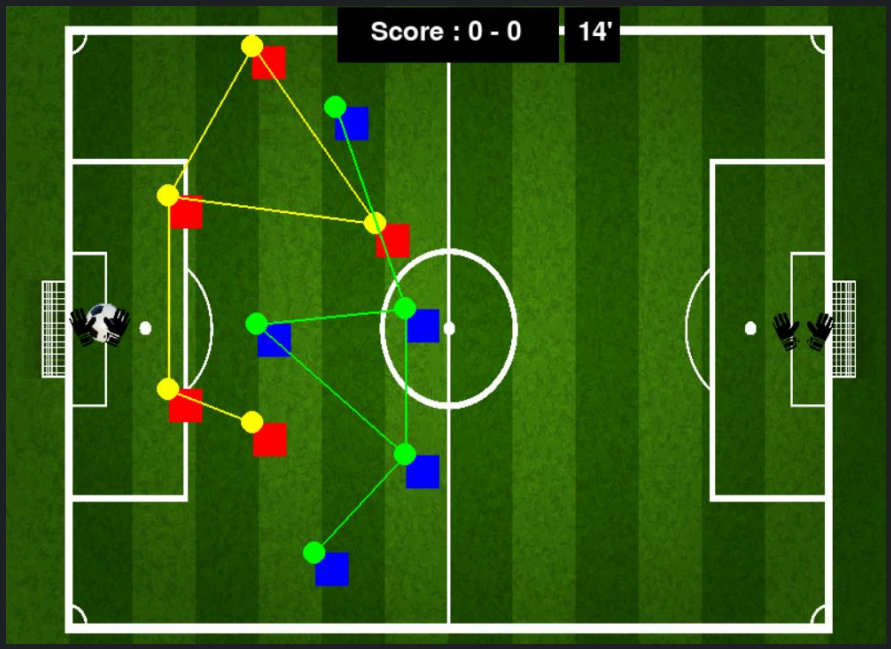
Modélisation et simulation d'un match de football virtuel

Nathan Robert

Numéro SCEI : 30458

Motivations

- Modéliser une partie de football avec des statistiques de joueurs prédéfinies
- Implémentation d’une interface utilisateur intuitive pour l'observation du jeu
- Mise en place de stratégies de jeu optimales afin de remporter la partie



Exemple d'une simulation

Plan de l'exposé

GESTION DES AFFICHAGES ET DESCRIPTION DE LA STRUCTURE GÉNÉRALE DU PROJET

IMPLÉMENTATIONS TECHNIQUES ET ÉVOLUTIONS DU MODÈLE

MISE EN PLACE DE STRATÉGIES DE JEU À L'AIDE DE LA THÉORIE DES GRAPHES

PERSPECTIVE D'AMÉLIORATION ET DE DÉVELOPPEMENT FUTUR

Utilisation de la technologie pygame

- PyGame fédère une vaste communauté très active
- permet le développement de jeu vidéo en temps réel sans les mécaniques de bas niveau du langage C et de ses dérivés
- Ne nécessite pas de GUI pour utiliser toutes les fonctions.

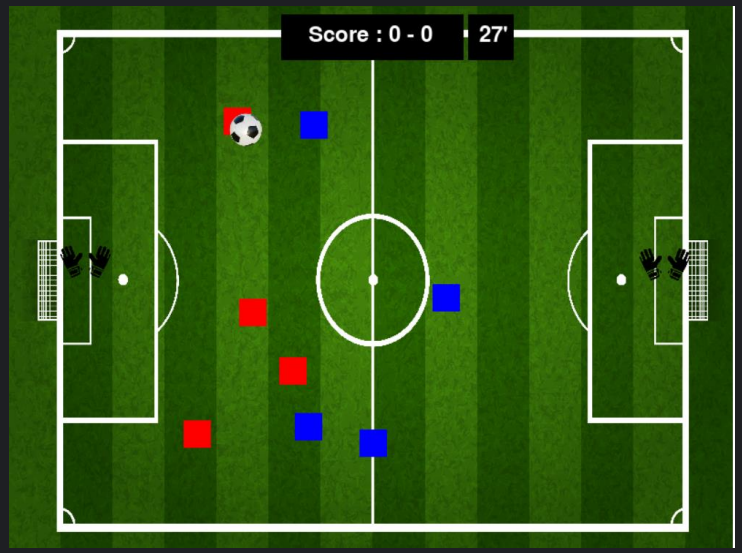
Initialisation des différents objets

```python
class Joueur:
    def __init__(self, precision, vitesse, defense, passe, distance_tir, poste):
        self.precision_tir = precision
        self.vitesse = vitesse
        self.defense = defense
        self.coord = (0, 0)
        self.porteur_de_balle = False
        self.distance_tir = distance_tir
        self.passe = passe
        self.poste = poste
        self.rect = pygame.Rect(0, 0, 30, 30)
```

```python
class Equipe:
    def __init__(self, numero, joueurs, gardien):
        self.numero = numero
        self.joueurs = joueurs
        self.gardien = gardien
        self.score = 0
        self.equipe_balle = False
        self.strategie = File()
```

- Ces objets sont immuables
- Grâce à l'héritage, les classes peuvent être dérivées d'autres classes
- Chaque classe représente une unité de code distincte, ce qui facilite la maintenance

Gestion des mouvements



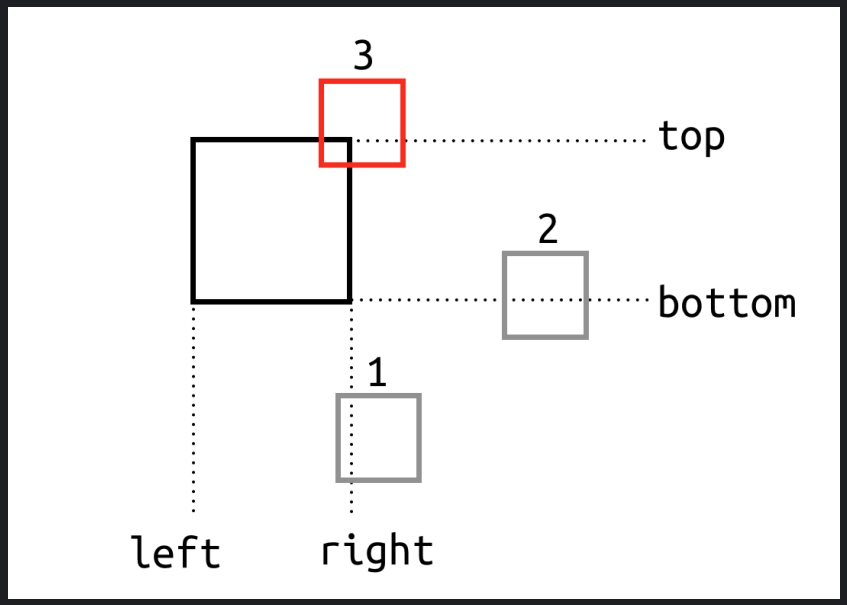
- Les joueurs se déplacent librement sur le terrain
- Un facteur d'aléa est rajouté afin de diversifier les simulations
- Les joueurs sont amenés à se bloquer dans les bordures

```
if y + b >= terrain_largeur - 55:
    y = terrain_largeur - 55
elif y + b <= 45:
    y = 45
else:
    y += b
self.coord = (x, y)
```

Déplacement bornés sur l'axe des ordonnées

Gestion des collisions et de la défense

- On utilise la méthode collide.rect prenant en argument 2 objets
- On évalue tous les couples de joueurs suscpetibles d'êtres en collision



Problèmes rencontrés lors des collisions

- Les rectangles de collision se chevauchent sur plusieurs frames.
- Cela occasionne des erreurs d'affichage et de gestion du ballon.

```
for joueur in self.joueurs:
    joueur.position(self.numero, self, equipe_adverse)
    joueur_adv = collision_joueurs(joueur, equipe_adverse)

    if joueur_adv is not None and un_contact_par_minute == 0:
        un_contact_par_minute = 1
        self.changer_porteur(joueur, joueur_adv, equipe_adverse)
```

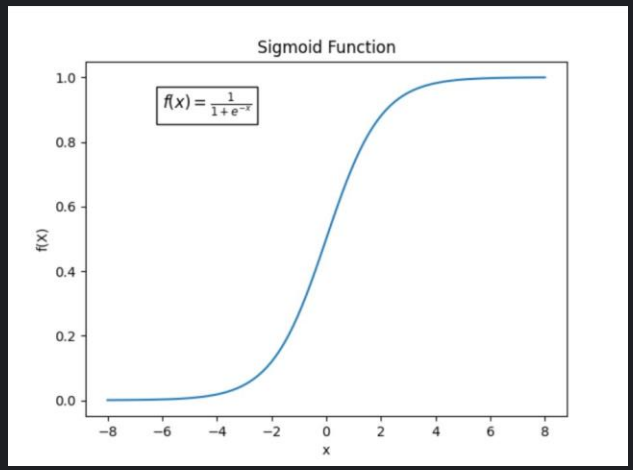
Implementation naive des tirs

Une fonction de répartition sera préférée dans la suite, afin d'obtenir un modèle probabiliste pus rigoureux.

```
def tir_au_but(self, equipe_num):
    """Le joueur est a distance raisonnable du but adverse"""
    distance_joueur_x, distance_joueur_y = self.coord
    if equipe_num == 1:
        distance_but = math.sqrt(
            (distance_joueur_x - but_equipe2_x) ** 2 + (distance_joueur_y - but_equipe2_y) ** 2)
    else:
        distance_but = math.sqrt(
            (distance_joueur_x - but_equipe1_x) ** 2 + (distance_joueur_y - but_equipe1_y) ** 2)

    return distance_but <= self.distance_tir
```

Ajustement de la probabilité de marquer

- Une approche efficiente et astucieuse du problème serait d'utiliser un algorithme de régression logistique
- Ce modèle permet de prédire la probabilité qu'un évènement arrive (valeur de 1, resp. 0)
- Un joueur se situant à l'infini ne devrait pas pouvoir marquer, respectivement pour un joueur proche du but.



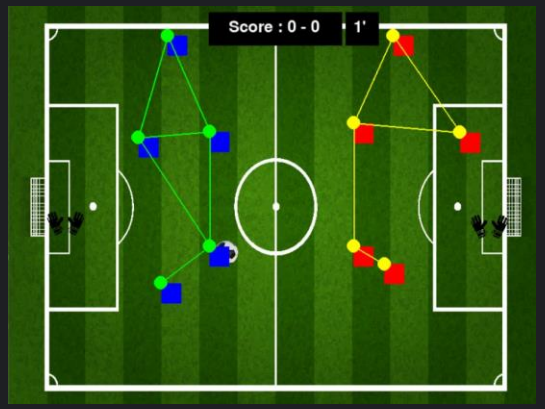

```python
def proba(tir, arret, dist):
    """Retourne la probabilité de marquer un but en fonction de la précision du tir,
    de la compétence du gardien et de la distance au but."""
    tir_norm = tir / 100
    arret_norm = arret / 100
    # Calcul de la base de la probabilité
    base_prob = (tir_norm - arret_norm) / (1 + dist / 50)
    # Transformation sigmoïde pour obtenir une probabilité entre 0 et 1
    prob = 1 / (1 + math.exp(-base_prob * 5))  # Ajustement de la sensibilité avec * 5
    # Ajustement final pour que la probabilité tende vers 0 quand la distance augmente
    adjusted_prob = prob * math.exp(-dist / 300)
    return adjusted_prob
```

Implémentation des compositions

- Avec trois joueurs en défense, chaque joueur peut couvrir une zone spécifique (gauche, centre, droite), réduisant ainsi les espaces disponibles pour les attaquants adverses.
- La défense en ligne permet une meilleure coordination et communication entre les joueurs pour éviter les passes et les mouvements de pénétration.

On sort de la boucle quand le meneur est trouvé

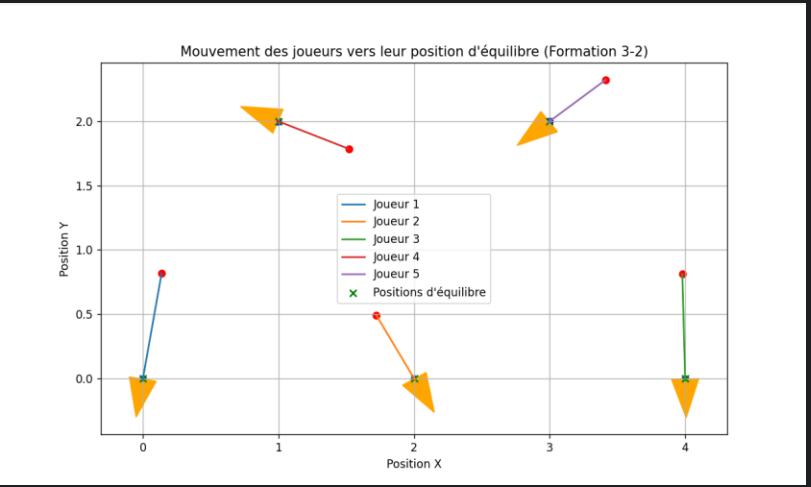
```
def motif(self):
    for j in self.joueurs:
        if j.poste == "atkg":
            lead = j.coord
            lead_x, lead_y = lead
            break
```



Mouvements des joueurs relativement à l'équipe

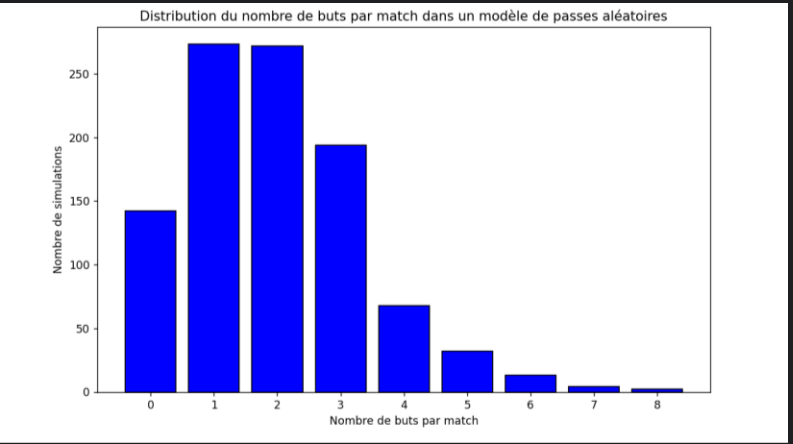
Modélisation classique de la force de rappel d'un ressort

```python
def simulation_mouvement(positions_initiales, positions_equilibre, num_steps=100, dt=0.1):
    def force_de_rappel(position, position_equilibre):
        return -k * (position - position_equilibre)
    positions = np.copy(positions_initiales)
    vitesses = np.zeros_like(positions)
    positions_historique = [np.copy(positions)]

    for _ in range(num_steps):
        for i in range(num_joueurs):
            force = force_de_rappel(positions[i], positions_equilibre[i])
            vitesses[i] += force * dt
            vitesses[i] *= (1 - damping)  # appliquer l'amortissement
            positions[i] += vitesses[i] * dt
        positions_historique.append(np.copy(positions))

    return np.array(positions_historique)
```



Implémentation du jeu en equipe : Passes aléatoires

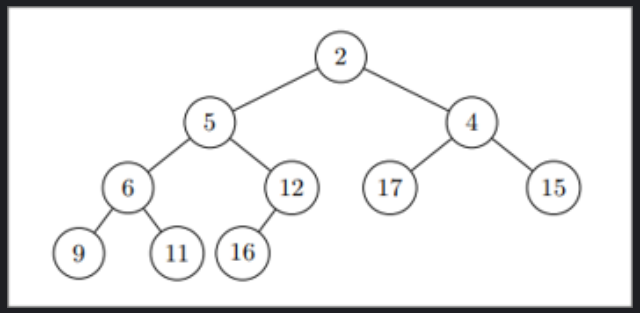
```python
def passer_balle(self, porteur):
    nouveau = porteur
    while(nouveau == porteur):
        nouveau = random.choice(self.joueurs)
    porteur.porteur_de_balle = False
    nouveau.porteur_de_balle = True
    return nouveau
```



- Les passes limitent les collisions
- Cet ajout a un rapport direct avec le nombre de buts inscrits par parties
- Le modèle aléatoire ne favorise pas la production d'occasions

Implémentation du jeu en équipe : joueur démarqué

- On associe à chaque équipe une file de priorité : chaque noeud est un joueur, dont la priorité est la distance qui le sépare du but adverse.
- Le joueur à la racine du tas est donc le plus proche du but adverse.



Tas où l'on a uniquement représenté les priorités

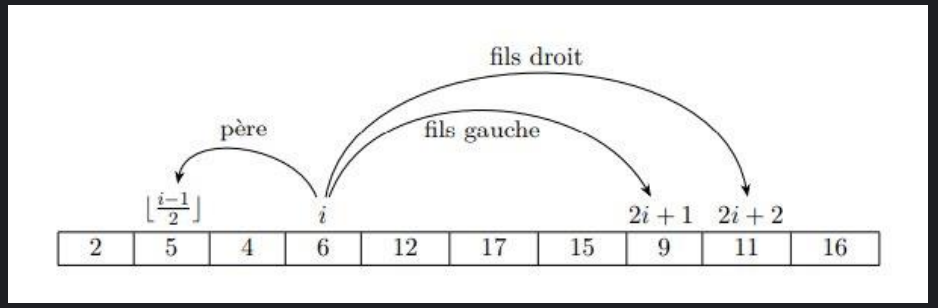
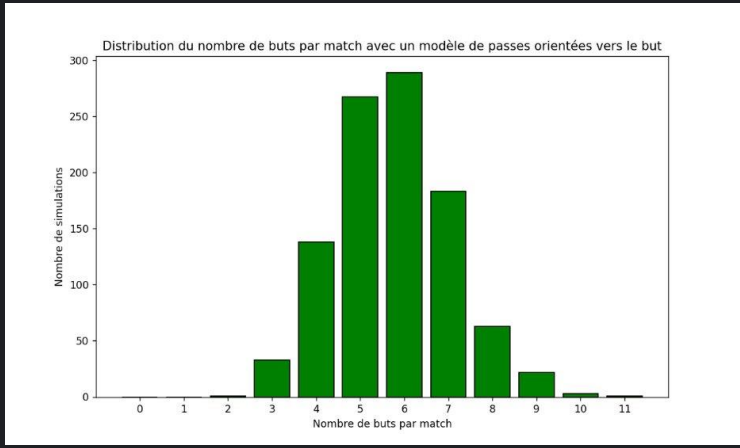


tableau stockant le tas

Implémentation du jeu en équipe : joueur démarqué

```python
def passer_balle(self, porteur):
    """donner la balle au joueur prioritaire"""
    prio = self.prio_equipe()
    meilleur_choix = prio.peek()
    if (porteur.passe // 100) > random.random():
        nouveau = self.joueurs[meilleur_choix - 1]
    else:
        nouveau = porteur
        while nouveau == porteur:
            nouveau = random.choice(self.joueurs)
    porteur.porteur_de_balle = False
    nouveau.porteur_de_balle = True
    return nouveau
```



- Prio_equipe() est une méthode renvoyant la file de priorité correspondante à l'équipe
- Le graphique montre une distribution plus large du nombre de buts marqués
- L'utilisation de cette stratégie doit s'utiliser avec parcimonie

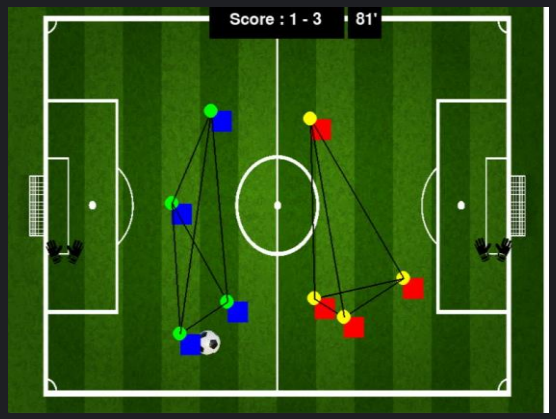
Implémentation du jeu en équipe : passe au joueur le plus proche

```python
def plus_proche(points, point_choisi):
    closest_point = None
    min_distance = float('inf')

    for point in points:
        d = distance(point, point_choisi)
        if d < min_distance:
            min_distance = d
            closest_point = point

    return closest_point
```

Cas le plus simple : on connait la position du porteur de balle.

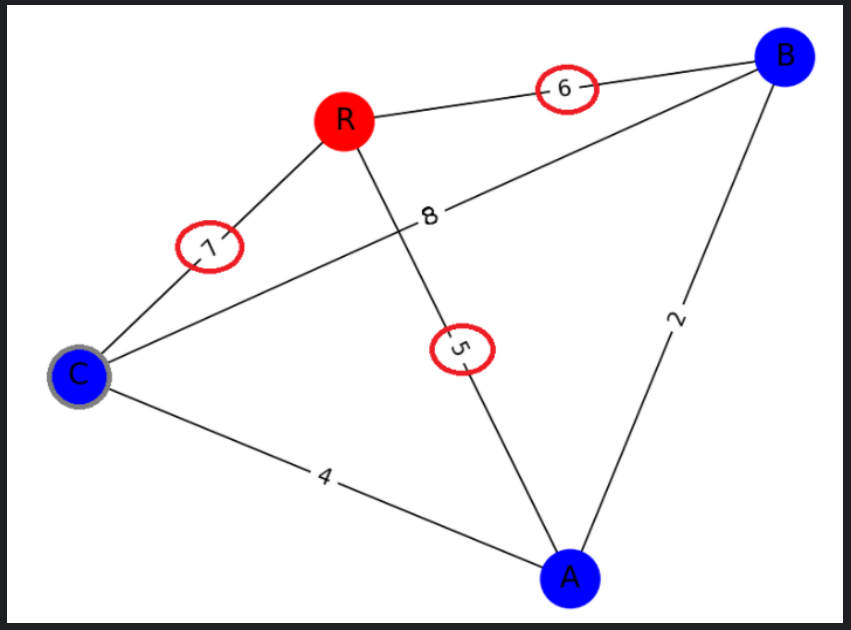
Implémentation moins naïve : chercher un joueur à distance raisonable et éloigné des joueurs adverses

On représente une équipe par un graphe complet pondéré par les distances séparant chaque joueur au sens de la norme euclidienne

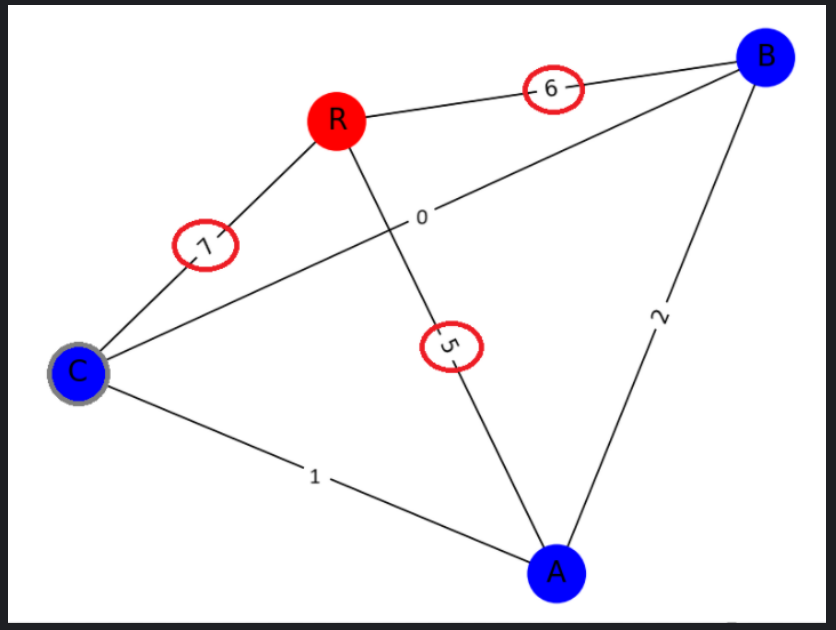


Passe au joueur libre

Poids = max(0, Poids_receveur – Poids_rouge)



Avant modification

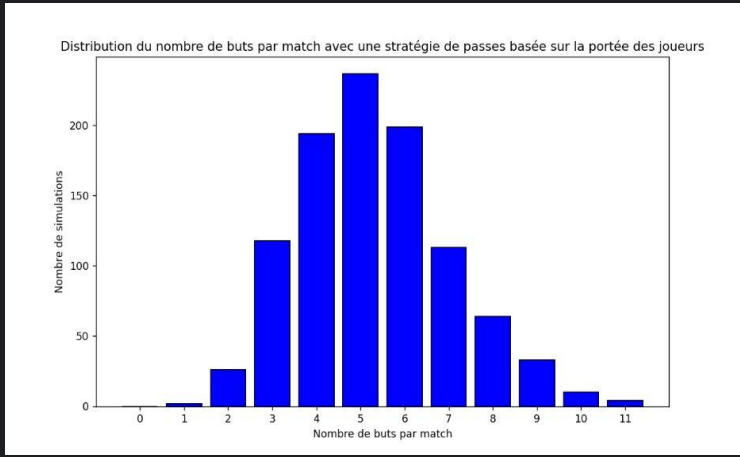


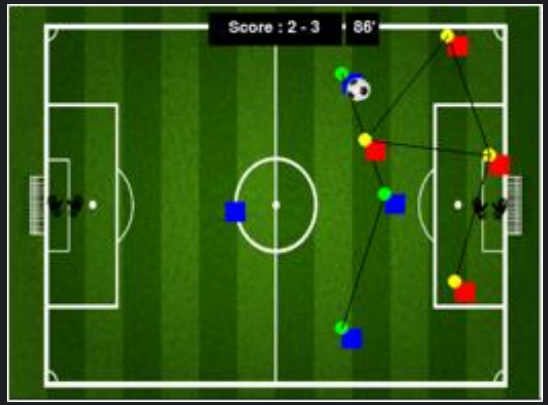
Après modification

Implémentation du jeu en équipe : Affichage et traitement des arêtes

On ne trace que les arêtes entre les joueurs qui ont une portée de passe suffisante

```
def remplir_graphe(self):
    g = Graph()
    alpha = 40
    g.set_graphe(self.joueurs)
    for i in range(len(self.joueurs)):
        for j in range(i + 1, len(self.joueurs)):
            weight = distance(self.joueurs[i].coord, self.joueurs[j].coord)
            if self.joueurs[i].passe + alpha > weight//2 + 20:
                g.add_edge(self.joueurs[i], self.joueurs[j], weight)
    return g
```





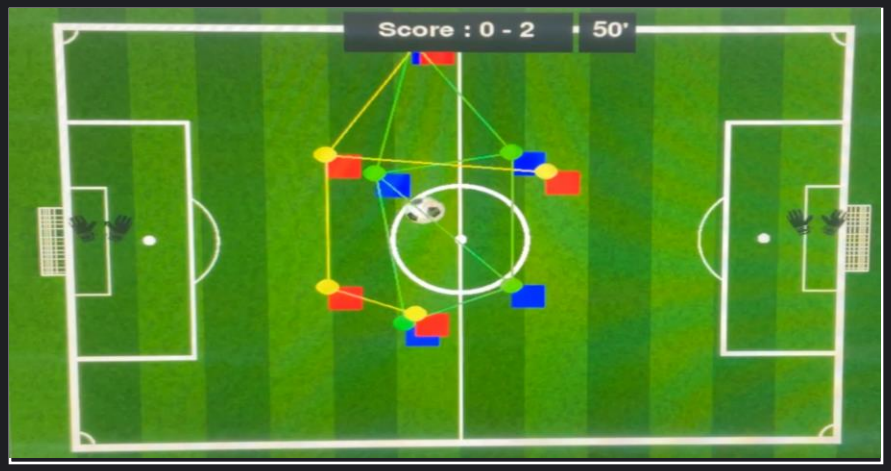
Implémentation du jeu en équipe : schéma de passes stratégiques

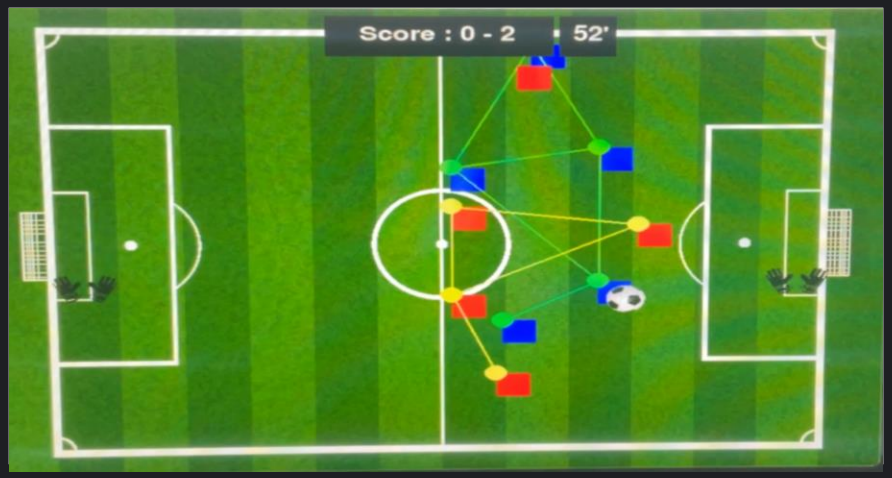
Algorithme 3 : Algorithme $A^\star$

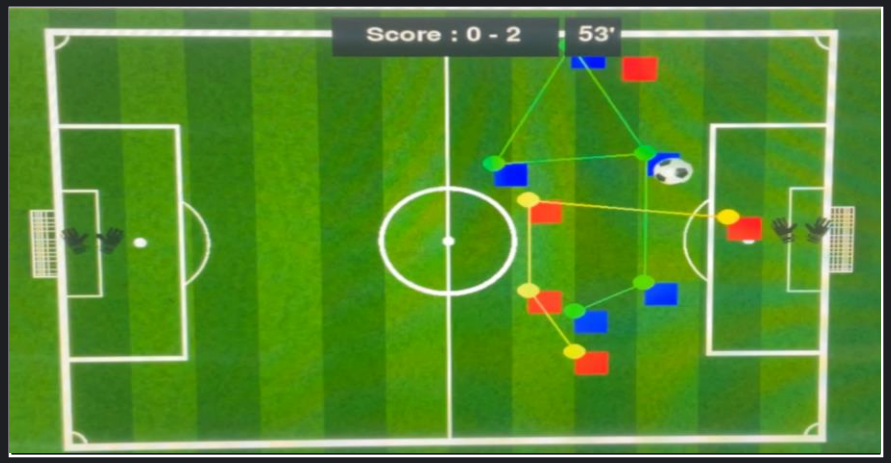
Données : Un graphe pondéré $G = (S, A, \omega)$ donné par listes d'adjacence avec $\omega(A) \subset \mathbb{R}_+$, deux sommets s et t, une heuristique h
Résultat : La distance $\delta(s,t)$
dist $\leftarrow [+\infty, \cdots, +\infty]$;
pred $\leftarrow [-1, \cdots, -1]$;
deja_vu $\leftarrow [Faux, \ldots, Faux]$;
dist$[s] \leftarrow 0$;
pred$[s] \leftarrow s$;
$F \leftarrow \{s\}$;
tant que $F \neq \emptyset$ **faire**
  $u \leftarrow$ Retirer de F un sommet v vérifiant $d_s[v] + h(v,t)$ minimal ;
  si $u = t$ **alors**
    retourner $d_s[t]$;
    pour *tout voisin* v *de* u **faire**
      si *dist*$[v] >$ *dist*$[u] + \omega(u,v)$ **alors**
        dist$[v] \leftarrow$ dist$[u] + \omega(u,v)$;
        pred$[v] = u$;
        si $v \notin F$ *et* *deja_vu*$[v]$ *est Faux* **alors**
          Ajouter v à F ;
    deja_vu$[u] \leftarrow Vrai$;
retourner $+\infty$ // pas de chemin entre s et t

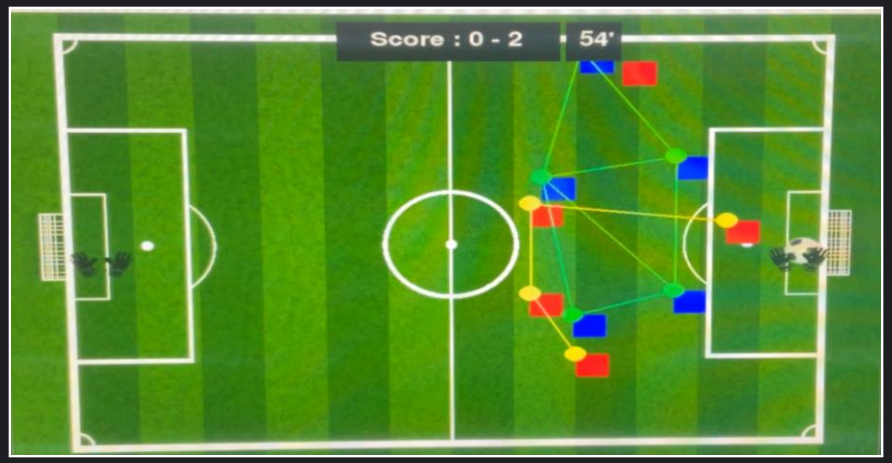
```
tas = self.prio_equipe()
chemin = g.a_star(porteur, tas.tasmin_pop())
if self.strategie.estVide() and chemin is not None:
    for joueur in chemin:
        self.strategie.enfiler(joueur)
else:
    self.strategie.defiler()
    nouveau = self.strategie.peek()
```

La logique d'enchaînement de passes utilise une file (self.strategie) pour gérer les joueurs dans une séquence déterminée par l'algorithme A*


Score : 0 - 2
50'


Score : 0 - 2
52'


Score : 0 - 2
53'


Score : 0 - 2
54'

Perspective d'amélioration et de développement futur

Implémenter une IA avancée pour les joueurs :

- Concevoir un système où l'IA s'adapte aux actions des joueurs et aux schémas tactiques de l'adversaire au fil du match.
- Par exemple, elle pourrait analyser les matchs précédents pour identifier les schémas de jeu efficaces et ajuster ses stratégies en conséquence
- Créer des modèles de comportement individuel pour chaque joueur contrôlé par l'IA
- Permettre à l'IA de s'adapter rapidement aux changements tactiques effectués par l'utilisateur ou l'adversaire pendant le match

Annexes :

```python
import math
import heapq

1 usage
class Graph:
    def __init__(self):
        self.graph = {}

    1 usage
    def set_graphe(self, liste):
        for joueur in liste:
            if joueur not in self.graph:
                self.graph[joueur] = []

    1 usage
    def add_edge(self, u, v, weight):
        if (v, weight) not in self.graph[u]:
            self.graph[u].append((v, weight))
        if (u, weight) not in self.graph[v]:
            self.graph[v].append((u, weight))

    1 usage
    def a_star(self, start, goal):
        open_set = []
        heapq.heappush( *args: open_set, (0, start))  # Ajouter le point de départ avec une priorité de 0
        came_from = {}
        g_score = {vertex: float('infinity') for vertex in self.graph}
        g_score[start] = 0
        f_score = {vertex: float('infinity') for vertex in self.graph}
        f_score[start] = self.heuristic(start, goal)
```

```
        while open_set:
            current = heapq.heappop(open_set)[1]

            if current == goal:
                path = self.reconstruct_path(came_from, start, goal)
                return path

            for neighbor, weight in self.graph[current]:
                tentative_g_score = g_score[current] + weight

                if tentative_g_score < g_score[neighbor]:
                    came_from[neighbor] = current
                    g_score[neighbor] = tentative_g_score
                    f_score[neighbor] = tentative_g_score + self.heuristic(neighbor, goal)

                    # Remplacez l'ajout dans open_set par le module heapq
                    heapq.heappush( *args: open_set, (f_score[neighbor], neighbor))

        return None
```

```
    def heuristic(self, playa1, playa2):
        # Heuristique basée sur la distance euclidienne
        x1, y1 = playa1.coord
        x2, y2 = playa2.coord
        return math.sqrt((x1 - x2)**2 + (y1 - y2)**2)

    1 usage
    def reconstruct_path(self, came_from, start, goal):
        current = goal
        path = [current]

        while current != start:
            current = came_from[current]
            path.insert( __index: 0, current)

        return path
```

```python
import random
import pygame
import time
from tasmin import *
from astar import *
import closest_point

pygame.init()
# Caractéristiques du terrain
terrain_longueur = 800  # en mètres
terrain_largeur = 600  # en mètres
largeur = 800
hauteur = 600
fenetre = pygame.display.set_mode((largeur, hauteur))
ballon_image = pygame.image.load("ballon.webp")
ballon_image = pygame.transform.scale(ballon_image, size: (50, 50))
pygame.display.set_caption("Simulation match")
terrain_image = pygame.image.load("terrain.jpg")
terrain_image = pygame.transform.scale(terrain_image, size: (800, 600))
panneau_image = pygame.image.load("panneau.png")
psg_img = pygame.image.load("psg.png")
psg_img = pygame.transform.scale(psg_img, size: (50, 50))
madrid_img = pygame.image.load("madrid.png")
madrid_img = pygame.transform.scale(madrid_img, size: (50, 50))
gardien_img = pygame.image.load("gardien.png")
gardien_img = pygame.transform.scale(gardien_img, size: (70, 70))
ballon_cord = (0, 0)
clock = pygame.time.Clock()
frame_rate = 10
```

```python
font = pygame.font.Font( name: None,  size: 36)
but_equipe1_x, but_equipe1_y = 40, terrain_largeur // 2
but_equipe2_x, but_equipe2_y = 730, terrain_largeur // 2
un_contact_par_minute = 0
score_rect = pygame.Rect(300, 10, 200, 50)  # Position et taille du rectangle
time_rect = pygame.Rect(505, 10, 50, 50)
tiempo = 0
pygame.mixer.init()

# Charger le fichier .wav
pygame.mixer.music.load('psg.wav')

# Jouer la musique en boucle
pygame.mixer.music.play(-1)


6 usages
def distance(point1, point2):
    return math.sqrt((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2)
```

```python
def collision_joueurs(joueur, equipe_adv):
    if joueur.porteur_de_balle:
        for joueur_adv in equipe_adv.joueurs:
            if joueur.rect.colliderect(joueur_adv.rect):
                if random.random() < 0.2 + (max(joueur.defense, joueur_adv.defense)) / 200:
                    if joueur.defense > joueur_adv.defense:
                        print("plus puissant\n")
                        return None
                    else:
                        print("défendu\n")
                        return joueur_adv
                print("récuperation classique\n")
                return joueur_adv
    else:
        return None


1 usage
def pos_balle(equipe_adv):
    if equipe_adv.get_porteur_de_balle() is not None:
        return equipe_adv.get_porteur_de_balle().coord
    else:
        return None
```

```python
def proba(tir, arret, dist):
    """Retourne la probabilité de marquer un but en fonction de la précision du tir,
    de la compétence du gardien et de la distance au but."""
    tir_norm = tir / 100
    arret_norm = arret / 100
    # Calcul de la base de la probabilité
    base_prob = (tir_norm - arret_norm) / (1 + dist / 50)
    # Transformation sigmoïde pour obtenir une probabilité entre 0 et 1
    prob = 1 / (1 + math.exp(-base_prob * 5))  # Ajustement de la sensibilité avec * 5
    # Ajustement final pour que la probabilité tende vers 0 quand la distance augmente
    adjusted_prob = prob * math.exp(-dist / 300)
    return adjusted_prob


2 usages
class Gardien:
    def __init__(self, reflexe, numero):
        self.reflexe = reflexe
        self.rect = pygame.Rect(0, 0, 30, 30)
        self.numero = numero
        if self.numero == 1:
            self.coord = (50, 295)
        else:
            self.coord = (685, 295)
```

```
98      def position(self, ballon_coord):
99          x, y = self.coord
100         b = random.uniform( a: 0,  b: 6)
101         # Gardien se déplace uniquement en Y en fonction de la position Y du ballon
102         if y < ballon_coord[1]:
103             if y > 330:
104                 y -= b
105             else:
106                 y += b
107         elif y > ballon_coord[1]:
108             if y < 245:
109                 y += b
110             else:
111                 y -= b
112         # Assurez-vous que le gardien reste dans les limites du terrain
113         y = max(10, min(y, terrain_largeur - 10))
114         self.coord = (x, y)
115         self.rect.x, self.rect.y = x, y
116         fenetre.blit(gardien_img,  dest: (self.rect.x, self.rect.y))
```

```python
class Joueur:
    def __init__(self, precision, vitesse, defense, passe, distance_tir, poste):
        self.precision_tir = precision
        self.vitesse = vitesse
        self.defense = defense
        self.coord = (0, 0)
        self.porteur_de_balle = False
        self.distance_tir = distance_tir
        self.passe = passe
        self.poste = poste
        self.rect = pygame.Rect(0, 0, 30, 30)

    2 usages (2 dynamic)
    def tir_au_but(self, equipe_num):
        """Le joueur est a distance raisonnable du but adverse"""
        distance_joueur_x, distance_joueur_y = self.coord
        if equipe_num == 1:
            distance_but = math.sqrt(
                (distance_joueur_x - but_equipe2_x) ** 2 + (distance_joueur_y - but_equipe2_y) ** 2)
        else:
            distance_but = math.sqrt(
                (distance_joueur_x - but_equipe1_x) ** 2 + (distance_joueur_y - but_equipe1_y) ** 2)

        return distance_but <= self.distance_tir
```

```python
    def position(self, num_equipe, equipe, equipe_adv):
        if self.poste == "atkg":
            vitesse = self.vitesse
            x, y = self.coord
            a = random.uniform( a: 0, vitesse - 20)
            b = random.uniform(-20,  b: 20)
            dist = 1000
            if equipe.equipe_balle:
                if num_equipe == 1:
                    if x + a >= terrain_longueur - 80:
                        x = terrain_longueur - 80
                    else:
                        x += a
                else:
                    if x - a <= 60:
                        x = 60
                    else:
                        x -= a
                if y + b >= terrain_largeur - 55:
                    y = terrain_largeur - 55
                elif y + b <= 45:
                    y = 45
                else:
                    y += b
                self.coord = (x, y)
                self.rect.x, self.rect.y = x, y
            else:
                equipe.position2(self, equipe_adv, dist)
            equipe.motif()
```

```python
class Equipe:
    def __init__(self, numero, joueurs, gardien):
        self.numero = numero
        self.joueurs = joueurs
        self.gardien = gardien
        self.score = 0
        self.equipe_balle = False
        self.strategie = File()

    2 usages
    def prio_equipe(self):
        """ donner la balle au joueur le plus proche du but adverse """
        global ballon_cord
        tas_prio = Tas()
        for joueur in self.joueurs:
            if joueur.porteur_de_balle:
                continue
            if self.numero == 2:
                dist = distance(joueur.coord,  point2: [but_equipe1_x, but_equipe1_y])
            else:
                dist = distance(joueur.coord,  point2: [but_equipe2_x, but_equipe2_y])
            tas_prio.tasmin_push(joueur, dist)
        return tas_prio
```

```python
def remplir_graphe(self):
    g = Graph()
    alpha = 40
    g.set_graphe(self.joueurs)
    for i in range(len(self.joueurs)):
        for j in range(i + 1, len(self.joueurs)):
            weight = distance(self.joueurs[i].coord, self.joueurs[j].coord)
            if self.joueurs[i].passe + alpha > weight//2 + 20:
                g.add_edge(self.joueurs[i], self.joueurs[j], weight)
    return g
```

5 usages (2 dynamic)

```python
def dessiner_arcs(self):
    g = self.remplir_graphe()
    for u in g.graph:
        for (v, weight) in g.graph[u]:
            pos1 = u.coord
            pos2 = v.coord
            if self.numero == 1:
                pygame.draw.circle(fenetre, color: (0,255,0), pos1, radius: 10)
                pygame.draw.line(fenetre, color: (0, 255, 0), pos1, pos2, width: 2)
            else:
                pygame.draw.circle(fenetre, color: (255, 255, 0), pos1, radius: 10)
                pygame.draw.line(fenetre, color: (255, 255, 0), pos1, pos2, width: 2)
```

```python
def afficher_joueurs(self):
    for joueur in self.joueurs:
        if self.numero == 2:
            pygame.draw.rect(fenetre, color: (255, 0, 0), joueur.rect)
            #fenetre.blit(psg_img, joueur.rect)
        else:
            pygame.draw.rect(fenetre, color: (0, 0, 255), joueur.rect)
            #fenetre.blit(madrid_img, joueur.rect)
```

```
221     def replacement(self):
222         for joueur in self.joueurs:
223             if joueur.poste == "atkg":
224                 if self.numero == 1:
225                     a, b = (338, 167)
226                 else:
227                     a, b = (462, 167)
228                 joueur.coord = a, b
229                 joueur.rect.x = a
230                 joueur.rect.y = b
231             elif joueur.poste == "atkd":
232                 if self.numero == 1:
233                     a, b = (354, 391)
234                 else:
235                     a, b = (462, 391)
236                 joueur.coord = a, b
237                 joueur.rect.x = a
238                 joueur.rect.y = b
239         lead = a, b
```

```
240     for j in self.joueurs:
241         if j.poste != "atkg" or j.poste != "atkd":
242             x, y = lead
243             if self.numero == 1:
244                 if j.poste == "def":
245                     x -= 150
246                     y += 200
247                 x -= 100
248                 if j.poste == "midd":
249                     y += 200
250                 else:
251                     y -= 200
252             else:
253                 if j.poste == "def":
254                     x += 125
255                     y += 200
256                 x += 60
257                 if j.poste == "midd":
258                     y += 200
259                 else:
260                     y -= 200
261             j.coord = (x, y)
262             j.rect.x, j.rect.y = x, y
```

```
272     def animation(self, passeur, destinataire, equipe_adverse, but):
273         global ballon_cord
274         start_x, start_y = passeur.coord  # La position du joueur qui a tiré
275         if destinataire is not None:
276             end_x, end_y = int(destinataire.coord[0]), int(destinataire.coord[1])
277         else:
278             if self.numero == 1:
279                 end_x, end_y = but_equipe2_x, but_equipe2_y
280             else:
281                 end_x, end_y = but_equipe1_x, but_equipe1_y
282         steps = 25  # Nombre d'étapes pour l'animation de la trajectoire du tir
283         dx = (end_x - start_x) / steps
284         dy = (end_y - start_y) / steps
285         for _ in range(steps):
286             start_x += dx
287             start_y += dy
288             ballon_cord = (start_x, start_y)
289             fenetre.blit(terrain_image, dest: (0, 0))  # Effacez l'image précédente
290             fenetre.blit(ballon_image, dest: (ballon_cord[0], ballon_cord[1]))  # Affichez le ballon
291             self.afficher_joueurs()
292             equipe_adverse.afficher_joueurs()
293             self.gardien.position(ballon_cord)
294             equipe_adverse.gardien.position(ballon_cord)
295             self.dessiner_arcs()
296             equipe_adverse.dessiner_arcs()
```

```
            if self.numero == 1:
                score_text = font.render( text: f"Score : {self.score} - {equipe_adverse.score}", antialias: True, color: (255, 255, 255))
            else:
                score_text = font.render( text: f"Score : {equipe_adverse.score} - {self.score}", antialias: True, color: (255, 255, 255))
            time_text = font.render( text: f"{tiempo//2}'", antialias: True, color: (255, 255, 255))
            pygame.draw.rect(fenetre, color: (0, 0, 0), score_rect)  # Couleur du rectangle (ici noir)
            pygame.draw.rect(fenetre, color: (0, 0, 0), time_rect)
            fenetre.blit(score_text, dest: (330, 20))
            fenetre.blit(time_text, dest: (517, 20))

            pygame.display.update()
            if destinataire is None:
                pygame.time.delay(15)  # Délai pour l'animation
            else:
                pygame.time.delay(20)
        if not but and destinataire is None:
            if equipe_adverse.numero == 2:
                ballon_cord = 700, equipe_adverse.gardien.coord[1]
            else:
                ballon_cord = equipe_adverse.gardien.coord
            fenetre.blit(terrain_image, dest: (0, 0))  # Effacez l'image précédente
            fenetre.blit(ballon_image, dest: (ballon_cord[0], ballon_cord[1]))  # Affichez le ballon
            self.afficher_joueurs()
            equipe_adverse.afficher_joueurs()
            self.gardien.position(ballon_cord)
            equipe_adverse.gardien.position(ballon_cord)
            self.dessiner_arcs()
            equipe_adverse.dessiner_arcs()
```

```
            if self.numero == 1:
                score_text = font.render( text: f"Score : {self.score} - {equipe_adverse.score}", antialias: True, color: (255, 255, 255))
            else:
                score_text = font.render( text: f"Score : {equipe_adverse.score} - {self.score}", antialias: True, color: (255, 255, 255))
            time_text = font.render( text: f"{tiempo//2}'", antialias: True, color: (255, 255, 255))
            pygame.draw.rect(fenetre, color: (0, 0, 0), score_rect)  # Couleur du rectangle (ici noir)
            pygame.draw.rect(fenetre, color: (0, 0, 0), time_rect)
            fenetre.blit(score_text, dest: (330, 20))
            fenetre.blit(time_text, dest: (517, 20))
            pygame.display.update()
            time.sleep(1)
    4 usages (4 dynamic)
    def get_porteur_de_balle(self):
        for joueur in self.joueurs:
            if joueur.porteur_de_balle:
                return joueur
        return None

    3 usages
    def changer_porteur(self, joueur, joueur_adv, equipe_adverse):
        joueur.porteur_de_balle = False
        self.equipe_balle = False
        joueur_adv.porteur_de_balle = True
        equipe_adverse.equipe_balle = True
```

```
def mouvements(self, equipe_adverse):
    global ballon_cord
    global un_contact_par_minute
    but = False

    # Vérifier s'il y a déjà un porteur de balle dans l'équipe

    for joueur in self.joueurs:
        joueur.position(self.numero, self, equipe_adverse)
        joueur_adv = collision_joueurs(joueur, equipe_adverse)

        if joueur_adv is not None and un_contact_par_minute == 0:
            un_contact_par_minute = 1
            self.changer_porteur(joueur, joueur_adv, equipe_adverse)

        if random.random() < 0.5 and joueur.porteur_de_balle:
            porteur = self.passer_balle(joueur)
            if porteur != None:
                self.animation(joueur, porteur, equipe_adverse, but)

        if joueur.porteur_de_balle and joueur.tir_au_but(self.numero):
            if self.numero == 1:
                prob = proba(joueur.precision_tir, equipe_adverse.gardien.reflexe, distance(joueur.coord, point2: [but_equipe2_x, but_equipe2_y]))
            else:
                prob = proba(joueur.precision_tir, equipe_adverse.gardien.reflexe,
                             distance(joueur.coord, point2: [but_equipe1_x, but_equipe1_y]))
            if prob > random.random():
                but = True
                self.score += 1
                print(f"But du joueur {self.joueurs.index(joueur) + 1} pour l'équipe {self.numero} !")
                self.animation(joueur, destinataire: None, equipe_adverse, but)
            else:
                but = False
                self.animation(joueur, destinataire: None, equipe_adverse, but)
                joueur_adv = random.choice(equipe_adverse.joueurs)
                self.changer_porteur(joueur, joueur_adv, equipe_adverse)
                self.replacement()
                equipe_adverse.replacement()

    self.afficher_joueurs()
    self.dessiner_arcs()
    for joueur in self.joueurs:
        if joueur.porteur_de_balle:
            ballon_cord = (joueur.coord[0], joueur.coord[1])
            fenetre.blit(ballon_image, dest: (ballon_cord[0], ballon_cord[1]))
    self.gardien.position(ballon_cord)
    return but
```

```
401     def passer_balle(self, porteur):
402         """donner la balle au joueur prioritaire"""
403         g = self.remplir_graphe()
404         prio = self.prio_equipe()
405         meilleur_choix = prio.peek()
406         liste = g.graph[porteur]
407         nouveau = None
408         joueurs = [joueur for joueur, w in liste]
409         if (porteur.passe // 400) > random.random():
410             nouveau = meilleur_choix
411         elif joueurs != []:
412             # choix destinataire aléatoire
413             #nouveau = random.choice(joueurs)
414             # passe au plus proche
415             """coord = [joueur.coord for joueur in joueurs]
416             joueur_proche_coord = closest_point.closest_point_to_given_point(coord, porteur.coord)
417             nouveau = [joueur for joueur in joueurs if joueur.coord == joueur_proche_coord][0]"""
418             # enchainement de passes
419             alea = random.random()
420             if alea + 80 > random.random():
421                 tas = self.prio_equipe()
422                 chemin = g.a_star(porteur, tas.tasmin_pop())
423                 if self.strategie.estVide() and chemin is not None:
424                     for joueur in chemin:
425                         self.strategie.enfiler(joueur)
426                 else:
427                     self.strategie.defiler()
428                     nouveau = self.strategie.peek()
429             else:
430                 coord = [joueur.coord for joueur in joueurs]
431                 joueur_proche_coord = closest_point.closest_point_to_given_point(coord, porteur.coord)
432                 nouveau = [joueur for joueur in joueurs if joueur.coord == joueur_proche_coord][0]
```

```
433         else:
434             nouveau = None
435         if nouveau != None:
436             porteur.porteur_de_balle = False
437             nouveau.porteur_de_balle = True
438         return nouveau
```

```
    def motif(self):
        for j in self.joueurs:
            if j.poste == "atkg":
                lead = j.coord
                lead_x, lead_y = lead
                break

        for j in self.joueurs:
            if j.poste != "atkg":
                x, y = lead_x, lead_y  # Initialiser avec les coordonnées du meneur
                a = random.uniform( a: 0, j.vitesse - 20)
                b1 = random.uniform( a: 0,  b: 20)
                b2 = random.uniform(-20,  b: 0)

                if self.numero == 1:
                    if j.poste == "def":
                        x -= 150 - a  # Augmenter l'aléa
                        y += 200 + b1  # Augmenter l'aléa
                        if y >= terrain_largeur - 55:
                            y = terrain_largeur - 55
                    elif j.poste != "atkd":
                        x -= 100 - a  # Augmenter l'aléa
                    if x > terrain_longueur - 80:
                        x = terrain_longueur - 80
                    if x < 60:
                        x = 60
                    if j.poste == "midd":
                        y += 210 + b1  # Augmenter l'aléa
                        if y >= terrain_largeur - 55:
                            y = terrain_largeur - 55
                    else:
                        y -= 200 + b2  # Augmenter l'aléa
                        if y <= 45:
                            y = 45
                else:
                    if j.poste == "def":
                        x += 150 + a  # Augmenter l'aléa
                        y += 200 + b1  # Augmenter l'aléa
                        if y >= terrain_largeur - 55:
                            y = terrain_largeur - 55
                    elif j.poste != "atkd":
                        x += 40 + a  # Augmenter l'aléa
                    if x > terrain_longueur - 80:
                        x = terrain_longueur - 80
                    if x < 60:
                        x = 60
                    if j.poste == "midd":
                        y += 200 + b1  # Augmenter l'aléa
                        if y >= terrain_largeur - 55:
                            y = terrain_largeur - 55
                    else:
                        y -= 200 + b2  # Augmenter l'aléa
                        if y <= 45:
                            y = 45
                if j.poste == "atkd":
                    y += 300 + b1
                    if y >= terrain_largeur - 55:
                        y = terrain_largeur - 55
                    if y <= 45:
                        y = 45
                j.coord = (x, y)
                j.rect.x, j.rect.y = x, y
```

```python
    def position2(self, joueur, equipe_adv, dist_max):
        if joueur.poste == "atkg":
            x, y = joueur.coord
            a = random.uniform( a: 0, joueur.vitesse - 20)
            b1 = random.uniform( a: 0,  b: 20)
            b2 = random.uniform(-20,  b: 0)
            ballon = pos_balle(equipe_adv)
            if distance(joueur.coord, ballon) < dist_max:
                if ballon[0] > joueur.coord[0]:
                    if x + a >= terrain_longueur - 80:
                        x = terrain_longueur - 80
                    else:
                        x += a
                else:
                    if x - a <= 60:
                        x = 60
                    else:
                        x -= a
                if ballon[1] > joueur.coord[1]:
                    if y + b1 >= terrain_largeur - 55:
                        y = terrain_largeur - 55
                    else:
                        y += b1
                else:
                    if y + b2 <= 45:
                        y = 45
                    else:
                        y += b2
                joueur.coord = (x, y)
                joueur.rect.x, joueur.rect.y = x, y
```

```python
equipe1 = Equipe( numero: 1,  joueurs: [Joueur( precision: 90,  vitesse: 71,  defense: 65,  passe: 80,  distance_tir: 180,  poste: "at
                  Joueur( precision: 87,  vitesse: 71,  defense: 67,  passe: 80,  distance_tir: 180,  poste: "def"), Joueur( prec
equipe2 = Equipe( numero: 2,  joueurs: [Joueur( precision: 81,  vitesse: 65,  defense: 69,  passe: 80,  distance_tir: 180,  poste: "de
                  Joueur( precision: 87,  vitesse: 71,  defense: 67,  passe: 100,  distance_tir: 180,  poste: "atkg"), Joueur( p
equipe1.replacement()
equipe2.replacement()
equipe1.joueurs[0].porteur_de_balle = True
equipe1.equipe_balle = True


while True:
    tiempo += 1
    fenetre.blit(terrain_image,  dest: (0, 0))
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()

    but_equipe1 = equipe1.mouvements(equipe2)
    if but_equipe1:
        equipe1.replacement()
        equipe2.replacement()
    but_equipe2 = equipe2.mouvements(equipe1)
    un_contact_par_minute = 0
    if but_equipe2:
        equipe1.replacement()
        equipe2.replacement()
    mouse_x, mouse_y = pygame.mouse.get_pos()
    if tiempo >= 183 + random.randint( a: 2,  b: 6) and equipe2.score - equipe1.score != 0:
        break
```

```
565         # Afficher les coordonnées de la souris dans la console
566         #print(f"Position de la souris : ({mouse_x}, {mouse_y})")
567         score_text = font.render( text: f"Score : {equipe1.score} - {equipe2.score}",  antialias: True,  color: (255, 255, 255))
568         time_text = font.render( text: f"{tiempo // 2}'",  antialias: True,  color: (255, 255, 255))
569         pygame.draw.rect(fenetre,  color: (0, 0, 0), score_rect)  # Couleur du rectangle (ici noir)
570         pygame.draw.rect(fenetre,  color: (0, 0, 0), time_rect)
571         fenetre.blit(score_text,  dest: (330, 20))
572         fenetre.blit(time_text,  dest: (517, 20))
573         pygame.display.update()
574         clock.tick(frame_rate)
575 
576     score_equipe1, score_equipe2 = equipe1.score, equipe2.score
577     print(f"Score final : {score_equipe1} - {score_equipe2}")
578     time.sleep(8)
579     pygame.mixer.music.stop()
580     pygame.quit()
581 
```