Université Joseph-Ky ZERBO/IFOAD
UFR : Sciences Informatiques Appliquées
Filière : **Sciences des Données**
Niveau : M1-S1

Burkina Faso
La Patrie ou la Mort, nous Vaincrons
Année académique 2024-2025
Ouagadougou le 06/08/2025

Rapport de Projet : Python POO

Système de Gestion NBA - Analyse et Gestion de Données Basketball

NOM	PRENOM	INE
OUEDRAOGO	LASSINA	N00069220051
POUBERE	ABDOURAZAKOU	N00145620141

Note	**Observation**

Projet réalisé dans le cadre du cours de Programmation Orientée Objet

Lien git : https://github.com/POUBERE/nba-management-system.git

Table des matières

RESUME

Ce projet vise la conception et le développement d'un système de gestion et d'analyse des données pour la NBA (National Basketball Association), conçu en Python en suivant les principes de la programmation orientée objet (POO). Il permet de gérer les équipes, les joueurs, les statistiques individuelles, les matchs, et de produire des analyses avancées et des classements, le tout via une interface graphique intuitive développée avec Tkinter.

Le système est structuré autour de plusieurs classes : Personne, Joueur, Equipe, Match, StatistiqueJoueur, NBASystem, et NBAGui, respectant les concepts clés de la POO tels que l'héritage, l'encapsulation, l'abstraction et le polymorphisme. L'architecture est modulaire, chaque entité étant gérée dans un fichier séparé pour faciliter la maintenance, l'extensibilité et la clarté du code.

Une attention particulière a été portée à la robustesse de la validation des données, à la persistance des informations via le format JSON, et à l'analyse statistique (moyennes, classements, efficacité des joueurs). Le système démarre avec des données réalistes (équipes et joueurs célèbres) et permet une démonstration complète des fonctionnalités.

Enfin, plusieurs pistes d'évolution sont proposées : intégration d'API officielles, migration vers une base de données relationnelle, développement d'une version web, ajout de fonctionnalités avancées (contrats, draft, statistiques approfondies). Ce projet constitue une base solide pour la digitalisation des opérations de gestion sportive, tout en illustrant la maîtrise de Python et des bonnes pratiques de développement logiciel.

1. DESCRIPTION DU PROJET

1.1. Contexte et Objectifs

Ce projet consiste en le développement d'un système complet de gestion et d'analyse de données pour la NBA (National Basketball Association). La NBA étant la principale ligue de basketball au monde avec 30 équipes réparties aux États-Unis, notre logiciel vise à fournir une solution complète pour les managers d'équipes afin de :

- Gérer les équipes et leurs effectifs
- Suivre les performances individuelles des joueurs
- Analyser les statistiques de matchs
- Maintenir un historique des résultats
- Générer des classements et des analyses comparatives

1.2. Public Cible

Le système s'adresse principalement aux :

- Managers d'équipes NBA
- Analystes sportifs
- Entraîneurs
- Personnel administratif des franchises

1.3. Fonctionnalités Principales

- **Gestion des équipes** : Création, modification et consultation des équipes NBA
- **Gestion des joueurs** : Ajout, transfert et suivi des joueurs
- **Gestion des statistiques** : Enregistrement et analyse des performances individuelles
- **Gestion des matchs** : Création et suivi des résultats de matchs
- **Analyses avancées** : Classements, top joueurs, statistiques d'équipe
- **Sauvegarde de données** : Persistance des informations en format JSON
- **Interface utilisateur** : Interface graphique développée avec Tkinter (classe NBAGui)

2. ARCHITECTURE ET CONCEPTION

2.1 Approche Orientée Objet

Le projet respecte intégralement les principes de la programmation orientée objet :

- **Encapsulation** : Tous les attributs sont privés (préfixe `_`) avec des propriétés d'accès
- **Héritage** : La classe `Joueur` hérite de la classe abstraite `Personne`

- **Abstraction** : Utilisation de classes abstraites et de méthodes abstraites
- **Polymorphisme** : Implémentation de méthodes abstraites dans les classes dérivées

2.2 Structure des Classes

2.2.1. Hiérarchie d'Héritage

Personne (Classe abstraite)

├── Joueur (Classe concrète)

La classe abstraite `Personne` définit les attributs communs (nom, origine) et impose l'implémentation de la méthode `afficher_informations()`.

2.2.2. Relations entre Classes

Association :

- `Equipe` ↔ `Joueur` : Une équipe contient plusieurs joueurs
- `Match` → `Equipe` : Un match implique deux équipes

Composition :

- `Joueur` ⊙ `StatistiqueJoueur` : Un joueur possède ses statistiques (cycle de vie lié)
- `NBASystem` ⊙ `Equipe`, `Joueur`, `Match` : Le système gère tous les objets

Agrégation :

- `NBASystem` ◊ `Equipe` : Le système référence les équipes sans les posséder exclusivement

3. DÉTAIL DES CLASSES

3.1 Classe Personne (Abstraite)

Rôle : Classe de base pour toute personne dans le système

Attributs : nom (str), origine (str)

Méthodes abstraites : `afficher_informations()`

Validation : Contrôle de la validité des données d'entrée

3.2 Classe Joueur

Héritage : Hérite de `Personne`

Attributs spécifiques : année_debut (int), poste (PosteJoueur), équipe (Equipe)

Collections : Liste de statistiques personnelles

Fonctionnalités : Calcul de moyennes, recherche de records personnels

3.3 Classe StatistiqueJoueur

Rôle : Encapsule les performances d'un joueur pour un match

Attributs : temps_jeu, points, passes, rebonds, date_match

Validation : Contrôles de cohérence (temps $\leq$ 48min, valeurs positives)

Calculs : Indice d'efficacité personnalisé

3.4 Classe Equipe

Rôle : Représente une franchise NBA

Contraintes : Maximum 15 joueurs (règlement NBA)

Statistiques : Bilan victoires/défaites, pourcentage de réussite

Fonctionnalités : Gestion d'effectif, calculs d'équipe

3.5 Classe Match

Rôle : Modélise une rencontre entre deux équipes

Validation : Cohérence des équipes et scores

Automatisation : Mise à jour automatique des bilans d'équipes

Analyses : Détection de matchs serrés, calcul d'écarts

3.6 Classe NBASystem

Rôle : Contrôleur principal du système

Gestion centralisée : Toutes les entités du système

Index optimisés : Recherche rapide par dictionnaires

Fonctionnalités avancées : Classements, transferts, analyses statistiques

3.7 Classe NBAGui (Interface Graphique)

Framework : Tkinter

Rôle : Interface utilisateur graphique

Intégration : Communication avec NBASystem

Ergonomie : Interface intuitive pour toutes les fonctionnalités

4. FONCTIONNALITÉS AVANCÉES

4.1. Gestion des Données

Validation robuste : Contrôles sur tous les inputs utilisateur

Gestion d'erreurs : Try-catch appropriés avec messages informatifs

Sauvegarde JSON : Sérialisation complète du système

4.2. Analyses Statistiques

Moyennes de joueurs : Calculs automatiques sur toutes les statistiques

Classements d'équipes : Tri par pourcentage de victoires

Top joueurs : Classements par critères multiples (points, passes, rebonds, efficacité)

Statistiques système : Vue d'ensemble globale

4.3. Fonctionnalités Métier

Transferts de joueurs : Gestion complète avec validation des contraintes

Détection matchs serrés : Algorithme d'analyse de compétitivité

Gestion des postes : Énumération des 5 postes NBA officiels

5. ARCHITECTURE TECHNIQUE

5.1. Structure des Fichiers

```
code/
├── personne.py           # Classe abstraite de base
├── statistique_joueur.py # Gestion des performances
├── joueur.py             # Classe joueur avec héritage
├── equipe.py             # Gestion des équipes
├── match.py              # Modélisation des rencontres
├── nba_system.py         # Contrôleur principal
├── nba_gui.py            # Interface graphique Tkinter
└── main.py               # Programme principal
```

5.2. Dépendances

Python standard : abc, enum, datetime, json

Interface graphique : tkinter

Aucune dépendance externe requise

5.3. Modularité

Chaque classe est dans son propre fichier, favorisant :

- La lisibilité du code
- La maintenance
- La réutilisabilité des composants
- Le développement collaboratif

6. RESPECT DES CONTRAINTES

6.1. Exigences Techniques

Langage Python : Projet intégralement développé en Python

POO : Architecture complètement orientée objet

5 classes minimum : 7 classes implémentées

Héritage : Joueur hérite de Personne

Association : Multiples relations entre classes

Agrégation/Composition : Relations appropriées implémentées

6.2. Structure de Livraison

Programme principal : main.py avec menu interactif

Dossier code : Organisation claire des fichiers

GitHub public : Repository accessible

Documentation : Code commenté et rapport complet

7. GUIDE D'UTILISATION

7.1. Installation et Lancement

- Cloner le repository :
  git clone https://github.com/POUBERE/nba-management-system.git
  cd nba-management-system/code
- Lancer le programme :
  python main.py

7.2. Interface Console

Le système propose un menu interactif avec 07 options :

1. Gestion des équipes

2. Gestion des joueurs

3. Transferts

4. Statistiques

5. Matchs

6. Consultations et analyses

7. Sauvegarde

8. DONNÉES D'EXEMPLE

Le système se lance avec des données préchargées :

3 équipes : Chicago Bulls, Los Angeles Lakers, Boston Celtics

5 joueurs légendaires : Jordan, Pippen, LeBron, Davis, Tatum

Statistiques réalistes pour démonstration

3 matchs d'exemple avec résultats

9. EXTENSIONS POSSIBLES

9.1. Améliorations Futures

API Integration : Connexion à l'API NBA officielle

Base de données : Migration vers SQLite/PostgreSQL

Interface web : Développement d'une interface Django/Flask

Analytics avancées : Machine Learning pour prédictions

Visualisations : Graphiques avec matplotlib/plotly

9.2. Fonctionnalités Additionnelles

- Gestion des contrats joueurs
- Calendrier de saison
- Système de draft
- Gestion des arbitres
- Statistiques avancées (PER, True Shooting, etc.)

10. CONCLUSIONS

Ce projet de système de gestion NBA démontre une maîtrise complète des concepts de programmation orientée objet en Python. L'architecture modulaire, les relations appropriées entre classes, et l'implémentation de fonctionnalité avancées créent un système robuste et extensible.

Le respect intégral des contraintes du projet, combiné à des fonctionnalités dépassant les exigences minimales (interface graphique, sauvegarde JSON, analyses statiques), illustre une approche professionnelle du développement logiciel.

L’organisation du code en modules distincts et la documentation compréhensive facilitent la maintenance et l’évolution future du système, rendant ce projet un excellent exemple d’application des principes de génie logiciel.