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
Professeur	M. Lemaire

Version du document	4
Date de modification	16/11/25

Réalisé par ZURKHANG Tenzin, YANG Kaiwei et COSTA Mathéo

Sommaire

Historique des versions

Versions apportées

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.

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

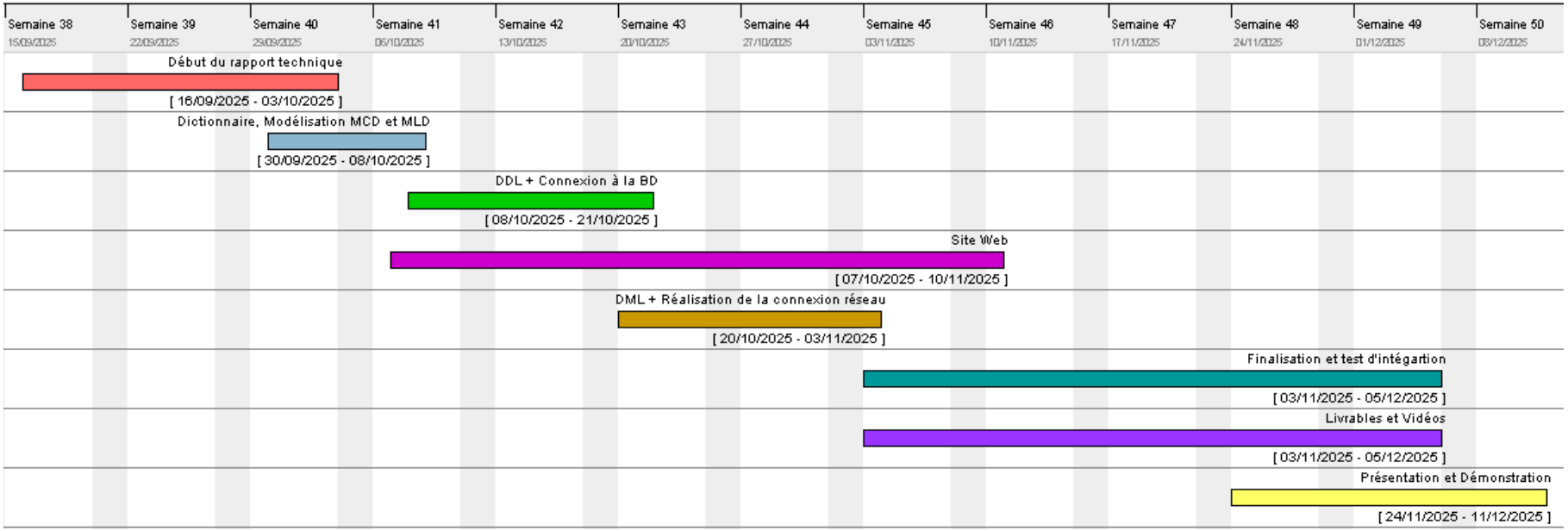

Figure 1: Diagramme de Gantt

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

- 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
etat_jeu	ENUM	(“Disponible”, “En Maintenance”, “HS”)	par défaut : Disponible	En Maintenance	NN
type_score	ENUM	(“Temps”, “Points”)		Points	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

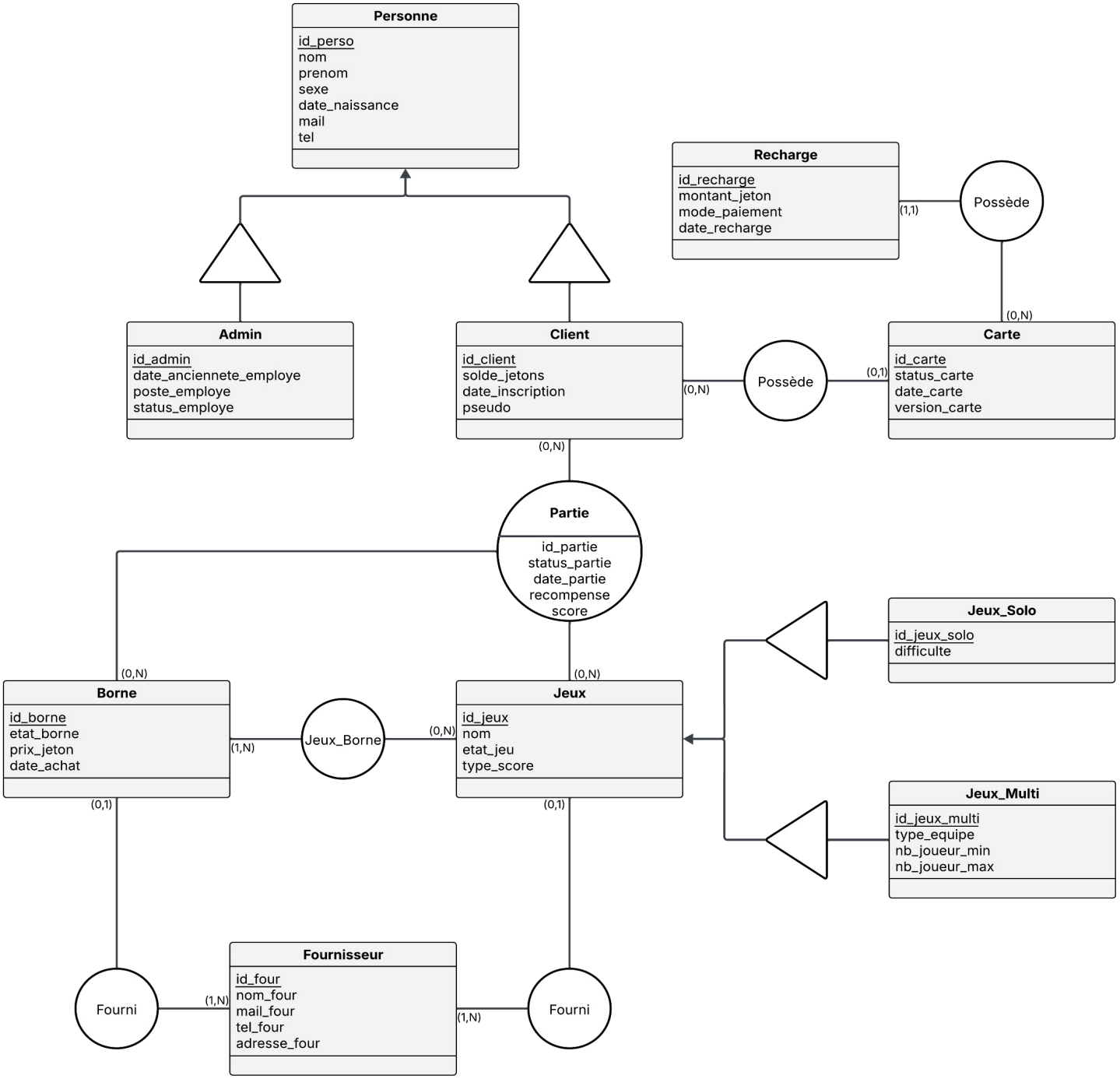

Figure 2: MCD

MLD

- Personne (id_perso, nom, prenom, sexe, date_naissance, mail, tel)
- 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, type_score, #id_four)
- 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
1	Dupon	Alice	femme	1995-02-15	alice.dupont@mail.com	0612345671
2	Martin	Bob	homme	1990-06-20	bob.martin@mail.com	0612345672
3	Durand	Caroline	femme	1988-11-05	caroline.durand@mail.com	0612345673
4	Lefevre	David	homme	1992-03-30	david.lefevre@mail.com	0612345674
5	Moreau	Emma	femme	1997-08-12	emma.moreau@mail.com	0612345675
6	Petit	Franck	homme	1985-05-10	franck.petit@mail.com	0612345676
7	Rousseau	Gina	femme	1991-07-22	gina.rousseau@mail.com	0612345677
8	Blanc	Hugo	homme	1993-12-11	hugo.blanc@mail.com	0612345678
9	Morel	Isabelle	femme	1989-09-01	isabelle.morel@mail.com	0612345679
10	Girard	Jacques	homme	1994-04-18	jacques.girard@mail.com	0612345680

- 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	type_score	id_four
1	Pacman	Disponible	Points	1
2	Super Mario	Disponible	Temps	2
3	Tetris	En Maintenance	Points	3
4	Zelda	Disponible	Points	4
5	Donkey Kong	HS	Temps	5

- 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

Réseau

Données échangées via le réseau

Tous les échanges entre la borne d'arcade et le serveur reposent sur des messages structurés envoyés via TCP, comprenant à chaque étape une commande claire et des paramètres précis (par exemple : PSEUDO card <id_carte>, SOLDE card <id_carte>, GAME card <id_carte>, borne <id_borne>, etc.). Chaque réponse du serveur commence systématiquement par un mot-clé (attribut) indiquant le type de résultat attendu : cela permet à la borne d'identifier immédiatement le contexte et de traiter correctement les données reçues, sans risque de confusion.

À chaque demande, le serveur répond soit avec une confirmation (OK pseudo <pseudo>, OK balance <solde_jetons>, etc.), soit signale une erreur avec un mot-clé explicite (ERR <raison>). En cas d'erreur spéciale (carte bloquée, solde insuffisant…), le serveur détaille toujours la cause afin que la borne affiche instantanément l'explication à l'utilisateur, sans nouvelle requête.

Si aucune réponse n'arrive (défaillance réseau ou serveur), la borne détecte le problème après un certain délai : elle peut alors renouveler la demande automatiquement ou informer un opérateur pour garantir la continuité du service.

L'ensemble du protocole est conçu pour que la borne affiche toujours l'information la plus pertinente sans multiplier les requêtes : chaque réponse serveur contient toutes les données nécessaires à la situation courante, pour une interaction fluide et fiable.

Diagrammes applicatifs

Échanges avant la partie

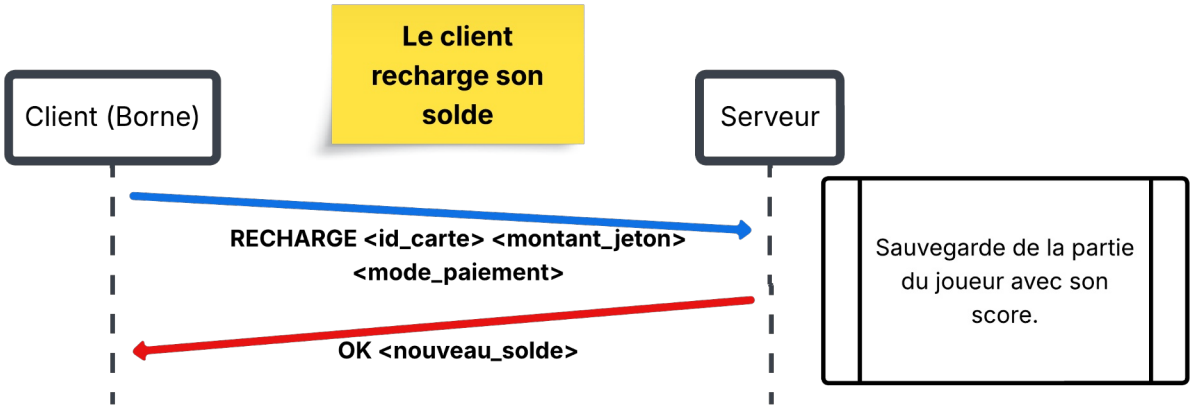

Figure 3: Échange réseau : Recharger son solde

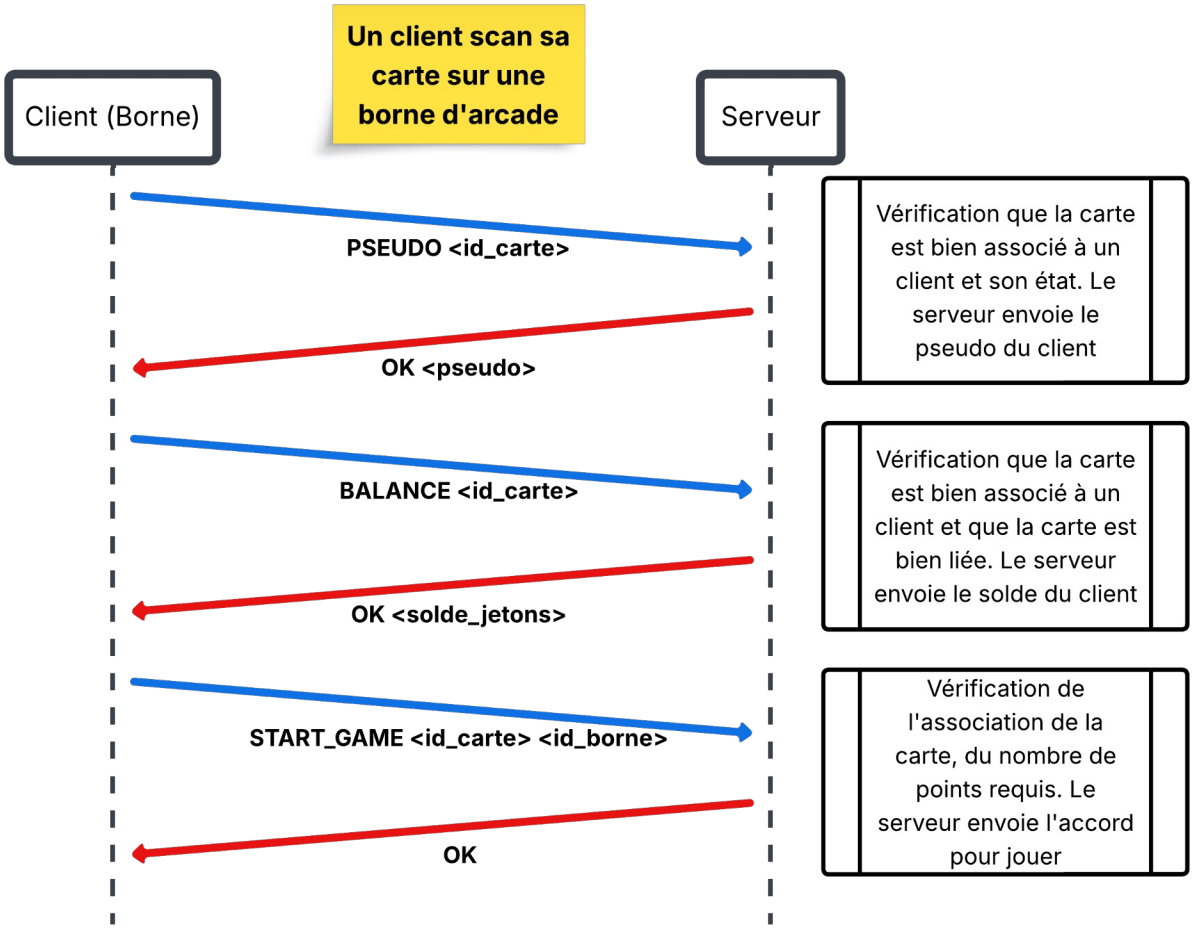

Figure 4: Échange réseau : Lancement de la partie

Échanges après la partie

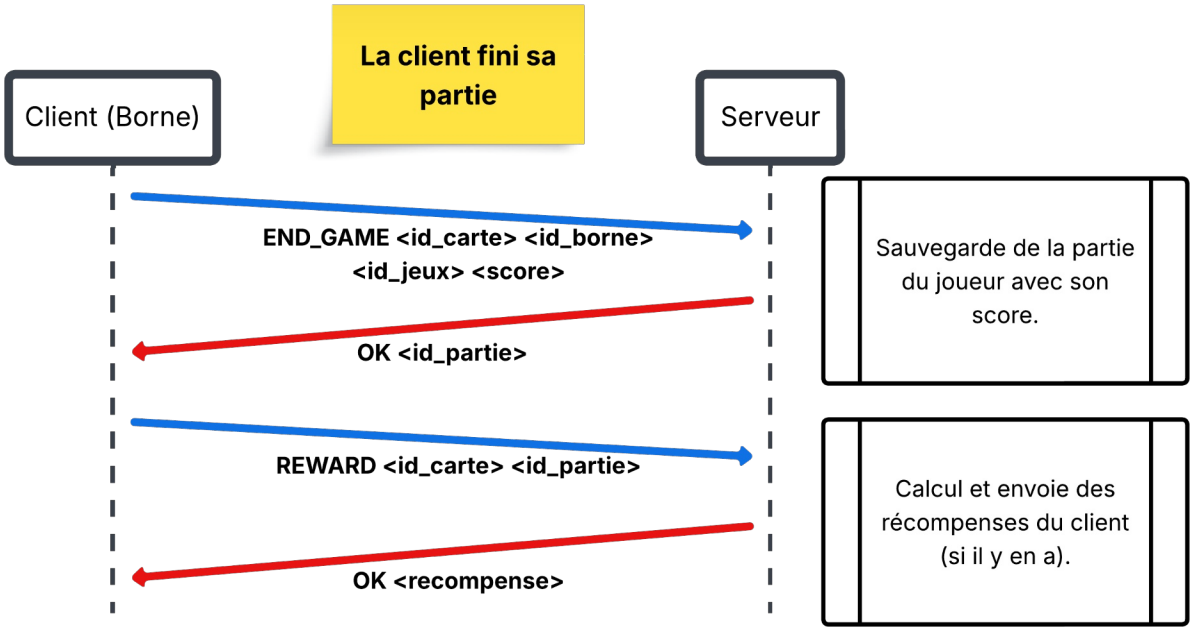

Figure 5: Échange réseau : Fin de la partie

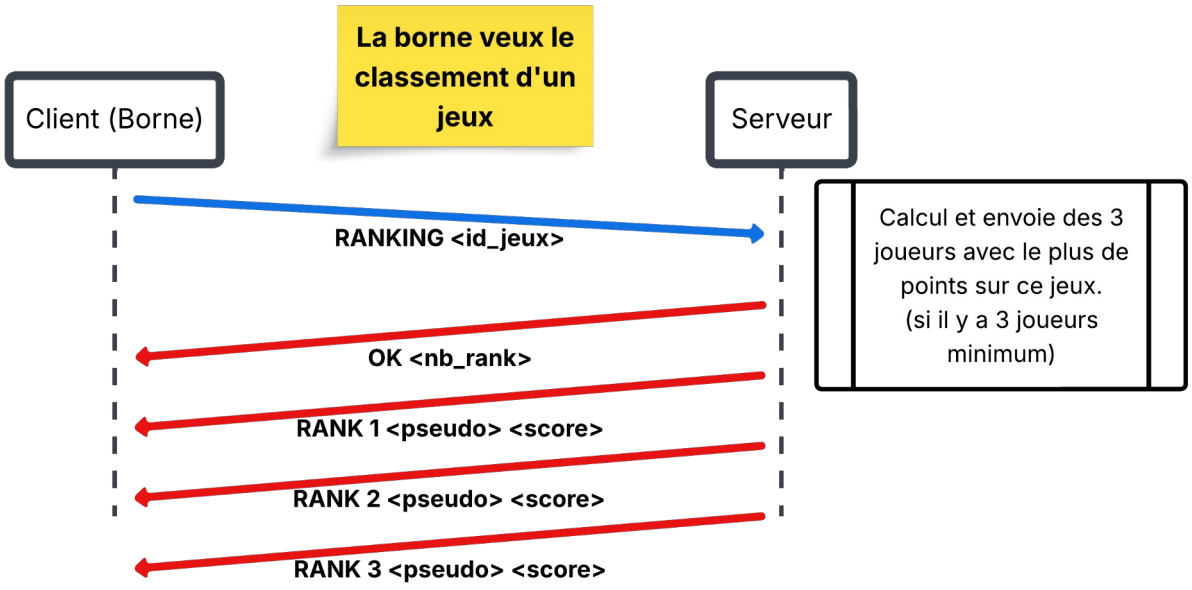

Figure 6: Échange réseau : Classement du jeu

Tests et validation

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

Test de démarrage du serveur

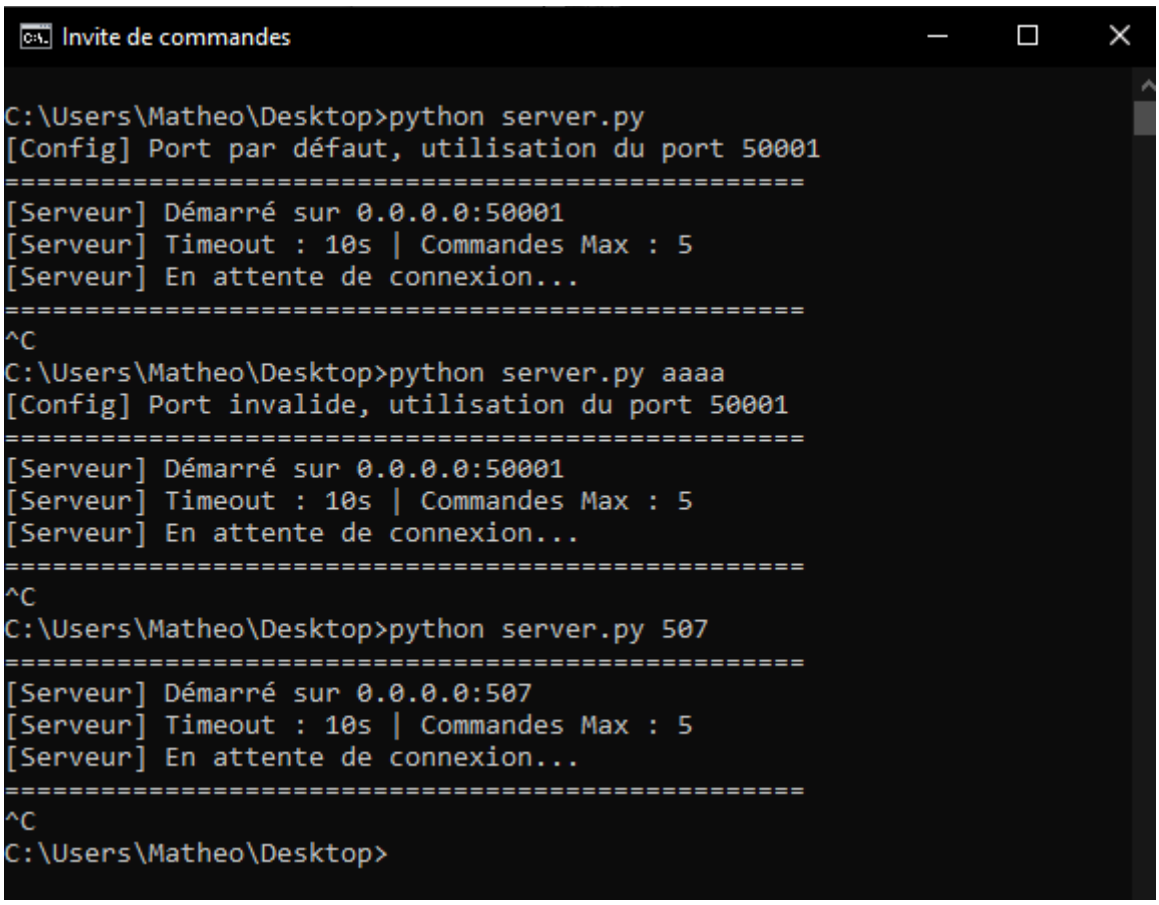

Figure 7: Test serveur : Configuration du numéro de port

Dans la Figure 7, 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

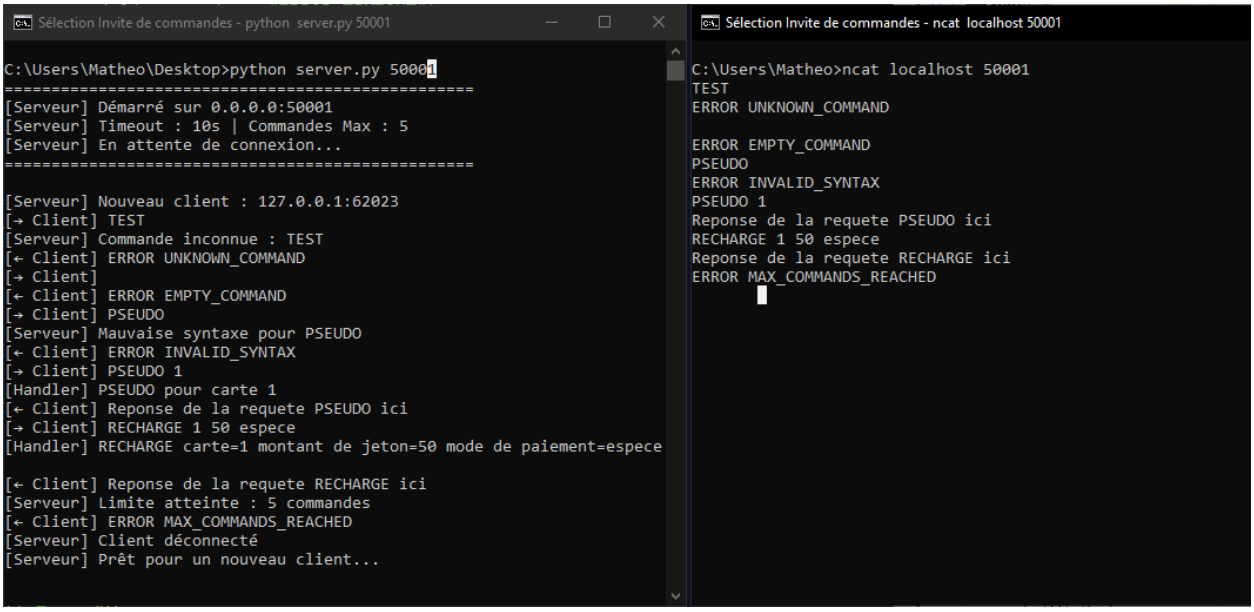

Figure 8: Test serveur : Commandes prise en charge

Dans la Figure 8, 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 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

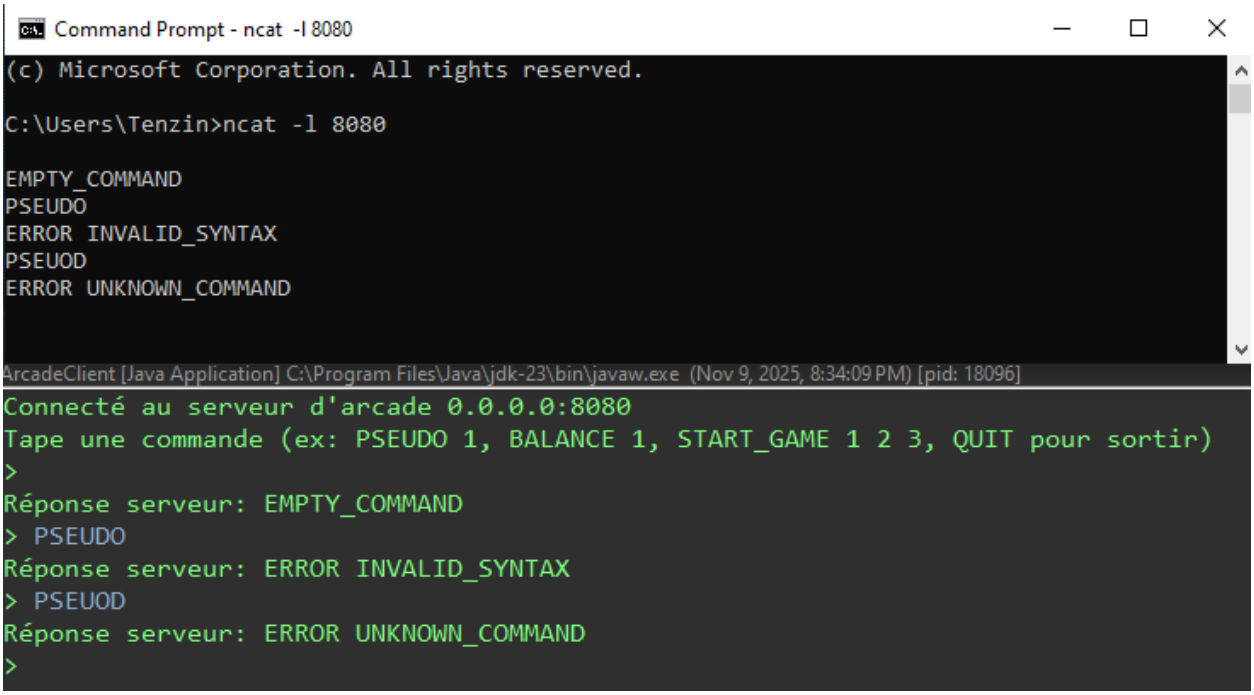

Figure 9: Test client : Commandes invalides

Dans la Figure 9, 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

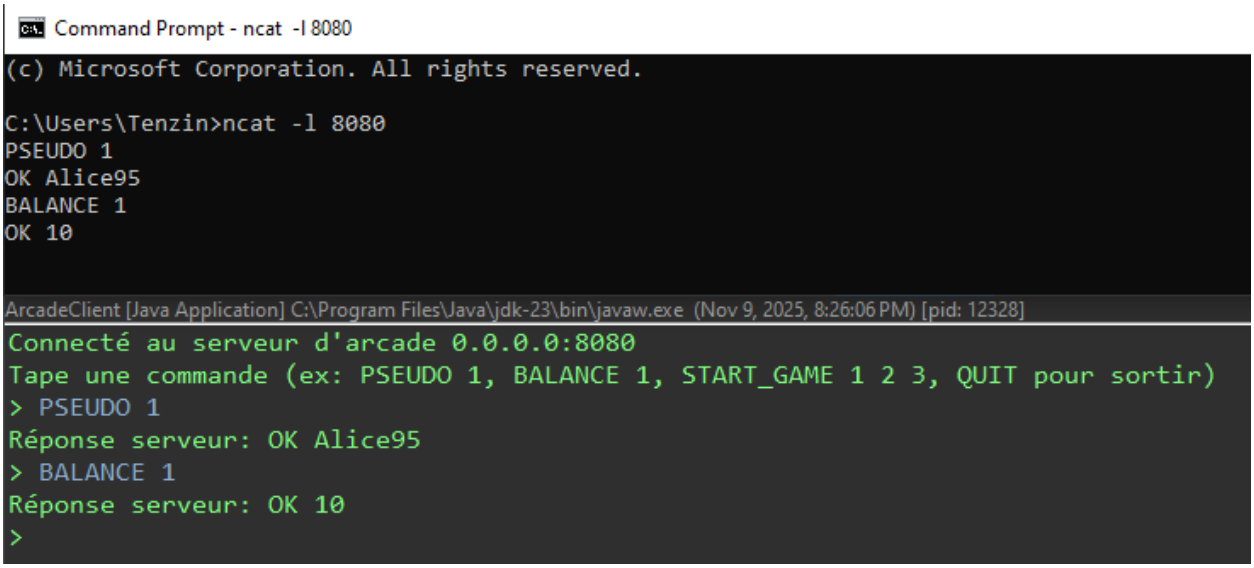

Figure 10: Test client : Commandes valides
Dans la Figure 10, nous testons des commandes correctes avec des données réelles issues de la base de données.
PSEUDO 1 : Le client envoie une demande de récupération du pseudo pour la carte ID 1. Le serveur interroge la base de données et répond OK Alice95, 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

Annexes

Code source

Serveur (Python)

```
import sys
import socket

# Configuration
SERVER_IP = "0.0.0.0"
SERVER_PORT = 50001 # Port choisi pour notre projet
CLIENT_TIMEOUT = 10
MAX_COMMANDS = 5


# ============================================================
# HANDLERS - Retournent les réponses à envoyer au client
# ============================================================

def handle_pseudo(id_carte):
    """Récupère le pseudo associé à la carte"""
    print(f"[Handler] PSEUDO pour carte {id_carte}")
    # TODO : requête SQL
    return "Reponse de la requete PSEUDO ici"


def handle_balance(id_carte):
    """Récupère le solde de la carte"""
    print(f"[Handler] BALANCE pour carte {id_carte}")
    # TODO : requête SQL
    return "Reponse de la requete BALANCE ici"


def handle_start_game(id_carte, id_borne, id_jeux):
    """Lance une partie"""
    print(f"[Handler] START_GAME carte={id_carte} borne={id_borne}
jeu={id_jeux}")
    # TODO : requête SQL
    return "Reponse de la requete START_GAME ici"
```

```
def handle_end_game(id_carte, id_borne, id_jeux, score):
    """Termine une partie"""
    print(f"[Handler] END_GAME carte={id_carte} score={score}")
    # TODO : requête SQL
    return "Reponse de la requete END_GAME ici"


def handle_reward(id_carte):
    """Calcule et renvoie la récompense"""
    print(f"[Handler] REWARD pour carte {id_carte}")
    # TODO : requête SQL
    return "Reponse de la requete REWARD ici"


def handle_ranking(id_jeux):
    """Récupère le classement d'un jeu"""
    print(f"[Handler] RANKING pour jeu {id_jeux}")
    # TODO : requête SQL
    return "Reponse de la requete RANKING ici"


def handle_recharge(id_carte, montant_jeton, mode_paiement):
    """Recharge les jetons d'un client"""
    print(f"[Handler] RECHARGE carte={id_carte} montant de
jeton={montant_jeton} mode de paiement={mode_paiement}")
    # TODO : requête SQL
    return "Reponse de la requete RECHARGE ici"


# ============================================================
# FONCTIONS UTILITAIRES
# ============================================================
```

```
def receive_command(client_socket):
    """Reçoit une commande du client"""
    try:
        data = client_socket.recv(1024).decode("utf-8")
        if not data:
            # Connexion fermée par le client
            return None
        command = data.split('\n')[0].strip()
        return command
    except socket.timeout:
        print("[Serveur] TIMEOUT du client")
        send_response(client_socket, "ERROR TIMEOUT")
        return None
    except Exception as e:
        print(f"[Serveur] ERREUR réception : {e}")
        send_response(client_socket, f"{e}")
        return None


def send_response(client_socket, response):
    """Envoie une réponse au client"""
    try:
        client_socket.send((response + "\n").encode("utf-8"))
        print(f"[← Client] {response}")
    except Exception as e:
        print(f"[Serveur] ERREUR envoi : {e}")


# ============================================================
# TRAITEMENT DES COMMANDES
# ============================================================
```

```
def process_command(command):
    """Parse et traite une commande"""
    parts = command.split()

    if not parts:
        return "ERROR EMPTY_COMMAND"

    command_name = parts[0].upper()

    # Mapping des commandes
    commands = {
        "PSEUDO": (2, handle_pseudo),
        "BALANCE": (2, handle_balance),
        "START_GAME": (4, handle_start_game),
        "END_GAME": (5, handle_end_game),
        "REWARD": (2, handle_reward),
        "RANKING": (2, handle_ranking),
        "RECHARGE": (4, handle_recharge),
    }

    if command_name not in commands:
        print(f"[Serveur] Commande inconnue : {command_name}")
        return "ERROR UNKNOWN_COMMAND"

    expected_args, handler = commands[command_name]
        if len(parts) != expected_args:
        print(f"[Serveur] Mauvaise syntaxe pour {command_name}")
        return "ERROR INVALID_SYNTAX"

    try:
        return handler(*parts[1:])
    except Exception as e:
        print(f"[Serveur] ERREUR handler : {e}")
        return "ERROR HANDLER_FAILED"


# ============================================================
# GESTION DU CLIENT
# ============================================================
```

```
def handle_client(client_socket, client_address):
    """Gère un client"""
    print(f"\n[Serveur] Nouveau client : {client_address[0]}:
{client_address[1]}")

    client_socket.settimeout(CLIENT_TIMEOUT)
    command_count = 0

    try:
        while True:

            if command_count >= MAX_COMMANDS:
                print(f"[Serveur] Limite atteinte : {MAX_COMMANDS}
commandes")
                send_response(client_socket, "ERROR MAX_COMMANDS_REACHED")
                break

            # Recevoir commande
            command = receive_command(client_socket)
            if command is None:
                break

            print(f"[→ Client] {command}")
            command_count += 1

            # Traiter et répondre
            response = process_command(command)
            send_response(client_socket, response)

    except socket.timeout:
        print("[Serveur] Timeout client")
        send_response(client_socket, "ERROR TIMEOUT")
    except Exception as e:
        print(f"[Serveur] ERREUR : {e}")
        send_response(client_socket, "ERROR SERVER_ERROR")
    finally:
        client_socket.close()
        print("[Serveur] Client déconnecté")


# ============================================================
# SERVEUR PRINCIPAL
# ============================================================
```

```
def run_server():
    """Lance le serveur"""


    try:
        server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        server.bind((SERVER_IP, SERVER_PORT))
        server.listen(0)

        print("=" * 50)
        print(f"[Serveur] Démarré sur {SERVER_IP}:{SERVER_PORT}")
        print(f"[Serveur] Timeout : {CLIENT_TIMEOUT}s | Commandes Max :
{MAX_COMMANDS}")
        print("[Serveur] En attente de connexion…")
        print("=" * 50)

        while True:
            client_socket, client_address = server.accept()
            handle_client(client_socket, client_address)
            print("[Serveur] Prêt pour un nouveau client…")

    except Exception as e:
        print(f"[Serveur] ERREUR : {e}")
    finally:
        server.close()
        print("[Serveur] Arrêté")


if len(sys.argv) > 1:
    try:
        SERVER_PORT = int(sys.argv[1])
    except ValueError:
        print("[Config] Port invalide, utilisation du port 50001")
else:
    print("[Config] Port par défaut, utilisation du port 50001")


if __name__ == "__main__":
    run_server()
```

Client (Java)

```
import java.io.*;
import java.net.*;
import java.util.Scanner;

public class ArcadeClient {

    public static void main(String[] args) {
        String host = "0.0.0.0";
        int port = 8080;

        try (Socket socket = new Socket(host, port)) {
            System.out.println("Connecté au serveur d'arcade " + host + ":" +
port);

            PrintWriter out = new PrintWriter(socket.getOutputStream(),
true);

            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

            Scanner scanner = new Scanner(System.in);
            String command, response;

            System.out.println("Tape une commande (ex: PSEUDO 1, BALANCE 1,
START_GAME 1 2 3, QUIT pour sortir)");

            while (true) {
                System.out.print("> ");
                command = scanner.nextLine();

                if (command.equalsIgnoreCase("QUIT")) {
                    System.out.println("Déconnexion du serveur...");
                    break;
                }

                out.println(command);

                response = in.readLine();

                if (response == null) {
                    System.out.println("Le serveur a fermé la connexion.");
                    break;
```

```
                }

                System.out.println("Réponse serveur: " + response);
            }

            scanner.close();

        } catch (IOException e) {
            System.err.println("Erreur de connexion au serveur : " +
e.getMessage());
        }
    }
}
```

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
);

-- 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')),
    type_score VARCHAR(10) NOT NULL CHECK (type_score IN ('Temps',
'Points')),
      id_four INT NOT NULL REFERENCES Fournisseur(id_four)
        ON DELETE CASCADE ON UPDATE CASCADE
);

-- 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)),
      id_client INT NOT NULL REFERENCES Client(id_client),
      status_partie VARCHAR(20) NOT NULL DEFAULT 'En cours' CHECK
(status_partie IN ('En cours', 'Termine', 'Recompense donnee')),
      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) VALUES
('Dupont', 'Alice', 'femme', '1995-02-15', 'alice.dupont@mail.com',
'0612345671'),
('Martin', 'Bob', 'homme', '1990-06-20', 'bob.martin@mail.com',
'0612345672'),
('Durand', 'Caroline', 'femme', '1988-11-05', 'caroline.durand@mail.com',
'0612345673'),
('Lefevre', 'David', 'homme', '1992-03-30', 'david.lefevre@mail.com',
'0612345674'),
('Moreau', 'Emma', 'femme', '1997-08-12', 'emma.moreau@mail.com',
'0612345675'),
('Petit', 'Franck', 'homme', '1985-05-10', 'franck.petit@mail.com',
'0612345676'),
('Rousseau', 'Gina', 'femme', '1991-07-22', 'gina.rousseau@mail.com',
'0612345677'),
('Blanc', 'Hugo', 'homme', '1993-12-11', 'hugo.blanc@mail.com',
'0612345678'),
('Morel', 'Isabelle', 'femme', '1989-09-01', 'isabelle.morel@mail.com',
'0612345679'),
('Girard', 'Jacques', 'homme', '1994-04-18', 'jacques.girard@mail.com',
'0612345680');

-- 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, type_score, id_four) VALUES
('Pacman', 'Disponible', 'Points', 1),
('Super Mario', 'Disponible', 'Temps', 2),
('Tetris', 'En Maintenance', 'Points', 3),
('Zelda', 'Disponible', 'Points', 4),
('Donkey Kong', 'HS', 'Temps', 5);

-- 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 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);
```

10 requêtes SQL

1) Quel est le solde de chaque client?

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

2) 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;
```

3) 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;
```

4) 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;
```

5) 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;
```

6) 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;
```

7) 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;
```

8) 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;
```

9) Quelle sont les top 3 joueurs pour chaque jeu ?

```
SELECT jeu, type_score, pseudo, meilleur_score
SELECT
    j.nom AS jeu,
    j.type_score,
    c.pseudo,
    p.score AS meilleur_score
FROM Partie p
JOIN Jeux j ON p.id_jeux = j.id_jeux
JOIN Client c ON p.id_client = c.id_client
WHERE p.score IS NOT NULL
  AND (
    (j.type_score = 'Points' AND
      (SELECT COUNT(DISTINCT p2.score)
       FROM Partie p2
       WHERE p2.id_jeux = p.id_jeux AND p2.score > p.score AND p2.score IS
NOT NULL) < 3
    )
    OR
    (j.type_score = 'Temps' AND
      (SELECT COUNT(DISTINCT p2.score)
       FROM Partie p2
       WHERE p2.id_jeux = p.id_jeux AND p2.score < p.score AND p2.score IS
NOT NULL) < 3
    )
  )
GROUP BY j.nom, j.type_score, c.pseudo, p.score
ORDER BY j.nom,
         CASE WHEN j.type_score = 'Points' THEN p.score END DESC,
         CASE WHEN j.type_score = 'Temps' THEN p.score END ASC;
```

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

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

Index des figures