TP MONOPOLY

SÉANCE 4 : IA ET ANALYSE (4h)

Objectifs pédagogiques

- Créer des stratégies d'IA avec héritage
- Analyser statistiquement le jeu

Exercice 4.1 : Trois stratégies d'IA

Contexte

Créer différentes intelligences artificielles qui jouent selon des stratégies distinctes.

Tâches à réaliser

Étape 1 : Classe de base StrategieIA

```
class StrategieIA:
    """Classe de base pour les stratégies"""
    def __init__(self, nom: str):
        self.nom = nom

    def decider_achat(self, joueur: 'Joueur', propriete: Propriete) -> bool:
        """Décide si acheter une propriété"""
        return False

    def decider_construction(self, joueur: 'Joueur') -> Optional[Propriete]:
        """Décide où construire"""
        return False
```

Étape 2 : IA Agressive

Achète toujours si elle a l'argent :

```
class IAAgressive(StrategieIA):
    """Achète systématiquement toutes les propriétés"""
    def __init__(self):
        super().__init__("Agressive")
```

```
    def decider_achat(self, joueur: 'Joueur', propriete: Propriete) -> bool:
        #TODO SÉANCE 4
```

Étape 3 : IA Conservative

N'achète que si elle garde une réserve d'argent :

```
class IAConservative(StrategieIA):
    """Achète seulement si argent > 2× prix"""
    def __init__(self):
        super().__init__("Conservative")

    def decider_achat(self, joueur: 'Joueur', propriete: Propriete) -> bool:
        # Garder au moins le double du prix
        # TODO SEANCE 4
```

Étape 4 : IA Stratégique

Privilégie les propriétés qui complètent un quartier :

```
class IAStrategique(StrategieIA):
    """Privilégie les quartiers et propriétés rentables"""
    def __init__(self):
        super().__init__("Stratégique")

    def decider_achat(self, joueur: 'Joueur', propriete: Propriete) -> bool:
        # Ne pas dépenser si trop peu d'argent ( moins de 1,5 x le prix d’achat
        #TODO SEANCE 4

        # Compter combien de propriétés de cette couleur possédées
        nb_possede = sum(1 for p in joueur.proprietes
                         if hasattr(p, 'couleur') and p.couleur == propriete.couleur)

        # On achete si ça rapproche d'un quartier complet
        # TODO SEANCE 4
```

```
    # Sinon acheter si beaucoup d'argent ( 3x le prix d’achat)
    # TODO SEANCE 4
```

Étape 5 : Construction intelligente

```
def decider_construction(self, joueur: 'Joueur') -> Optional[Propriete]:
  """Décide sur quelle propriété construire"""
  # Trouver les quartiers
  quartiers = self._trouver_quartiers(joueur)

  if not quartiers:
    return None

  # Chercher la propriété avec le meilleur ROI
  meilleure = None
  meilleur_roi = 0

  for proprietes in quartiers.values():
    for prop in proprietes:
      if prop.peut_construire(joueur):
        # Calculer le retour sur investissement
        loyer_actuel = prop.calculer_loyer()
        prop.nb_maisons += 1  # Simulation
        loyer_futur = prop.calculer_loyer()
        prop.nb_maisons -= 1  # Annuler

        roi = (loyer_futur - loyer_actuel) / prop.prix_maison

        if roi > meilleur_roi:
          meilleur_roi = roi
          meilleure = prop

  return meilleure
```

```
def _trouver_quartiers(self, joueur: 'Joueur') -> Dict[str, List[Propriete]]:
    """Liste les quartiers du joueur"""
    quartiers = {}
    for prop in joueur.proprietes:
        if isinstance(prop, Propriete) and not isinstance(prop, (Gare, Compagnie)):
            couleur = prop.couleur
            if joueur.possede_quartier_complet(couleur):
                if couleur not in quartiers:
                    quartiers[couleur] = []
                quartiers[couleur].append(prop)
    return quartiers
```

Test des IA

```
def tester_strategies():
    print("\nTEST DES STRATÉGIES IA")

    strategies = [
        IAAgressive(),
        IAConservative(),
        IAStrategique()
    ]

    for strat in strategies:
        print(f"\n{'='*60}")
        print(f"Stratégie: {strat.nom}")
        print(f"{'='*60}")

        jeu = Monopoly(["IA1", "IA2", "IA3"], strategie=strat)
        gagnant = jeu.jouer_partie(max_tours=50)

        if gagnant:
```

```
print(f"Gagnant: {gagnant.nom}")
```

Exercice 4.2 : Système de statistiques

Contexte

Collecter des données pendant les parties pour analyser le jeu.

Métriques à suivre

1. **Passages par case** : Quelle case est la plus visitée ?
2. **Revenus par propriété** : Quelle propriété rapporte le plus ?
3. **Durée des parties** : Combien de tours en moyenne ?
4. **Taux de victoire** : Quelle stratégie gagne le plus ?

Tâches à réaliser

Étape 1 : Créer la classe StatistiquesPartie

```
class StatistiquesPartie:
  """Collecte des statistiques sur une partie"""
  def __init__(self):
    self.passages_par_case: Dict[int, int] = {}
    self.revenus_par_propriete: Dict[str, int] = {}
    self.duree_partie = 0
    self.nb_tours = 0
    self.gagnant = None

  def enregistrer_passage(self, case: Case):
    """Enregistre le passage sur une case.
       Ajout dans le dictionnaire si 1er passage,incrémentation du nombre de passage
       sur cette case  """
    # TODO SÉANCE 4

  def enregistrer_loyer(self, propriete: Propriete, montant: int):
    """Enregistre un paiement de loyer. Ajout du nom de la proprieté si 1er loyer payé
       cumul du montant payé"""
    # TODO SÉANCE 4
```

Étape 2 : Intégrer dans le jeu

```
# Dans Monopoly.__init__()
self.stats = StatistiquesPartie()

# Dans jouer_tour()
self.stats.enregistrer_passage(case_arrivee)

# Dans Propriete.action() lors d'un paiement de loyer
jeu.stats.enregistrer_loyer(self, loyer)
```

Étape 3 : Affichage des statistiques

```
def afficher_statistiques(self):
    """Affiche un résumé des statistiques"""
    print("\n" + "="*60)
    print("STATISTIQUES DE LA PARTIE")
    print("="*60)

    print(f"\nDurée: {self.nb_tours} tours")
    if self.gagnant:
        print(f"Gagnant: {self.gagnant.nom} ({self.gagnant.argent}€)")

    # Top 5 cases visitées
    print("\nTop 5 des cases les plus visitées:")
    top_cases = sorted(self.passages_par_case.items(),
                       key=lambda x: x[1], reverse=True)[:5]
    for position, nb in top_cases:
        print(f"   Position {position}: {nb} passages")

    # Top 5 propriétés rentables
    if self.revenus_par_propriete:
        print("\nTop 5 des propriétés les plus rentables:")
        top_props = sorted(self.revenus_par_propriete.items(),
```

```
                key=lambda x: x[1], reverse=True)[:5]
    for nom, revenus in top_props:
        print(f"   {nom}: {revenus}€ de loyers")
```

Test final séance 4

```
# Comparaison complète
print("="*60)
print("TEST FINAL : ANALYSE COMPLÈTE DU MONOPOLY")
print("="*60)

# 1. Test des 3 IA
for strat in [IAAgressive(), IAConservative(), IAStrategique()]:
    print(f"\n{'='*60}")
    print(f"Stratégie: {strat.nom}")
    simuler_parties(50, 3, strat)

# 2. Comparaison directe
comparer_strategies(100, 4)

# 3. Analyse probabiliste
analyser_probabilites_cases()

print("\nTOUS LES TESTS RÉUSSIS!")
```