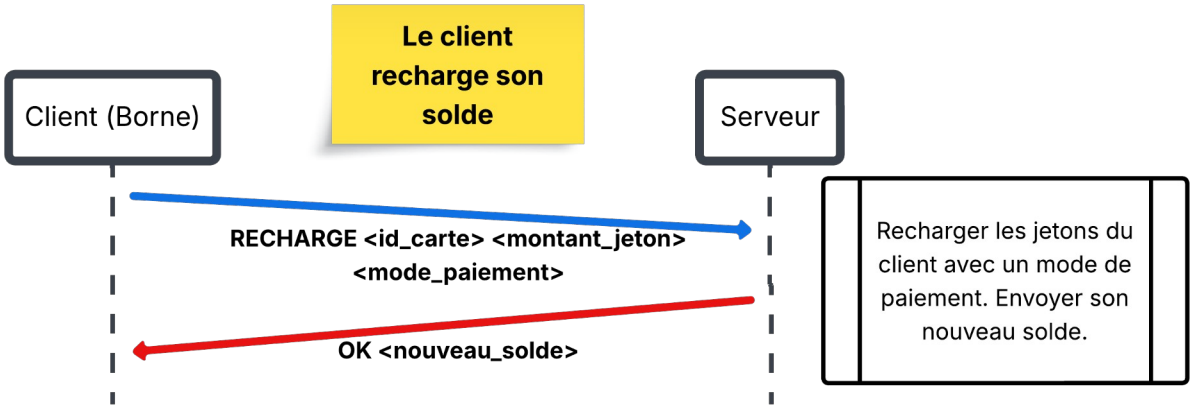

Figure 14: Échange réseau : Recharger son solde

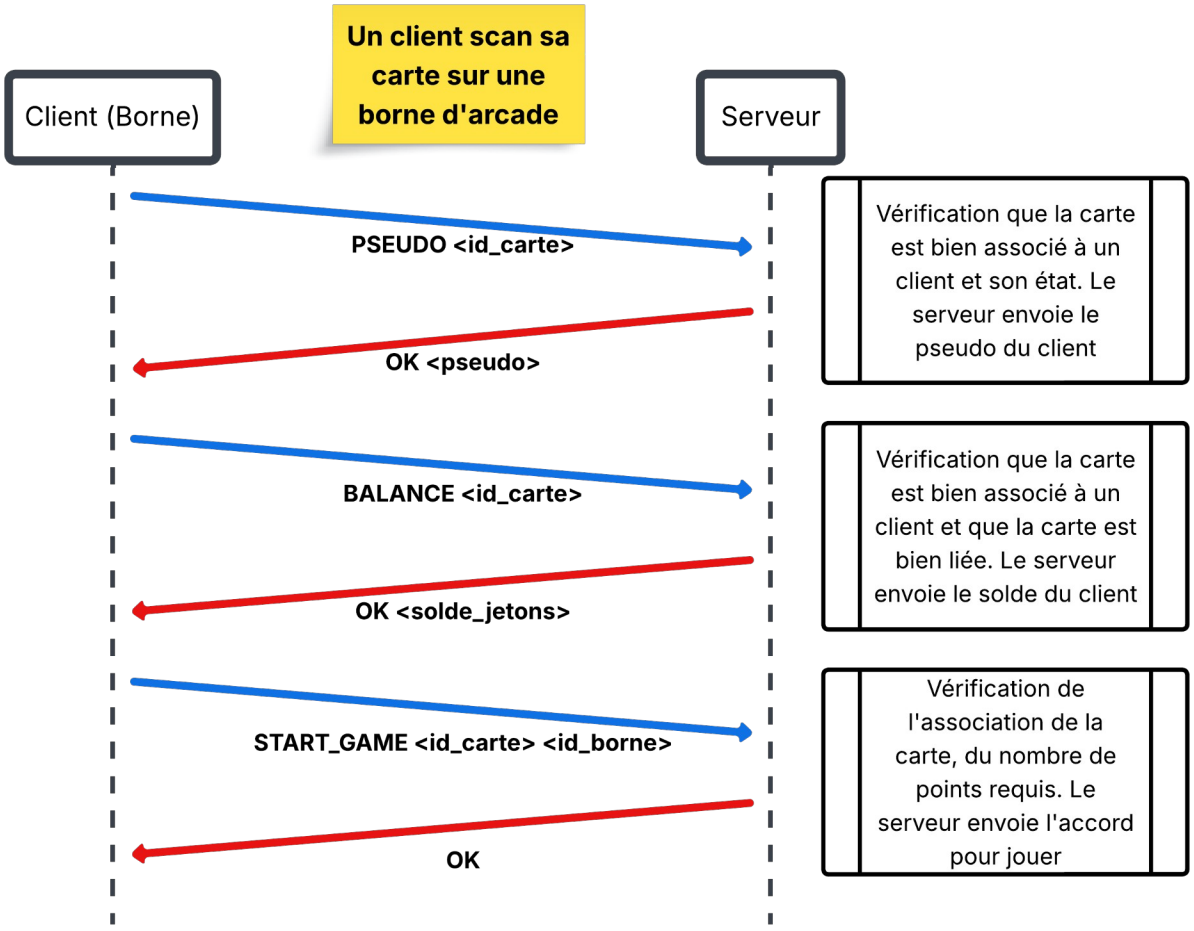

Figure 15: Échange réseau : Lancement de la partie

Échanges après la partie

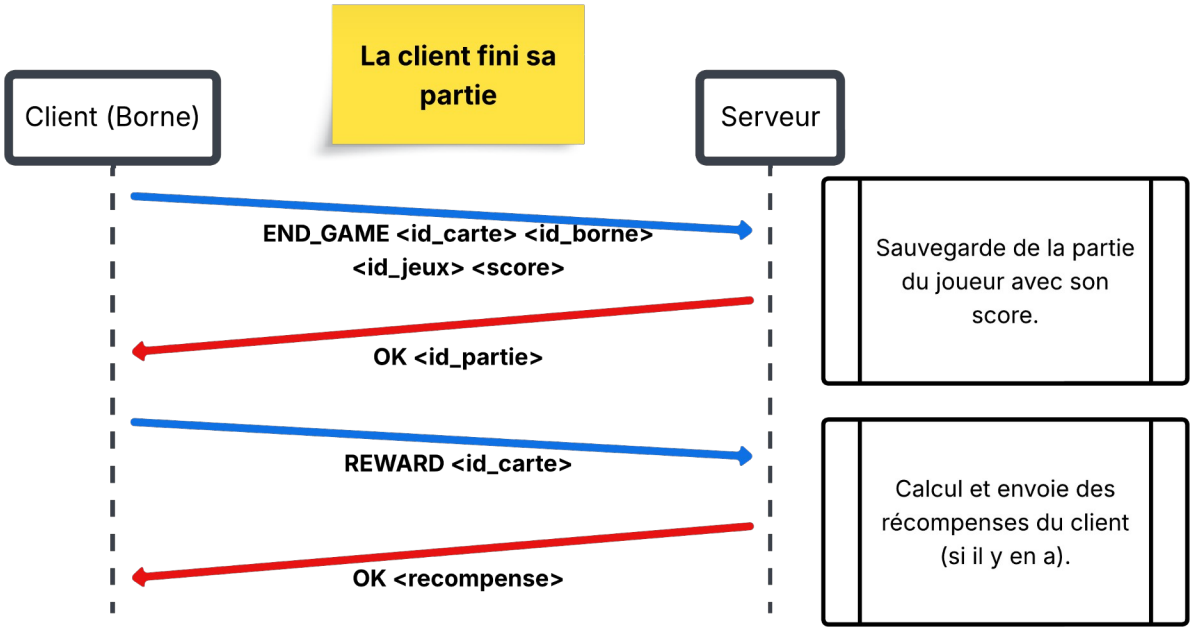

Figure 16: Échange réseau : Fin de la partie

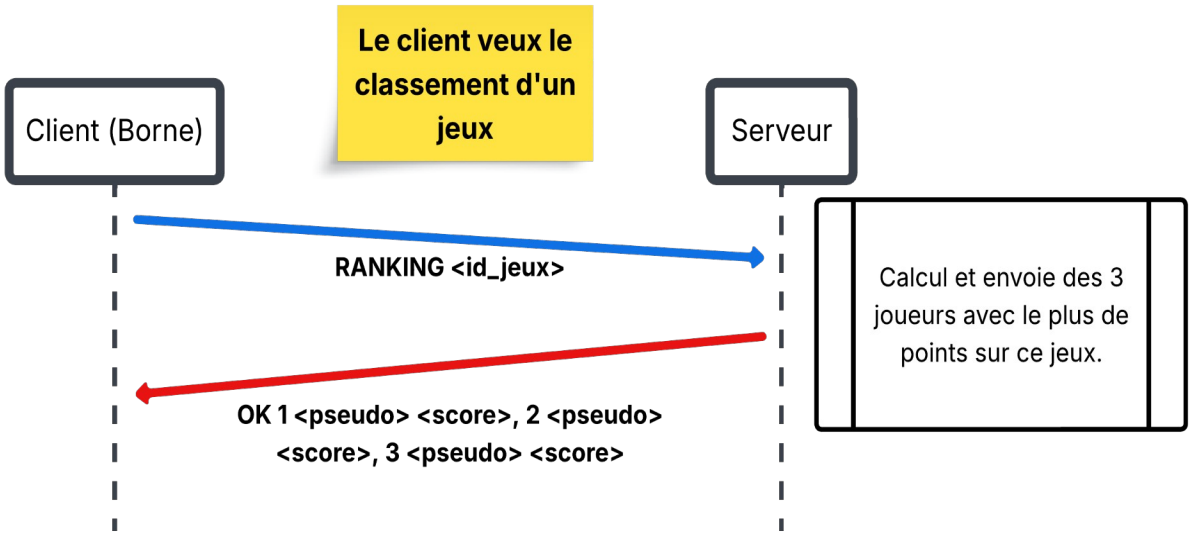

Figure 17: Échange réseau : Classement du jeu

Tests et validation

Les tests suivants valident l'implémentation du protocole TCP décrit précédemment. Le code source est disponible en annexe.

Test de démarrage du serveur

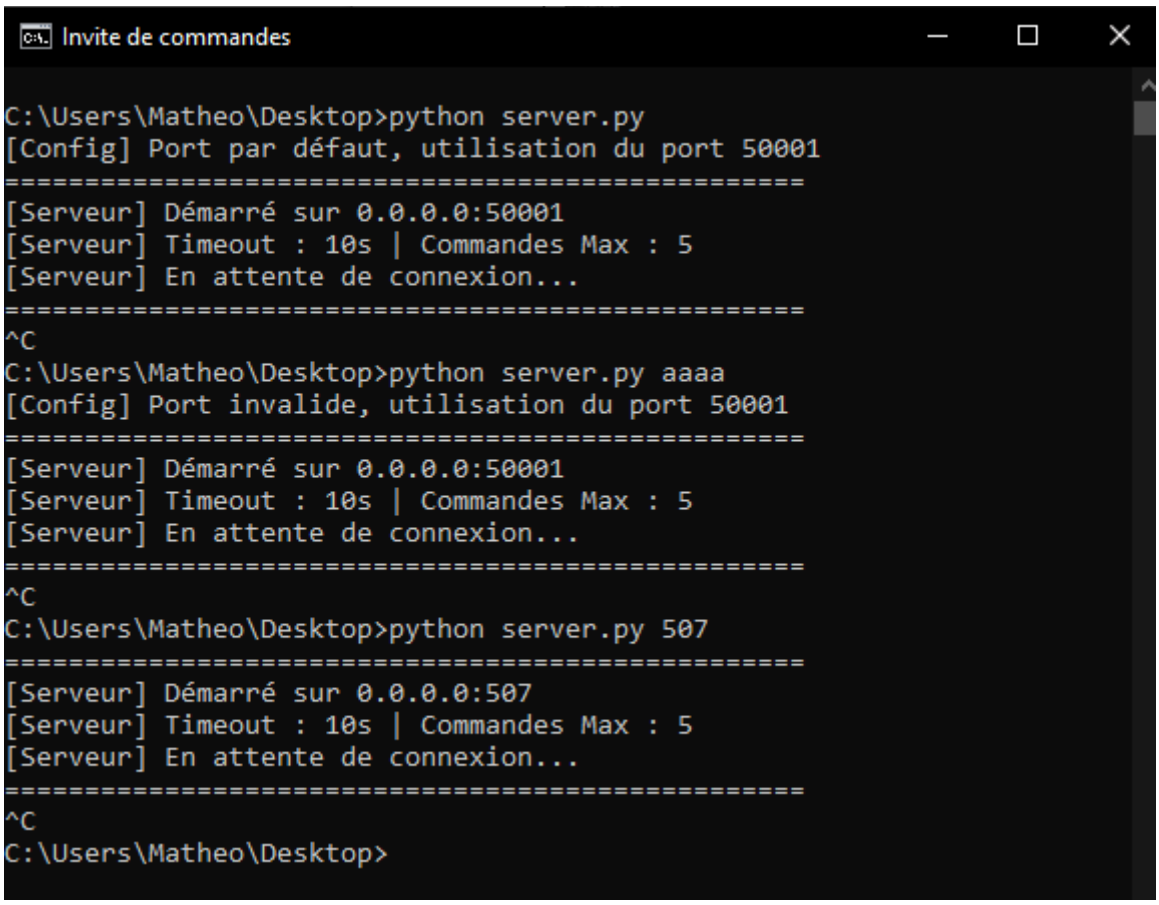

Figure 18: Test serveur : Configuration du numéro de port

Dans la Figure 18, nous observons trois tests de démarrage du serveur Python avec différents paramètres de port.

Le serveur accepte un paramètre optionnel lors de son lancement : le numéro de port. Si aucun port n'est spécifié, le serveur utilise le port 50001 par défaut, situé dans la plage dynamique des ports (49152-65535) non attribués par l'IANA.

Premier test : Le serveur démarre sans paramètre et utilise correctement le port par défaut 50001.

Deuxième test : Un paramètre invalide (aaaa) est passé en argument. Le serveur détecte l'erreur, affiche un message d'avertissement et démarre sur le port par défaut 50001
Troisième test : Le serveur démarre avec le port 507 spécifié en argument. Le démarrage s'effectue correctement sur ce port.
Au démarrage, le serveur affiche plusieurs informations de configuration :

- L'adresse IP et le numéro de port d'écoute
- Le timeout par client : 10 secondes d'inactivité maximum
- La limite de commandes : 5 commandes maximum par connexion
- L'état du serveur : "En attente de connexion…"

Test de communication client-serveur

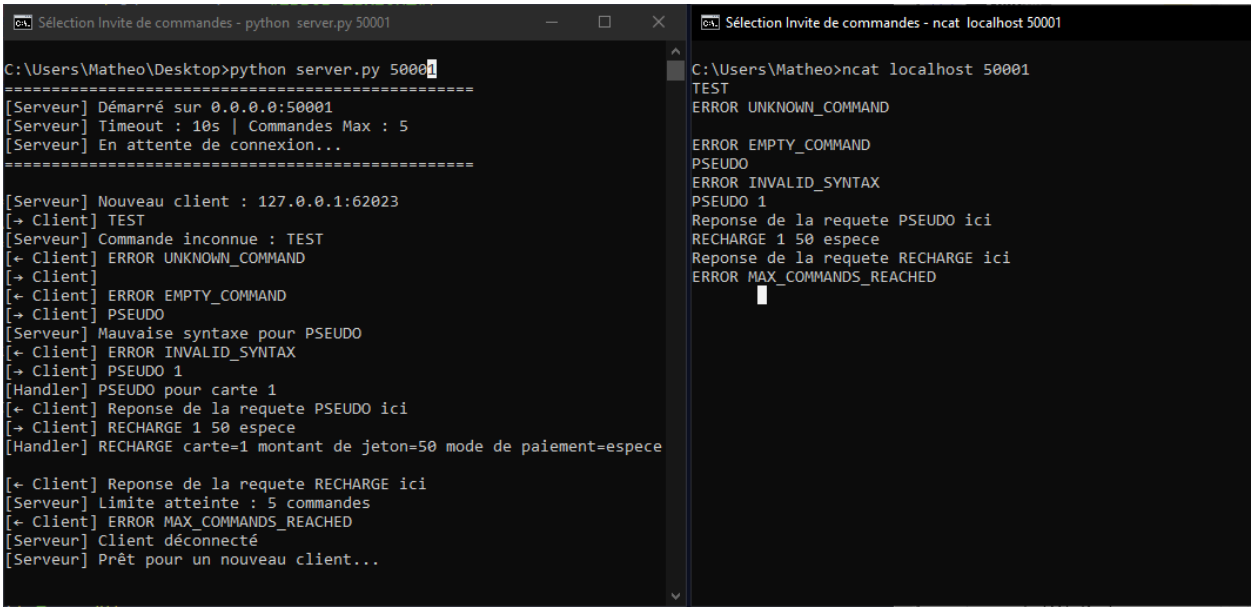

Figure 19: Test serveur : Commandes prise en charge

Dans la Figure 19, nous testons la communication entre le serveur (fenêtre de gauche) et un client simulé avec ncat (fenêtre de droite). Le client se connecte en localhost sur le port 50001.

- Test 1 - Commande inconnue : Le client envoie TEST. Le serveur répond ERROR UNKNOWN_COMMAND car cette commande n'existe pas dans le protocole.
- Test 2 - Commande vide : Le client envoie uniquement un retour à la ligne (\n). Le serveur répond ERROR EMPTY_COMMAND.
- Test 3 - Syntaxe invalide : Le client envoie PSEUDO sans argument. Le serveur répond ERROR INVALID_SYNTAX car il manque l'ID de carte requis.

- Test 4 - Commande valide : Le client envoie PSEUDO 1 avec la syntaxe correcte. Le serveur traite la commande et répond "Réponse de la requête PSEUDO ici" (message temporaire car la base de données n'est pas encore configurée).
- Test 5 - Autre commande valide : Le client envoie RECHARGE 1 50 espece. Le serveur valide la syntaxe et répond "Réponse de la requête RECHARGE ici".
- Test 6 - Limite atteinte : Après 5 commandes, le serveur répond ERROR MAX_COMMANDS_REACHED et ferme la connexion proprement. Il se remet ensuite en attente d'un nouveau client.

Test de connexion BD

Pour se connecter à la base de données PostgreSQL, nous utilisons la bibliothèque psycopg2 en Python. La fonction get_db_connection(), voir utils.py - Connexion à la bd, établit la connexion en utilisant les paramètres de configuration (hôte, port, nom de la base, utilisateur et mot de passe).

Pour valider que la connexion fonctionne correctement, j'ai testé une requête complète avec le protocole RECHARGE. Cette fonction effectue plusieurs opérations sur la base de données :

Lecture : Récupération de l'id_client via une jointure entre Carte et Client

Insertion : Enregistrement de la recharge dans la table Recharge

Mise à jour : Modification du solde dans la table Client

Pour tester, nous allons tenter de recharger la carte 1 avec 50 jetons avec le mode de paiement par CB.

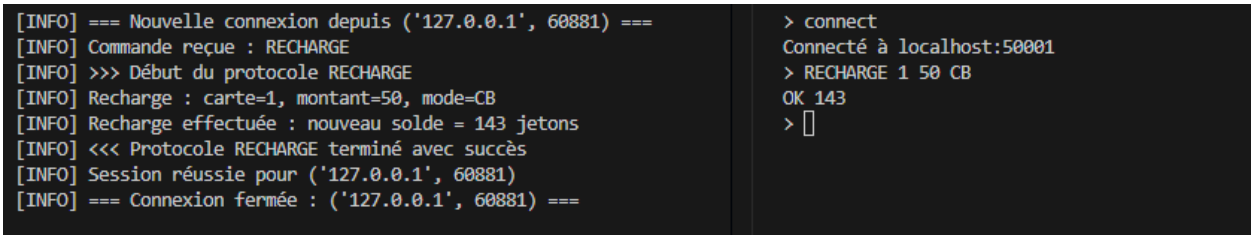

Figure 20: Test serveur : Connexion à la BD

La connexion est fonctionnelle, toutes les opérations (SELECT, INSERT, UPDATE) se sont exécutées sans erreur.

Test du client Java

Nous testons maintenant le client Java que nous avons développé. Le client se connecte au serveur sur le port 8080 et permet d'envoyer des commandes de manière interactive.
Interface du client : Au lancement, le client affiche un message de connexion et invite l'utilisateur à saisir des commandes. Le format attendu est indiqué : PSEUDO 1, BALANCE 1, START_GAME 1 2 3, etc. La commande QUIT permet de fermer proprement la connexion.

Test 1 - Commandes invalides

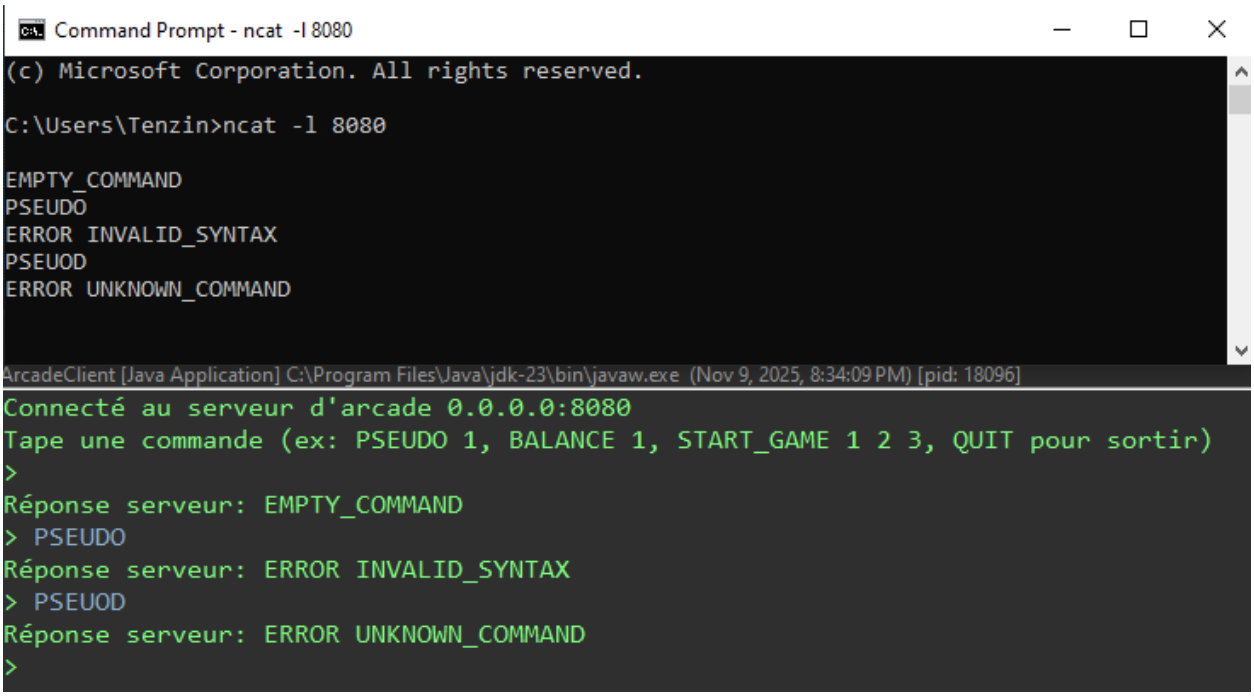

Figure 21: Test client : Commandes invalides

Dans la Figure 21, nous testons différentes commandes invalides pour vérifier la robustesse du protocole.
Commande vide : L'utilisateur appuie sur Entrée sans saisir de texte. Le serveur répond EMPTY_COMMAND.
Commande incomplète : L'utilisateur tape PSEUDO sans l'ID de carte. Le serveur détecte l'erreur de syntaxe et répond ERROR INVALID_SYNTAX.
Deuxième tentative incomplète : L'utilisateur retape PSEUDO seul. Le serveur renvoie la même erreur : ERROR INVALID_SYNTAX.
Commande inconnue : L'utilisateur tape PSEUOD (faute de frappe). Le serveur ne reconnaît pas la commande et répond ERROR UNKNOWN_COMMAND.

Le client affiche clairement les réponses du serveur précédées de Réponse serveur:, permettant une distinction nette entre les commandes envoyées et les réponses reçues.

Test 2 - Commandes valides

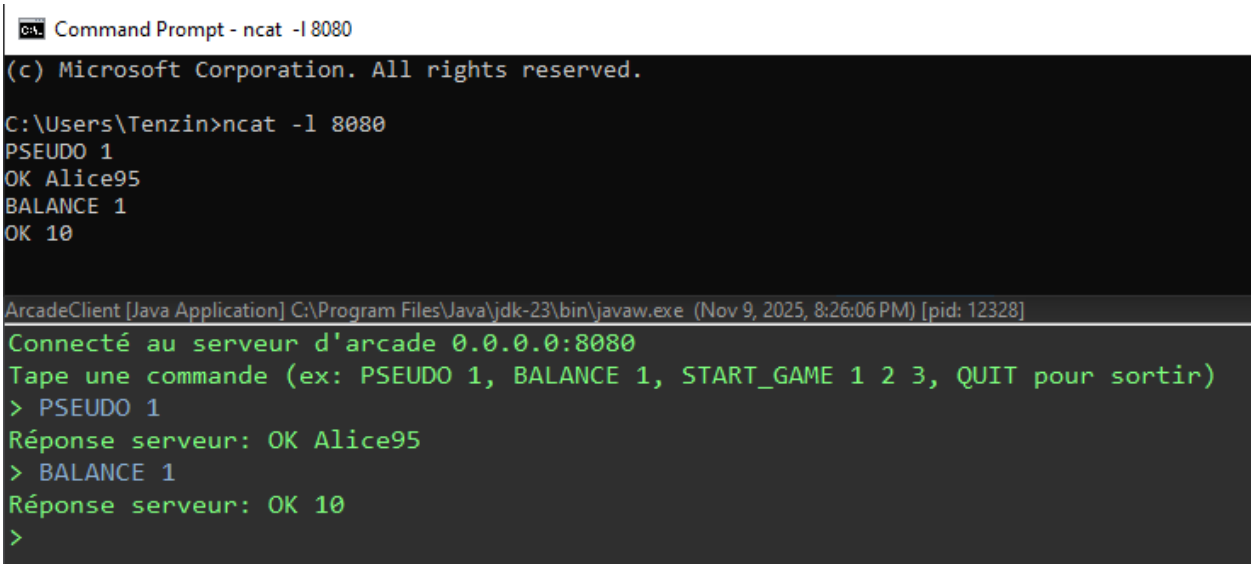

Figure 22: Test client : Commandes valides

Dans la Figure 22, nous testons des commandes correctes avec des données réelles issues de la base de données.

PSEUDO 1 : Le client envoie une demande de récupération du pseudo pour la carte ID 1. Le serveur interroge la base de données et répond OK Alice95, confirmant que le pseudo associé à cette carte est "Alice95".

BALANCE 1 : Le client demande le solde de jetons pour la carte ID 1. Le serveur répond OK 10, indiquant que le compte possède 10 jetons disponibles.

Ces tests démontrent que :

- Le client Java communique correctement avec le serveur Python via TCP
- Les réponses du serveur sont bien reçues et affichées
- La validation des commandes fonctionne côté serveur
- L'intégration avec la base de données PostgreSQL est opérationnelle
- L'interface utilisateur est claire et guidée

Site Web

Pour le site web du projet, nous proposons une connexion pour les clients ainsi que pour les administrateurs de la salle d'arcade. Les utilisateurs doivent se connecter au site avec leur login et mot de passe respectifs. Ensuite, chacun dispose de fonctionnalités différentes selon son rôle.

Client

Les clients en se connectant peuvent accéder à leur profile et visionner :

- Leur pseudo
- Le solde de leur compte
- La version de leur carte
- Le nombre de parties jouées
- La date de la dernière partie
- Leur meilleur score toutes catégories de jeux confondues
- Le jeu le plus joué

Ils ont également la possibilité de recharger leur compte en ligne en choisissant un montant.

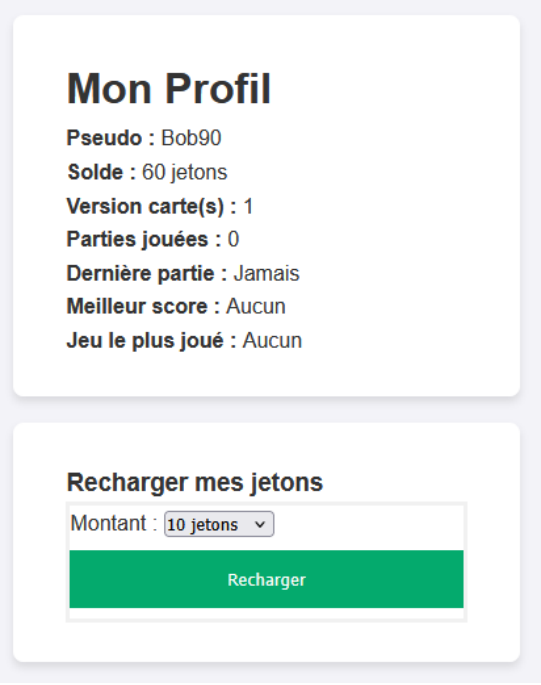

Figure 23: Capture web - Client

Administrateur

Les administrateurs disposent tous de la même interface, où ils peuvent consulter les informations globales de la salle d'arcade :

- Le nombre de clients inscrits
- Les bornes installés
- Le nombre total de parties jouées
- Le nombre de cartes distribuées

Ils peuvent également gérer les bornes et les jeux de la salle. Ils ont la possibilité de modifier l'état de chaque borne et de chaque jeu, en les mettant en « disponible », « hors service » ou « en maintenance ».

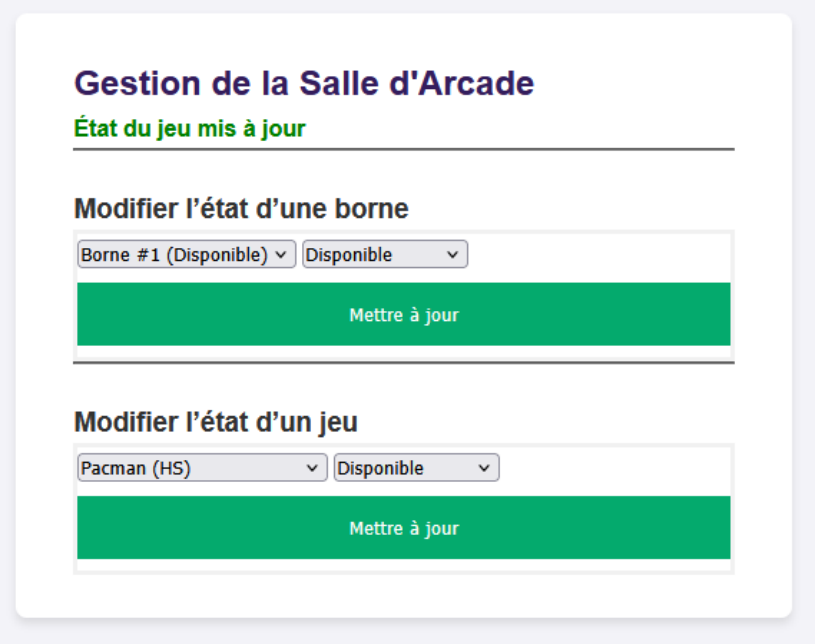

Figure 24: Capture web - Administrateur

Annexes

Code source

Le code source complet de notre solution est disponible, comme demandé, dans un fichier à part contenant tous les codes du projet. Cette annexe présente simplement les extraits les plus significatifs permettant de comprendre l'architecture et le fonctionnement de notre système.

Nous avons sélectionné pour la communication client-serveur :

- Serveur Python : la boucle principale, la connexion à la base de donnée, le protocole START et la requête START_GAME
- Client Java : la gestions des commandes et 2 fonctions de communication serveur.

Ces extraits illustrent nos choix techniques : gestion d'erreurs par exceptions, validation des données, logging complet, et séparation protocoles/requêtes/utilitaires.Et les autres fonctions suivent une structure semblable.

Pour la partie SQL de notre base de données, l'intégralité du contenu y est inclus.

Serveur (Python)

server.py - Fonction principale

```
def run_server():
    """Lance le serveur principal"""
    logging.info("=" * 50)
    logging.info("Démarrage du serveur Cy_Arcade")
    logging.info("=" * 50)

    # Connexion à la BD
    db_conn = get_db_connection(config)
    if db_conn is None:
        logging.critical("Impossible de démarrer sans connexion BD")
        return

    # Création du socket serveur
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    try:
        server.bind((SERVER_IP, SERVER_PORT))
```

```
    except OSError as e:
        logging.critical(f"Port {SERVER_PORT} déjà utilisé : {e}")
        return

    server.listen(0)

    logging.info(f"Serveur en écoute sur {SERVER_IP}:{SERVER_PORT}")
    logging.info("En attente de connexions...\n")

    try:
        while True:
            # Accepter une nouvelle connexion
            client_socket, client_address = server.accept()
            handle_client(client_socket, client_address, db_conn)

    except Exception as e:
        logging.critical(f"Erreur fatale : {e}")
    finally:
        db_conn.close()
        server.close()
        logging.info("Serveur arrêté")
```

utils.py - Connexion à la bd

```
def get_db_connection(config):
    """Connexion à la base de données"""
    try:
        db_connection = psycopg2.connect(
            host=config['DB_HOST'],
            port=config['DB_PORT'],
            database=config['DB_NAME'],
            user=config['DB_USER'],
            password=config['DB_PASSWORD']
        )
        logging.info("Connexion à la BD réussie")
        return db_connection
    except Exception as e:
        logging.critical(f"Erreur de connexion à la BD : {e}")
        return None
```

protocols.py - Protocole START

```
def protocol_start(client_socket, db_conn, first_command):
    """Protocole START : PSEUDO → BALANCE → START_GAME"""
    logging.info("Début du protocole START")
```

```
    try:
        # Étape 1 : PSEUDO
        parts = parse_command(first_command, "PSEUDO", 2)
        id_carte = parts[1]
        id_carte_int = validate_id(id_carte, "CARD_ID")

        pseudo = query_pseudo(db_conn, id_carte)
        send_response(client_socket, f"OK {pseudo}")
        # Étape 2 : BALANCE
        parts = expect_command(client_socket, "BALANCE", 2)
        verify_card_id(parts, id_carte)

        balance = query_balance(db_conn, id_carte)
        send_response(client_socket, f"OK {balance}")
        # Étape 3 : START_GAME
        parts = expect_command(client_socket, "START_GAME", 4)
        verify_card_id(parts, id_carte)

        id_borne = parts[2]
        id_jeux = parts[3]

        id_borne_int = validate_id(id_borne, "BORNE_ID")
        id_jeux_int = validate_id(id_jeux, "GAME_ID")

        query_start_game(db_conn, id_carte, id_borne, id_jeux)
        send_response(client_socket, "OK")

        logging.info("Protocole START terminé avec succès")
        return "SUCCESS"

    except ValueError as e:
        return str(e)
    except Exception as e:
        logging.error(f"Erreur inattendue: {e}")
        return "ERROR DATABASE_ERROR"
```

queries.py - Requête START_GAME

```
def query_start_game(db_conn, id_carte, id_borne, id_jeux):
    """Lance une partie"""
    logging.info(f"Démarrage partie : carte={id_carte}, borne={id_borne},
jeu={id_jeux}")
```

```
    cursor = db_conn.cursor()

    # Récupérer id_client et solde
    query = """
    SELECT Client.id_client, Client.solde_jetons
    FROM Carte
    JOIN Client ON Carte.id_client = Client.id_client
    WHERE Carte.id_carte = %s AND Carte.status_carte = 'Liee'
    """
    cursor.execute(query, (id_carte,))
    client_result = cursor.fetchone()
    if client_result is None:
        cursor.close()
        logging.warning(f"Carte {id_carte} introuvable")
        raise ValueError("ERROR CARD_NOT_FOUND")

    id_client, solde_jetons = client_result

    # Vérifier que le jeu existe et est disponible
    query = "SELECT Jeux.etat_jeu FROM Jeux WHERE Jeux.id_jeux = %s"
    cursor.execute(query, (id_jeux,))
    jeu_result = cursor.fetchone()
    if jeu_result is None:
        cursor.close()
        logging.warning(f"Jeu {id_jeux} introuvable")
        raise ValueError("ERROR GAME_NOT_FOUND")

    if jeu_result[0] != 'Disponible':
        cursor.close()
        logging.warning(f"Jeu {id_jeux} indisponible")
        raise ValueError("ERROR GAME_UNAVAILABLE")

    # Vérifier la borne
    query = "SELECT Borne.etat_borne, Borne.prix_jeton FROM Borne WHERE
Borne.id_borne = %s"
    cursor.execute(query, (id_borne,))
    borne_result = cursor.fetchone()
    if borne_result is None:
        cursor.close()
        logging.warning(f"Borne {id_borne} introuvable")
        raise ValueError("ERROR ARCADE_NOT_FOUND")

    if borne_result[0] != 'Disponible':
        cursor.close()
```

```
        logging.warning(f"Borne {id_borne} indisponible")
        raise ValueError("ERROR ARCADE_UNAVAILABLE")

    prix_jeton = borne_result[1]

    # Vérifier le solde
    if solde_jetons < prix_jeton:
        cursor.close()
        logging.warning(f"Solde insuffisant : {solde_jetons} < {prix_jeton}")
        raise ValueError("ERROR INSUFFICIENT_BALANCE")

    # Débiter le compte
    query = "UPDATE Client SET solde_jetons = solde_jetons - %s WHERE
id_client = %s"
    cursor.execute(query, (prix_jeton, id_client))

    # Créer l'entrée de partie
    query = """
    INSERT INTO Partie (id_client, id_jeux, id_borne, status_partie,
date_partie)
    VALUES (%s, %s, %s, 'En cours', CURRENT_DATE)
    """
    cursor.execute(query, (id_client, id_jeux, id_borne))

    db_conn.commit()
    cursor.close()
    logging.info(f"Partie démarrée avec succès ({prix_jeton} jetons
débités)")
```

Client (Java)

ArcadeClient.java - Gestion des commandes

```
    /**
     * Gère les commandes utilisateur.
     * @param input La commande entrée par l'utilisateur
     */
    private static void handleCommand(String input) {
        String[] parts = input.split(" ");
        String cmd = parts[0].toLowerCase();
        switch (cmd) {
            case "connect":
                connect(config.getHost(), config.getPort());
                break;
```

```
            case "disconnect":
                disconnect();
                break;
            case "help":
                help();
                break;
            default:
                if (!isValidCommand(input)){
                    System.out.println("La commande ne respecte pas la forme:
COMMAND <parametres>.");
                    break;
                }
                if (sendCommand(input)) {
                    read(1);
                }
        }
    }
```

SocketConnexion.java - Envoi de commandes et lecture des réponses.

```
/**
     * Envoie une commande au serveur.
     * @param command La commande à envoyer
     * @throws IOException Si une erreur d'E/S se produit
     */
    public void sendCommand(String command) throws IOException {
        if (socket == null || socket.isClosed()) {
            throw new IOException("Non connecté au serveur");
        }
        out.println(command);
        log("ENVOI: " + command);
    }

    /**
     * Lit une ligne de la réponse du serveur.
     * @return String La ligne lue
     * @throws IOException Si une erreur d'E/S se produit
     */
    public String readLine() throws IOException {
        if (socket == null || socket.isClosed()) {
            throw new IOException("Non connecté au serveur");
        }
        String line = in.readLine();
        if (line == null) {
```

```
            throw new IOException("Connexion fermée par le serveur");
        }
        log("REÇU: " + line);
        return line;
    }
```

Serveur Web (PHP)

gestion_systeme.php - Gestion des bornes par les admins

```
<?php
require_once __DIR__ . '/config/config.php';

if (!isset($_SESSION['role']) || $_SESSION['role'] !== 'admin') {
    header("Location: index.php");
    exit();
}

$message = "";

// Update borne
if (isset($_POST['update_borne'])) {
    $id_borne = $_POST['id_borne'];
    $etat = $_POST['etat_borne'];

    $q = "UPDATE Borne SET etat_borne = $1 WHERE id_borne = $2";
    if (pg_query_params($db, $q, array($etat, $id_borne))) {
        $message = "<p style='color:green;font-weight:bold;'> État de la
borne mis à jour</p>";
    } else {
        $message = "<p style='color:red;'> Erreur lors de la mise à
jour</p>";
    }
}

// Update jeu
if (isset($_POST['update_jeu'])) {
    $id_jeu = $_POST['id_jeu'];
    $etat = $_POST['etat_jeu'];

    $q = "UPDATE Jeux SET etat_jeu = $1 WHERE id_jeux = $2";
    if (pg_query_params($db, $q, array($etat, $id_jeu))) {
        $message = "<p style='color:green;font-weight:bold;'> État du jeu mis
à jour</p>";
```

```
    } else {
        $message = "<p style='color:red;'> Erreur lors de la mise à
jour</p>";
    }
}

$bornes = pg_query($db, "SELECT id_borne, etat_borne FROM Borne ORDER BY
id_borne ASC");
$jeux = pg_query($db, "SELECT id_jeux, nom, etat_jeu FROM Jeux ORDER BY
id_jeux ASC");
?>
```

profile.php - Profil du client

```
<?php
require_once __DIR__ . '/config/config.php';

if (!isset($_SESSION['role']) || $_SESSION['role'] !== 'client') {
    header("Location: index.php");
    exit();
}

$id = $_SESSION['id_client'];

//Infos client
$user_q = "SELECT pseudo, solde_jetons FROM Client WHERE id_client = $1";
$user = pg_fetch_assoc(pg_query_params($db, $user_q, array($id)));

//Cartes
$cards_q = "SELECT version_carte FROM Carte WHERE id_client = $1";
$cards_res = pg_query_params($db, $cards_q, array($id));
$versions = [];
while ($row = pg_fetch_assoc($cards_res)) {
    $versions[] = $row['version_carte'];
}

//Parties + dernière partie
$games_q = "
    SELECT COUNT(*) AS nb_parties, MAX(date_partie) AS derniere
    FROM Partie WHERE id_client = $1
";
$games = pg_fetch_assoc(pg_query_params($db, $games_q, array($id)));

// Jeu le plus joué
```

```
$q_top = "
    SELECT j.nom, COUNT(*) AS nb
    FROM Partie p
    JOIN Jeux j ON p.id_jeux = j.id_jeux
    WHERE p.id_client = $1
    GROUP BY j.nom
    ORDER BY nb DESC
    LIMIT 1
";
$r_top = pg_query_params($db, $q_top, array($id));
$top = pg_fetch_assoc($r_top);

// Meilleur score
$q_best = "SELECT MAX(score) AS best FROM Partie WHERE id_client = $1";
$r_best = pg_query_params($db, $q_best, array($id));
$best = pg_fetch_assoc($r_best);
?>
```

Scripts SQL

DDL

```
-- Table : Personne
CREATE TABLE Personne (
      id_perso SERIAL PRIMARY KEY,
      nom VARCHAR(25) NOT NULL,
      prenom VARCHAR(25) NOT NULL,
      sexe VARCHAR(10) NOT NULL CHECK (sexe IN ('homme', 'femme')),
      date_naissance DATE NOT NULL,
      mail VARCHAR(40) NOT NULL UNIQUE,
      tel CHAR(10) UNIQUE,
      login VARCHAR(25) NOT NULL UNIQUE,
      mot_de_passe VARCHAR(255) NOT NULL
);
```

```
-- Table : Client
CREATE TABLE Client(
      id_client INT PRIMARY KEY REFERENCES Personne(id_perso)
        ON DELETE CASCADE ON UPDATE CASCADE,
      pseudo VARCHAR(25) NOT NULL UNIQUE,
      solde_jetons INT NOT NULL DEFAULT 0 CHECK (solde_jetons >= 0),
      date_inscription DATE NOT NULL DEFAULT (CURRENT_DATE)
);

-- Table : Admin
CREATE TABLE Admin(
      id_admin INT PRIMARY KEY REFERENCES Personne(id_perso)
        ON DELETE CASCADE ON UPDATE CASCADE,
      date_anciennete_employe DATE NOT NULL DEFAULT (CURRENT_DATE),
      poste_employe VARCHAR(10) NOT NULL CHECK (poste_employe IN ('gerant',
'nettoyage', 'personnel')),
      status_employe VARCHAR(10) NOT NULL DEFAULT 'travail' CHECK
(status_employe IN ('travail', 'en_pause', 'repos'))
);

-- Table : Carte
CREATE TABLE Carte(
      id_carte SERIAL PRIMARY KEY,
      status_carte VARCHAR(10) NOT NULL DEFAULT 'Liee' CHECK (status_carte IN
('Perdu', 'Liee')),
      date_carte DATE NOT NULL DEFAULT (CURRENT_DATE),
      version_carte INT NOT NULL CHECK (version_carte > 0),
      id_client INT NOT NULL REFERENCES Client(id_client)
        ON DELETE CASCADE ON UPDATE CASCADE
);

-- Table : Recharge
CREATE TABLE Recharge(
      id_recharge SERIAL PRIMARY KEY,
      montant_jeton INT NOT NULL CHECK (montant_jeton >0),
      mode_paiement VARCHAR(10) NOT NULL CHECK (mode_paiement IN ('CB',
'espece')),
      date_recharge DATE NOT NULL,
      id_carte INT NOT NULL REFERENCES Carte(id_carte)
            ON DELETE CASCADE ON UPDATE CASCADE
);
```

```
-- Table : Fournisseur
CREATE TABLE Fournisseur(
      id_four SERIAL PRIMARY KEY,
      nom_four VARCHAR(20) NOT NULL,
      mail_four VARCHAR(40) NOT NULL UNIQUE,
      tel_four CHAR(10) NOT NULL UNIQUE,
      adresse_four VARCHAR(30) NOT NULL
);

-- Table : Jeux
CREATE TABLE Jeux(
      id_jeux SERIAL PRIMARY KEY,
      nom VARCHAR(50) NOT NULL UNIQUE,
      etat_jeu VARCHAR(15) NOT NULL DEFAULT 'Disponible' CHECK (etat_jeu IN
('Disponible', 'En Maintenance', 'HS'))
);

-- Table : Jeux_Solo
CREATE TABLE Jeux_Solo(
      id_jeux_solo INT PRIMARY KEY REFERENCES Jeux(id_jeux)
        ON DELETE CASCADE ON UPDATE CASCADE,
      difficulte VARCHAR(10) NOT NULL CHECK (difficulte IN ('Facile',
'Normal', 'Difficile'))
);

-- Table : Jeux_Multi
CREATE TABLE Jeux_Multi(
      id_jeux_multi INT PRIMARY KEY REFERENCES Jeux(id_jeux)
        ON DELETE CASCADE ON UPDATE CASCADE,
      type_equipe VARCHAR(4) NOT NULL CHECK (type_equipe IN ('Coop', 'vs')),
      nb_joueur_min INT NOT NULL,
      nb_joueur_max INT NOT NULL,
      CHECK (nb_joueur_min >= 2 AND nb_joueur_max <= 4 AND nb_joueur_min <=
nb_joueur_max)
);
```

```
-- Table : Borne
CREATE TABLE Borne(
      id_borne SERIAL PRIMARY KEY,
      etat_borne VARCHAR(15) NOT NULL DEFAULT 'Disponible' CHECK (etat_borne
IN ('Disponible', 'En Maintenance', 'HS')),
      prix_jeton INT NOT NULL CHECK (prix_jeton > 0),
      date_achat DATE NOT NULL DEFAULT (CURRENT_DATE),
      id_four INT NOT NULL REFERENCES Fournisseur(id_four)
        ON DELETE CASCADE ON UPDATE CASCADE
);

-- Table : Jeux_Borne
CREATE TABLE Jeux_Borne (
    id_jeux INT NOT NULL REFERENCES Jeux(id_jeux)
        ON DELETE CASCADE ON UPDATE CASCADE,
    id_borne INT NOT NULL REFERENCES Borne(id_borne)
        ON DELETE CASCADE ON UPDATE CASCADE,
    PRIMARY KEY (id_jeux, id_borne)
);

-- Table : Partie
CREATE TABLE Partie (
      id_partie SERIAL PRIMARY KEY,
      score INT CHECK (score >= 0),
      date_partie DATE,
      recompense INT NOT NULL DEFAULT 0 CHECK (recompense IN (5, 2, 1, 0)),
      status_partie VARCHAR(20) NOT NULL DEFAULT 'En cours' CHECK
(status_partie IN ('En cours', 'Termine', 'Recompense donnee')),
      id_client INT NOT NULL REFERENCES Client(id_client),
      id_jeux INT NOT NULL REFERENCES Jeux(id_jeux)
        ON DELETE CASCADE ON UPDATE CASCADE,
      id_borne INT NOT NULL REFERENCES Borne(id_borne)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

DML

```
-- Insertion dans Personne
INSERT INTO Personne (nom, prenom, sexe, date_naissance, mail, tel, login,
mot_de_passe) VALUES
('Dupont', 'Alice', 'femme', '1995-02-15', 'alice.dupont@mail.com',
'0612345671', 'user01',
'$2y$10$5ozcep67XkxrvBztGBOtZepHS6DMcoDK/9g0DxFz3hKQfFF810c9u'),
('Martin', 'Bob', 'homme', '1990-06-20', 'bob.martin@mail.com', '0612345672',
'bob_', '$2y$10$5ozcep67XkxrvBztGBOtZepHS6DMcoDK/9g0DxFz3hKQfFF810c9u'),
('Durand', 'Caroline', 'femme', '1988-11-05', 'caroline.durand@mail.com',
'0612345673', 'Caro',
'$2y$10$ZaWNAk9dr3SHxBGPWd2bee7TwgiQhgAImMa7/PXwH6OibjMCOFfB6'),
('Lefevre', 'David', 'homme', '1992-03-30', 'david.lefevre@mail.com',
'0612345674', 'DAVID',
'$2y$10$NayLrL1dtZ0aG4uYPIdVyuCmGDfreyOb3TLZ/OKK1u5kjzb6jlrs.'),
('Moreau', 'Emma', 'femme', '1997-08-12', 'emma.moreau@mail.com',
'0612345675', 'Ma',
'$2y$10$3ZXeNSYAEPySHB9SmSfEt.BXGgWCJlWcXR7mihzoWQrTX4ypYEb02'),
('Petit', 'Franck', 'homme', '1985-05-10', 'franck.petit@mail.com',
'0612345676', 'Franc',
'$2y$10$1t7ME9ZTIEqPaF.He7m8FejbLd0zH/OvSROgtJCeb4LLGSRutJyIy'),
('Rousseau', 'Gina', 'femme', '1991-07-22', 'gina.rousseau@mail.com',
'0612345677', 'Gina',
'$2y$10$56WrwMzFld5nWzCT3Les6.VUC84.6UnrHCDTQDfBh8pNf0lfePPTK'),
('Blanc', 'Hugo', 'homme', '1993-12-11', 'hugo.blanc@mail.com', '0612345678',
'white', '$2y$10$65w5XV2PslXmSj7z7nR3CeJvqTpSXAPTyff7uXuUw3cqVqj23pr9G'),
('Morel', 'Isabelle', 'femme', '1989-09-01', 'isabelle.morel@mail.com',
'0612345679', 'ISA',
'$2y$10$C5bgpDXUJPLzt5PB2SQgfe2YugPR352qLZzlMcDCn2.guzGicnfRi'),
('Girard', 'Jacques', 'homme', '1994-04-18', 'jacques.girard@mail.com',
'0612345680', 'PasDebol',
'$2y$10$INBblDp34vs9EkXC3AOyT.WTXloiCorJYMy.ohmi6nYV/SHx07/Ga');

-- Insertion dans Client
INSERT INTO Client (id_client, pseudo, solde_jetons, date_inscription) VALUES
(1, 'Alice95', 100, '2023-01-10'),
(2, 'Bob90', 50, '2023-02-15'),
(3, 'Caroline88', 200, '2023-03-20'),
(4, 'David92', 75, '2023-04-25'),
(5, 'Emma97', 150, '2023-05-30');
```

```
-- Insertion dans Admin
INSERT INTO Admin (id_admin, date_anciennete_employe, poste_employe,
status_employe) VALUES
(6, '2020-01-01', 'gerant', 'travail'),
(7, '2021-06-15', 'nettoyage', 'en_pause'),
(8, '2019-09-10', 'personnel', 'travail'),
(9, '2022-03-20', 'gerant', 'repos'),
(10, '2020-11-05', 'personnel', 'travail');

-- Insertion dans Fournisseur
INSERT INTO Fournisseur (nom_four, mail_four, tel_four, adresse_four) VALUES
('Nintendo', 'contact@nintendo.com', '0611122233', '12 rue du Jeu, Tokyo'),
('Capcom', 'support@capcom.com', '0622233344', '45 avenue du Fun, Osaka'),
('Sega', 'info@sega.com', '0633344455', '78 boulevard Play, Tokyo'),
('Konami', 'sales@konami.com', '0644455566', '9 rue Arcade, Tokyo'),
('Atari', 'contact@atari.com', '0655566677', '22 rue Jeton, Paris');

-- Insertion dans Jeux
INSERT INTO Jeux (nom, etat_jeu) VALUES
('Pacman', 'Disponible'),
('Super Mario', 'Disponible'),
('Tetris', 'En Maintenance'),
('Zelda', 'Disponible'),
('Donkey Kong', 'HS');

-- Insertion dans Jeux_Solo
INSERT INTO Jeux_Solo (id_jeux_solo, difficulte) VALUES
(1, 'Facile'),
(2, 'Normal'),
(3, 'Difficile');

-- Insertion dans Jeux_Multi
INSERT INTO Jeux_Multi (id_jeux_multi, type_equipe, nb_joueur_min,
nb_joueur_max) VALUES
(4, 'vs', 2, 4),
(5, 'Coop', 4, 4);
```

```
-- Insertion dans Carte
INSERT INTO Carte (status_carte, date_carte, version_carte, id_client) VALUES
('Liee', '2023-01-11', 1, 1),
('Liee', '2023-02-16', 1, 2),
('Perdu', '2023-03-21', 2, 4),
('Liee', '2023-04-26', 2, 4),
('Liee', '2023-05-31', 1, 5);

-- Insertion dans Recharge
INSERT INTO Recharge (montant_jeton, mode_paiement, date_recharge, id_carte)
VALUES
(50, 'CB', '2023-01-15', 1),
(100, 'espece', '2023-02-20', 2),
(75, 'CB', '2023-03-25', 3),
(150, 'CB', '2023-04-30', 4),
(200, 'espece', '2023-06-05', 5);

-- Insertion dans Borne
INSERT INTO Borne (etat_borne, prix_jeton, date_achat, id_four) VALUES
('Disponible', 5, '2022-01-01', 1),
('En Maintenance', 10, '2022-06-15', 2),
('Disponible', 7, '2022-09-10', 3),
('HS', 5, '2023-03-20', 4),
('Disponible', 8, '2023-11-05', 5);
```

```
-- Insertion dans Jeux_Borne
INSERT INTO Jeux_Borne (id_jeux, id_borne) VALUES
-- Insertion dans Borne 1
(1, 1),
(2, 1),
-- Insertion dans Borne 2
(5, 2),
-- Insertion dans Borne 3
(1, 3),
(3, 3),
(4, 3),
-- Insertion dans Borne 5
(1, 5),
(2, 5),
(3, 5),
(4, 5),
(5, 5);

-- Insertion dans Partie
INSERT INTO Partie (score, date_partie, recompense, status_partie, id_client,
id_jeux, id_borne) VALUES
(100, '2023-07-01', 0, 'Termine', 5, 1, 1),
(85000, '2023-07-02', 2,'Recompense donnee', 5, 2, 5),
(50, '2023-07-03', 1,'Recompense donnee', 1, 1, 3),
(120, '2023-07-04', 5,'Recompense donnee', 3, 4, 5),
(NULL, '2025-11-09', 0,'En cours', 4, 2, 5);
```

Index des figures