1- Mini Campo de Futebol com Mapeamento de Jogadores (TrueField)

2- Equipe:

- Cauã Borges Faria(834437)
- Vitor Rodrigues da Mata (831591)
- Yuri Arita de Carvalho (832916)

5. Principais Unidades de Software

O projeto desenvolvido consiste em um sistema de simulação de partida de futebol em 2D utilizando Python, Pygame, e visualização de mapas de calor. As principais unidades de software implementadas são:

Estrutura de Classes e Componentes:

```
                                          |
+-------------------------------------------------+
|                    Heatmap                      |
+-------------------------------------------------+
| + generate_heatmap()                            |

+-------------------------------------------------+
```

Destaques:

- **Campo**: Responsável pelo desenho do campo, detecção de gols e verificação se a bola está em campo.
- **Jogador**: Representa cada jogador com número, posição e métodos para movimentação.
- **Bola**: Controla posição e movimentação da bola.
- **Heatmap**: Gera visualizações de calor a partir do histórico de posições dos jogadores durante a simulação.
- **Quadtree**: Estrutura de dados eficiente para particionar as posições no campo e otimizar a consulta de pontos para o mapa de calor final.

A principal contribuição técnica está na utilização da **Quadtree** para particionar espacialmente as posições coletadas durante o jogo, reduzindo o custo de consulta e atualizações em regiões específicas do campo.

---

6. Andamento: O que já foi feito e o que falta fazer

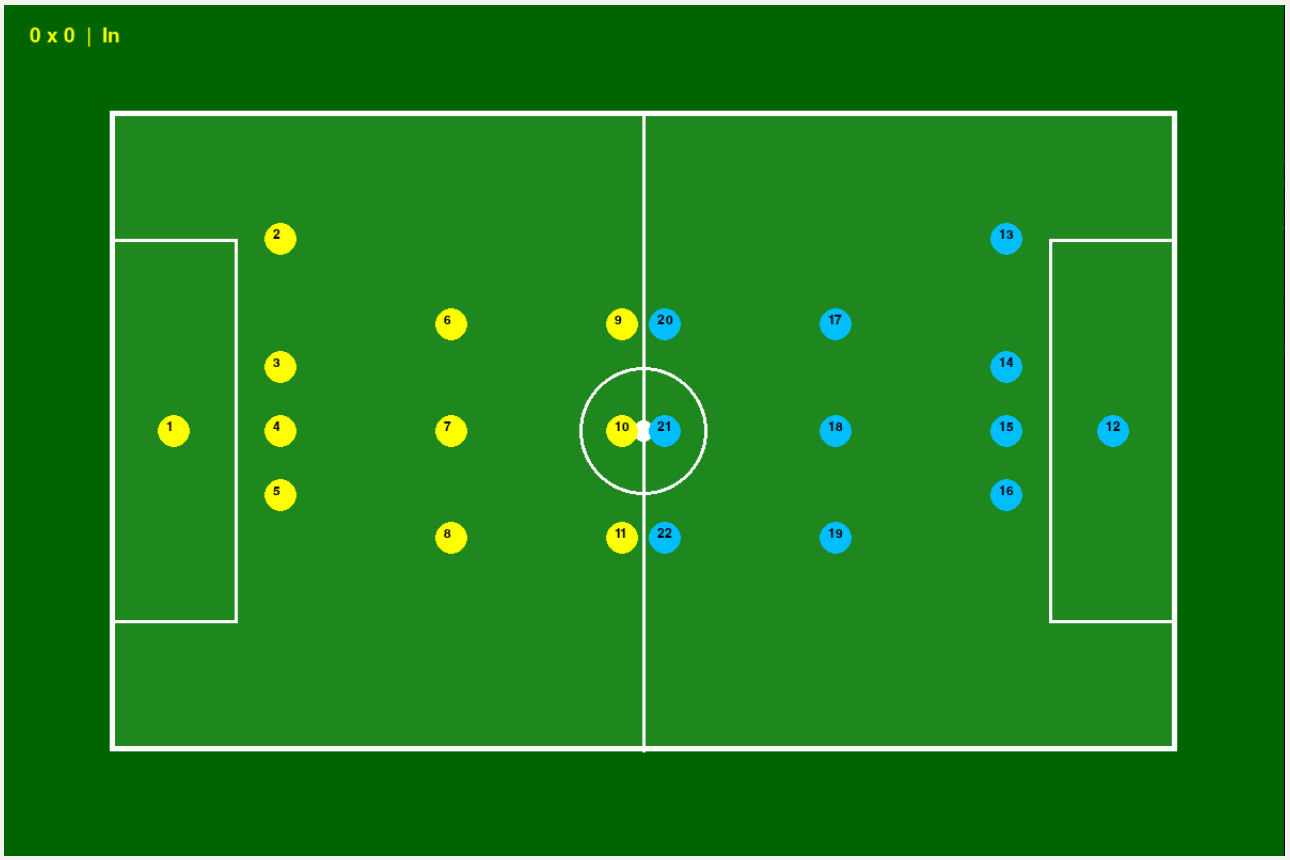
O que já foi desenvolvido:

- **Simulação gráfica completa** de uma partida em 2D usando Pygame.
- **Implementação das classes Campo, Jogador e Bola** com movimentação interativa.
- **Sistema de detecção de gol** com verificação das laterais e da posição da bola.
- **Geração de mapa de calor dinâmico em tempo real** exibido sobre o campo durante a execução.

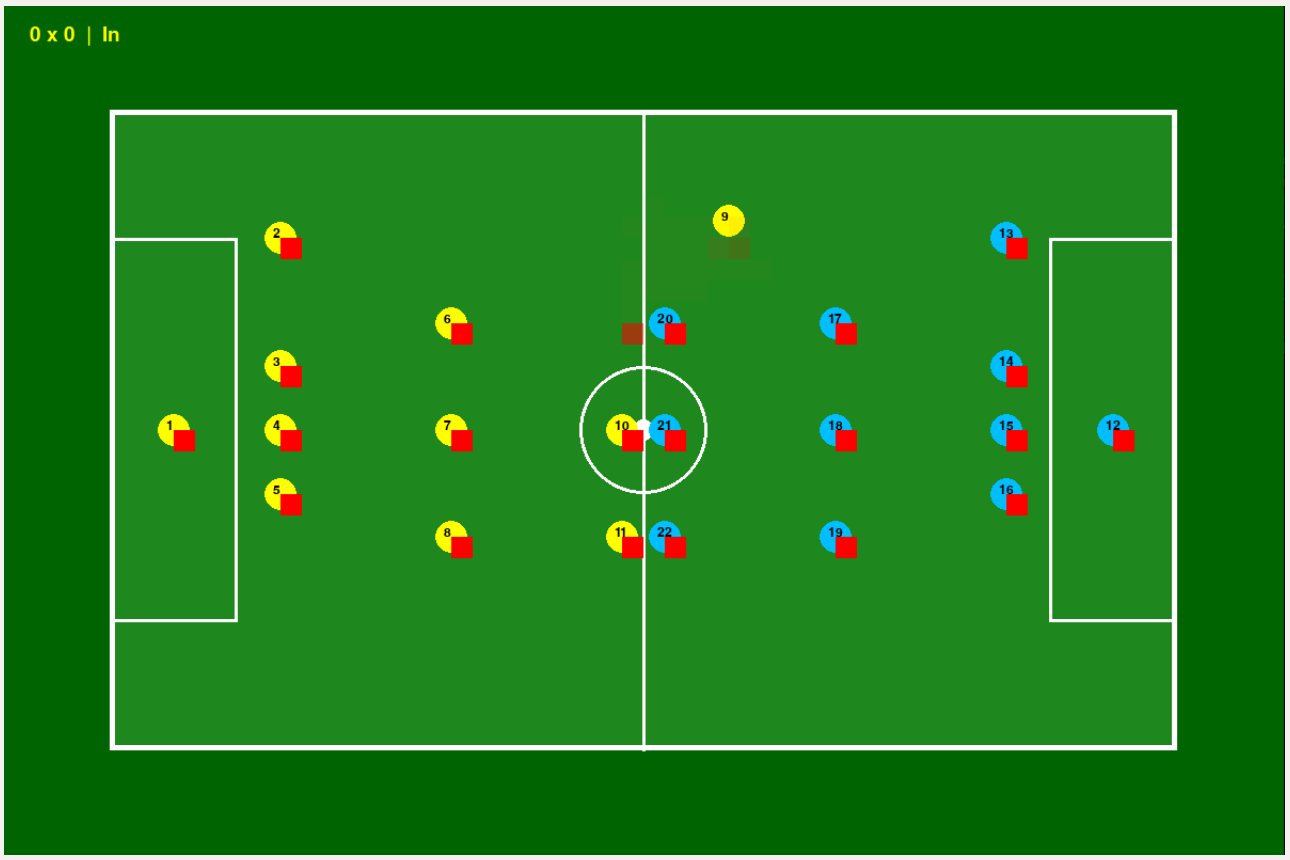
- **Opção de exibir mapa geral ou individual por jogador**, alternando com a tecla `H` e `N`.
- **Histórico de posições armazenado** a cada frame.
- **Uso de Quadtree** para organizar as posições da bola e jogadores, permitindo consultas otimizadas na geração do heatmap final.
- **Geração e salvamento de heatmap final** utilizando Seaborn e Matplotlib ao final da simulação.

Prints da Execução:

Campo de Jogo Simulado:



Mapa de Calor Dinâmico:



O que falta fazer:

- **Melhorar heatmap**
- **Opcional: Implementar suporte para simular diferentes padrões de movimentação de jogadores automaticamente** (ex: zonas de ataque, defesa e marcação).

---

7 - Implementação - Estágio Atual

```
import pygame
from pygame.locals import *

class Quadtree:
    def __init__(self, boundary, capacity):
        self.boundary = boundary
        self.capacity = capacity
```

```
        self.points = []
        self.divided = False

    def subdivide(self):
        x, y, w, h = self.boundary
        hw, hh = w // 2, h // 2
        self.nw = Quadtree(pygame.Rect(x, y, hw, hh),
self.capacity)
        self.ne = Quadtree(pygame.Rect(x + hw, y, hw, hh),
self.capacity)
        self.sw = Quadtree(pygame.Rect(x, y + hh, hw, hh),
self.capacity)
        self.se = Quadtree(pygame.Rect(x + hw, y + hh, hw, hh),
self.capacity)
        self.divided = True

    def insert(self, point):
        if not self.boundary.collidepoint(point):
            return False
        if len(self.points) < self.capacity:
            self.points.append(point)
            return True
        if not self.divided:
            self.subdivide()
        return (self.nw.insert(point) or self.ne.insert(point) or
self.sw.insert(point) or self.se.insert(point))

    def query(self, range_rect, found=None):
        if found is None:
            found = []
        if not self.boundary.colliderect(range_rect):
            return found
        for p in self.points:
            if range_rect.collidepoint(p):
                found.append(p)
        if self.divided:
            self.nw.query(range_rect, found)
            self.ne.query(range_rect, found)
            self.sw.query(range_rect, found)
            self.se.query(range_rect, found)
```

```
        return found

    def clear(self):
        self.points = []
        if self.divided:
            self.nw.clear()
            self.ne.clear()
            self.sw.clear()
            self.se.clear()
        self.divided = False



SCREEN_WIDTH, SCREEN_HEIGHT = 1200, 800
FIELD_WIDTH, FIELD_HEIGHT = 1000, 600
BALL_RADIUS, PLAYER_RADIUS = 10, 15
BALL_SPEED, PLAYER_SPEED = 3, 4
SLICE_TIME = 100

class Campo:
    def __init__(self, x, y, w, h):
        self.rect = pygame.Rect(x, y, w, h)
        self.quadtree = Quadtree(self.rect, 4)

    def draw(self, surface):
        pygame.draw.rect(surface, (34,139,34), self.rect)
        pygame.draw.rect(surface, (255,255,255), self.rect, 5)
        cx, cy = self.rect.center
        pygame.draw.circle(surface, (255,255,255), (cx,cy), 60, 3)
        pygame.draw.line(surface, (255,255,255), (cx,
self.rect.top), (cx, self.rect.bottom), 3)
        goal_area_w, goal_area_h = 120, 360
        pygame.draw.rect(surface, (255,255,255),
pygame.Rect(self.rect.left, cy-goal_area_h//2, goal_area_w,
goal_area_h), 3)
        pygame.draw.rect(surface, (255,255,255),
pygame.Rect(self.rect.right-goal_area_w, cy-goal_area_h//2,
goal_area_w, goal_area_h), 3)

    def verificar_gol(self, bola):
```

```
        goal_area_h = 360
        cy = self.rect.centery
        esquerda = pygame.Rect(self.rect.left-1, cy-goal_area_h//2,
1, goal_area_h)
        direita  = pygame.Rect(self.rect.right, cy-goal_area_h//2,
1, goal_area_h)
        ball_rect = pygame.Rect(bola.x-BALL_RADIUS, bola.y-
BALL_RADIUS, BALL_RADIUS*2, BALL_RADIUS*2)

        if ball_rect.right <= self.rect.left:
            if esquerda.top <= ball_rect.centery <=
esquerda.bottom:
                return "direita"
            else:
                return "Out"
        elif ball_rect.left >= self.rect.right:
            if direita.top <= ball_rect.centery <= direita.bottom:
                return "esquerda"
            else:
                return "Out"
        elif not self.is_ball_inside(bola):
            return "Out"
        else:
            return "In"

    def is_ball_inside(self, ball):
        ball_rect = pygame.Rect(
            ball.x - BALL_RADIUS, ball.y - BALL_RADIUS, BALL_RADIUS
* 2, BALL_RADIUS * 2
        )
        return self.rect.colliderect(ball_rect)

class Bola:
    def __init__(self, x, y):
        self.x, self.y = x, y

    def draw(self, surface):
        pygame.draw.circle(surface, (255,255,255), (int(self.x),
int(self.y)), BALL_RADIUS)
```

```
    def move(self, dx, dy):
        self.x += dx
        self.y += dy

class Jogador:
    def __init__(self, x, y, numero, cor):
        self.x, self.y = x, y
        self.numero, self.cor = numero, cor
        self.active = False

    def draw(self, surface):
        pygame.draw.circle(surface, self.cor, (int(self.x),
int(self.y)), PLAYER_RADIUS)
        font = pygame.font.SysFont(None, 18)
        num = font.render(str(self.numero), True, (0,0,0))
        surface.blit(num, (self.x-7, self.y-10))

    def move(self, dx, dy, campo):
        new_x = min(max(self.x+dx, campo.rect.left+PLAYER_RADIUS),
campo.rect.right-PLAYER_RADIUS)
        new_y = min(max(self.y+dy, campo.rect.top+PLAYER_RADIUS),
campo.rect.bottom-PLAYER_RADIUS)
        self.x, self.y = new_x, new_y

def generate_heatmap(surface, player_positions_history, field_rect,
cell_size=20, target_player_id=None):
    """Gera e desenha um mapa de calor das posições dos
jogadores."""
    heatmap_data = {}
    max_count = 0

    for x in range(field_rect.left, field_rect.right, cell_size):
        for y in range(field_rect.top, field_rect.bottom,
cell_size):
            heatmap_data[(x, y)] = 0

    for frame_positions in player_positions_history:
        for player_id, (px, py) in frame_positions.items():
            if target_player_id is None or player_id ==
target_player_id:
```

```
                cell_x = int((px - field_rect.left) // cell_size) *
cell_size + field_rect.left
                cell_y = int((py - field_rect.top) // cell_size) *
cell_size + field_rect.top
                cell_x = max(field_rect.left, min(cell_x,
field_rect.right - cell_size))
                cell_y = max(field_rect.top, min(cell_y,
field_rect.bottom - cell_size))
                heatmap_data[(cell_x, cell_y)] =
heatmap_data.get((cell_x, cell_y), 0) + 1
                if heatmap_data[(cell_x, cell_y)] > max_count:
                    max_count = heatmap_data[(cell_x, cell_y)]

    for (x, y), count in heatmap_data.items():
        if count > 0:
            alpha = int(255 * (count / max_count)) if max_count > 0
else 0
            pygame.draw.rect(surface, (255, 0, 0, alpha), (x, y,
cell_size, cell_size))



def main():
    pygame.init()
    screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
    pygame.display.set_caption("Truefield")
    clock = pygame.time.Clock()
    campo = Campo((SCREEN_WIDTH-FIELD_WIDTH)//2, (SCREEN_HEIGHT-
FIELD_HEIGHT)//2, FIELD_WIDTH, FIELD_HEIGHT)
    bola = Bola(SCREEN_WIDTH//2, SCREEN_HEIGHT//2)

    placar = {"esquerda":0, "direita":0}
    last_slice = pygame.time.get_ticks()

    running,selected_player = True, None
    jogadores = []
    num = 1

    # Time da esquerda (esquerda para direita)
    # Goleiro
```

```
    jogadores.append(Jogador(campo.rect.left + 60,
campo.rect.centery, num, (255,255,0)))
    num += 1
    # Defesa (4)
    y_def = [
        campo.rect.top + 120,
        campo.rect.top + 240,
        campo.rect.centery,
        campo.rect.bottom - 240,
        campo.rect.bottom - 120
    ]
    for y in y_def[:4]:  # Corrigido para 4 defensores
        jogadores.append(Jogador(campo.rect.left + 160, y, num,
(255,255,0)))
        num += 1
    # Meio (3)
    y_meio = [
        campo.rect.top + 200,
        campo.rect.centery,
        campo.rect.bottom - 200
    ]
    for y in y_meio:
        jogadores.append(Jogador(campo.rect.left + 320, y, num,
(255,255,0)))
        num += 1
    # Ataque (3)
    y_ataque = [
        campo.rect.top + 200,
        campo.rect.centery,
        campo.rect.bottom - 200
    ]
    for y in y_ataque:
        jogadores.append(Jogador(campo.rect.left + 480, y, num,
(255,255,0)))
        num += 1

    # Time da direita (direita para esquerda)
    # Goleiro
    jogadores.append(Jogador(campo.rect.right - 60,
campo.rect.centery, num, (0,191,255)))
```

```
    num += 1
    # Defesa (4)
    for y in y_def[:4]:
        jogadores.append(Jogador(campo.rect.right - 160, y, num,
(0,191,255)))
        num += 1
    # Meio (3)
    for y in y_meio:
        jogadores.append(Jogador(campo.rect.right - 320, y, num,
(0,191,255)))
        num += 1
    # Ataque (3)
    for y in y_ataque:
        jogadores.append(Jogador(campo.rect.right - 480, y, num,
(0,191,255)))
        num += 1

    player_positions_history = []

    show_heatmap = False
    individual_heatmap = False
    current_player_id = 1

    while running:
        for event in pygame.event.get():
            if event.type == QUIT:
                running = False
            elif event.type == pygame.MOUSEBUTTONDOWN:
                mx, my = pygame.mouse.get_pos()
                for jogador in jogadores:
                    if (mx - jogador.x)**2 + (my - jogador.y)**2 <=
PLAYER_RADIUS**2:
                        selected_player = jogador
                        break
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_h:
                    show_heatmap = not show_heatmap
                    individual_heatmap = False
                if event.key == pygame.K_n and show_heatmap:
                    individual_heatmap = True
```

```
                    current_player_id += 1
                    if current_player_id > len(jogadores):
                        current_player_id = 1

        keys = pygame.key.get_pressed()
        if keys[K_w]: bola.move(0, -BALL_SPEED)
        if keys[K_s]: bola.move(0, BALL_SPEED)
        if keys[K_a]: bola.move(-BALL_SPEED, 0)
        if keys[K_d]: bola.move(BALL_SPEED, 0)

        # Move o jogador selecionado com as setas
        if selected_player:
            if keys[K_UP]:    selected_player.move(0, -
PLAYER_SPEED, campo)
            if keys[K_DOWN]:  selected_player.move(0, PLAYER_SPEED,
campo)
            if keys[K_LEFT]:  selected_player.move(-PLAYER_SPEED,
0, campo)
            if keys[K_RIGHT]: selected_player.move(PLAYER_SPEED, 0,
campo)

        now = pygame.time.get_ticks()
        if now - last_slice >= SLICE_TIME:
            campo.quadtree.insert((bola.x, bola.y))
            last_slice = now

        status = campo.verificar_gol(bola)
        if status in ["esquerda", "direita"]:
            placar[status] += 1
            bola.x, bola.y = SCREEN_WIDTH//2, SCREEN_HEIGHT//2

        screen.fill((0,100,0))
        campo.draw(screen)
        bola.draw(screen)

        for jogador in jogadores:
                jogador.draw(screen)

        font = pygame.font.SysFont(None, 28)
```

```
        placar_txt = font.render(f" {placar['esquerda']} x
{placar['direita']}  |  {status}", True, (255,255,0))
        screen.blit(placar_txt, (20, 20))

        # Salva as posições dos jogadores neste frame
        frame_positions = {i+1: (jogador.x, jogador.y) for i,
jogador in enumerate(jogadores)}
        player_positions_history.append(frame_positions)

        if show_heatmap:
            heatmap_surface = pygame.Surface((SCREEN_WIDTH,
SCREEN_HEIGHT), pygame.SRCALPHA)
            if individual_heatmap:
                generate_heatmap(heatmap_surface,
player_positions_history, campo.rect,
target_player_id=current_player_id)
            else:
                generate_heatmap(heatmap_surface,
player_positions_history, campo.rect)
            screen.blit(heatmap_surface, (0, 0))
            # Mostra o número do jogador se for individual
            if individual_heatmap:
                font = pygame.font.SysFont(None, 36)
                txt = font.render(f"Jogador {current_player_id}",
True, (255,255,0))
                screen.blit(txt, (SCREEN_WIDTH//2-80, 30))

        pygame.display.flip()
        clock.tick(60)

    pygame.quit()
    gerar_heatmap(campo)

    # Gera e salva o heatmap após o jogo
    heatmap_surface = pygame.Surface((SCREEN_WIDTH, SCREEN_HEIGHT),
pygame.SRCALPHA)
    generate_heatmap(heatmap_surface, player_positions_history,
campo.rect)
    pygame.image.save(heatmap_surface, "heatmap_jogadores.png")
```

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