PROJET D'EXAMEN · MULTI-LANGAGES

# Projet 3
# Gestionnaire de tournoi (ELO)

Classement Elo pour tournois multi-joueurs

---

| | |
|---|---|
| **Domaine :** | Jeux · Sport · Classement compétitif |
| **Composantes :** | C (DLL) · Python (module + tests) · Flask (API + dashboard) · C# (GUI) |
| **Cours :** | 5e TTR Informatique · 6e TTR Informatique |
| **Format :** | Projet individuel |
| **Édition :** | 19 mai 2026 |

## 1 Contexte et objectifs

Tu construis un système de gestion d'un tournoi (échecs, eSport, tennis, jeux de société) avec mise à jour automatique du classement selon la formule Elo classique. Tu enregistres les joueurs, tu saisis les matchs, et les Elo se recalculent à la volée. Le dashboard montre le classement et la progression individuelle.

### Objectifs pédagogiques

- Compiler une bibliothèque C en DLL et l'appeler depuis Python via `ctypes`.
- Écrire un module Python testable, avec validation et persistance des données.
- Concevoir une API REST Flask et un dashboard Chart.js qui la consomme.
- Développer une interface C# Windows Forms qui communique avec l'API.
- Comparer trois approches de calcul (Python pur, ctypes scalaire, ctypes batch).

## 2 Cahier des charges fonctionnel

L'application doit permettre à l'utilisateur de :

- Création / modification / suppression de joueurs (pseudo, Elo initial = 1200).
- Saisie d'un match : joueur A, joueur B, résultat ∈ {1.0, 0.5, 0.0}.
- Calcul Elo automatique avec K-factor configurable (16 / 24 / 32).
- Affichage du classement courant.
- Historique des matchs d'un joueur.
- Courbe d'évolution Elo d'un joueur sélectionné.
- Mode « recalcul total » qui rejoue tous les matchs dans l'ordre.

> **Formule Elo —** E_a = 1 / (1 + 10^((Elo_b − Elo_a) / 400)) new_Elo_a = Elo_a + K × (Result_a − E_a)

> **K-factor recommandé —** K = 32 pour les joueurs débutants (< 30 parties) K = 16 pour les joueurs confirmés (Elo > 2100)

## 3 Module C — `calculs.c` → `calculs.dll`

Les fonctions ci-dessous sont à exporter dans la DLL. Chaque fonction doit être présente en version **scalaire** (pour l'application normale) ; les fonctions marquées *batch* servent au benchmark de la Partie 5.

| Signature | Rôle |
| --- | --- |
| `float expected_score(int elo_a, int elo_b);` | 1 / (1 + 10^((b−a)/400)) |
| `int new_elo(int current, float expected, float actual, int k);` | current + k × (actual − expected) |
| `void expected_score_batch(int* a, int* b, float* out, int n);` | Calcul espéré batch |
| `void new_elo_batch(int* cur, float* exp, float* act, int k, int* out, int n);` | Recalcul batch |

Compilation Windows : `gcc -shared -o calculs.dll calculs.c -lm`

## 4 Module Python — `module.py`

Le module Python est le seul pont entre la DLL et le reste de l'application. Il contient les wrappers `ctypes` et la logique métier (validation, persistance, statistiques).

- `expected_score(elo_a, elo_b) -> float — wrapper ctypes`
- `new_elo(current, expected, actual, k=32) -> int — wrapper ctypes`
- `add_player(name) -> dict`
- `add_match(player_a_id, player_b_id, result, k=32) -> dict — applique l'update`
- `get_ranking() -> list[dict] — trié par Elo desc`
- `get_player_history(player_id) -> list — historique Elo`
- `recalculate_chain() -> int — rejoue tous les matchs`

## 5 API Flask — `app.py`

Toutes les réponses respectent le format `{"status": "ok|error", "data": ...}`. Statuts HTTP : 200 (lecture), 201 (création), 400 (validation), 404 (non trouvé), 500 (erreur serveur).

| Méthode | Chemin | Description |
|---|---|---|
| GET | `/api/players` | Liste |
| POST | `/api/players` | Création (Elo initial 1200) |
| GET | `/api/players/{id}` | Détail + historique |
| DELETE | `/api/players/{id}` | Suppression |
| GET | `/api/matches` | Liste paginée |
| POST | `/api/matches` | Création (déclenche calcul Elo) |
| GET | `/api/ranking` | Classement trié |
| POST | `/api/recalculate` | Recalcul complet |

## 6 Dashboard web — `/dashboard`

Dashboard lecture seule, rafraîchissement automatique toutes les 30 secondes via fetch JS. Trois graphiques Chart.js obligatoires :

1. Top 10 du classement (bar chart horizontal).
2. Courbe Elo d'un joueur sélectionné (line chart).
3. Distribution des Elo par tranche de 100 (histogram).

## 7 Interface C# — Windows Forms

Application Windows Forms qui communique avec l'API uniquement via `HttpClient`. Aucun calcul ne doit être fait côté C#.

- PlayersForm — CRUD joueurs (DataGridView).
- MatchEntryForm — sélection des 2 joueurs, résultat radio, K-factor.
- RankingForm — classement en direct.
- PlayerDetailForm — historique + mini-graph d'évolution.

## 8 Format des données

### Variante 5e — Persistance JSON

```
{
  "player": { "id": 1, "name": "Magnus", "elo": 1820 },
  "match": {
    "id": 7, "player_a_id": 1, "player_b_id": 2,
    "result": 1.0,
    "elo_a_before": 1820, "elo_a_after": 1825,
    "elo_b_before": 1790, "elo_b_after": 1785,
    "date": "2026-05-19T18:00:00"
  }
}
```

### Variante 6e — Persistance SQL (MySQL/SQLite)

```
CREATE TABLE players (
  id   INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL UNIQUE,
  elo  INTEGER NOT NULL DEFAULT 1200
);
CREATE TABLE matches (
  id            INTEGER PRIMARY KEY AUTOINCREMENT,
  player_a_id   INTEGER NOT NULL REFERENCES players(id),
  player_b_id   INTEGER NOT NULL REFERENCES players(id),
  result        REAL NOT NULL CHECK (result IN (0.0, 0.5, 1.0)),
  elo_a_before  INTEGER NOT NULL,
  elo_a_after   INTEGER NOT NULL,
  elo_b_before  INTEGER NOT NULL,
  elo_b_after   INTEGER NOT NULL,
  date          TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

## 9 Plan de tests Pytest

Minimum 10 tests Pytest dans `tests_module.py`. Tous doivent retourner **PASSED**.

1. `test_expected_score_equal_elos — joueurs égaux → 0.5`
2. `test_expected_score_400_diff — différence 400 → favori ≈ 0.909`
3. `test_expected_score_symmetry — E_a + E_b = 1.0`
4. `test_new_elo_win_gain_positive`
5. `test_new_elo_loss_loses_points`
6. `test_new_elo_draw_balanced_pair`
7. `test_add_match_updates_both_elos`
8. `test_ranking_sorted_desc`
9. `test_history_returns_player_matches`

10. `test_recalculate_chain_idempotent`

## 10 Critères de validation et remise

### Barème (sur 24)

| Composante | Points |
|---|---|
| Module C — fonctions scalaires + batch, compile sans warning | 4 |
| Module Python — wrapper ctypes + logique métier | 4 |
| Tests Pytest — 10 tests PASSED | 3 |
| API Flask — tous les endpoints fonctionnels | 4 |
| Dashboard web — 3 graphiques opérationnels | 3 |
| GUI C# — communication API et écrans demandés | 4 |
| Benchmark — 3 scénarios + tableau + explication orale | 2 |
| **Total** | **/24** |

### Structure du dossier de remise

```
projet-3-tournoi_elo/
├── calculs.c
├── calculs.dll
├── module.py
├── tests_module.py
├── app.py
├── generator.py
├── benchmark.py
├── templates/
│   └── dashboard.html
├── data/                        (5e — JSON)   ou   db.sqlite   (6e — SQL)
└── GUI/
    └── Projet3.sln
```

### Démonstration orale (5 minutes)

1. Lance ton API Flask et ton dashboard.
2. Lance le générateur de données et observe le dashboard se remplir.
3. Ouvre la GUI C# et démontre 2 cas d'usage.
4. Présente le résultat du benchmark et explique pourquoi le scénario batch est plus rapide.