Análise Completa do Sistema e Proposta de Melhorias

📊 Análise do Sistema Atual

✅ Pontos Fortes

1. **Base de Dados Robusta**: Esquema Prisma bem estruturado com `RaceGoal` para múltiplas corridas
2. **Gerador de Planos com IA**: Sistema inteligente usando GPT-4o para personalização
3. **Disponibilidade Flexível**: Sistema de `trainingActivities` permite configuração detalhada por atividade
4. **TrainingLog**: Sistema de feedback diário do atleta
5. **AI Analysis**: Capacidade de análise contínua

❌ Gaps Críticos Identificados

🚨 GAP #1: Desconexão entre RaceGoal e Plan Generator

Problema: O sistema possui:
- `AthleteProfile` com `goalDistance` e `targetRaceDate` (usado pela IA)
- Tabela `RaceGoal` com múltiplas corridas (COMPLETAMENTE IGNORADA)

Impacto:

```
// ATUAL: IA recebe apenas UMA corrida
const aiProfile: AIUserProfile = {
  goalDistance: profile.goalDistance,  // ❌ Apenas uma corrida
  targetRaceDate: profile.targetRaceDate, // ❌ Apenas uma data
  // ... sem acesso a profile.raceGoals
}

// RESULTADO: Plano ignora outras corridas cadastradas
```

Exemplo Real:
- Usuário cadastra: Maratona em 20 semanas (principal) + 10K em 2 semanas (preparatória)
- Sistema atual: Gera plano APENAS para a maratona, IGNORA a 10K
- Resultado: Atleta corre a 10K sem tapering, sem ajuste no plano

---

🚨 GAP #2: Ausência de Periodização para Múltiplas Corridas

Base Científica (da pesquisa):
- **Non-Linear Periodization**: Ideal para 11+ corridas/ano
- **Block Periodization**: Blocos curtos (2-4 semanas) para tune-ups
- **Tapering Escalonado**:
- Corrida A (principal): 2-3 semanas
- Corrida B (secundária): 1 semana
- Corrida C (treino): 3-5 dias

Problema Atual:

```
// generateWeekWorkouts() não verifica:
// - Se há corrida próxima (próximas 2 semanas)
// - Importância da corrida (A, B, C)
// - Necessidade de tapering
// - Recovery pós-corrida
```

Exemplo Real:

```
Semana 1-2: Construção (40km/semana)
Semana 3: CORRIDA 10K (sem tapering) ❌
Semana 4: Continua 40km (sem recovery) ❌
```

🚨 GAP #3: Sistema Estático sem Re-planejamento

Problema: Usuário adiciona/remove corridas mas plano não se ajusta

Cenário:
1. Gera plano para Meia Maratona em 16 semanas
2. Na semana 4, cadastra 10K na semana 8
3. Plano NÃO SE AJUSTA para incluir tapering da 10K
4. Sistema continua com plano original (obsoleto)

Impacto: Plano perde relevância, atleta desconsidera o sistema

---

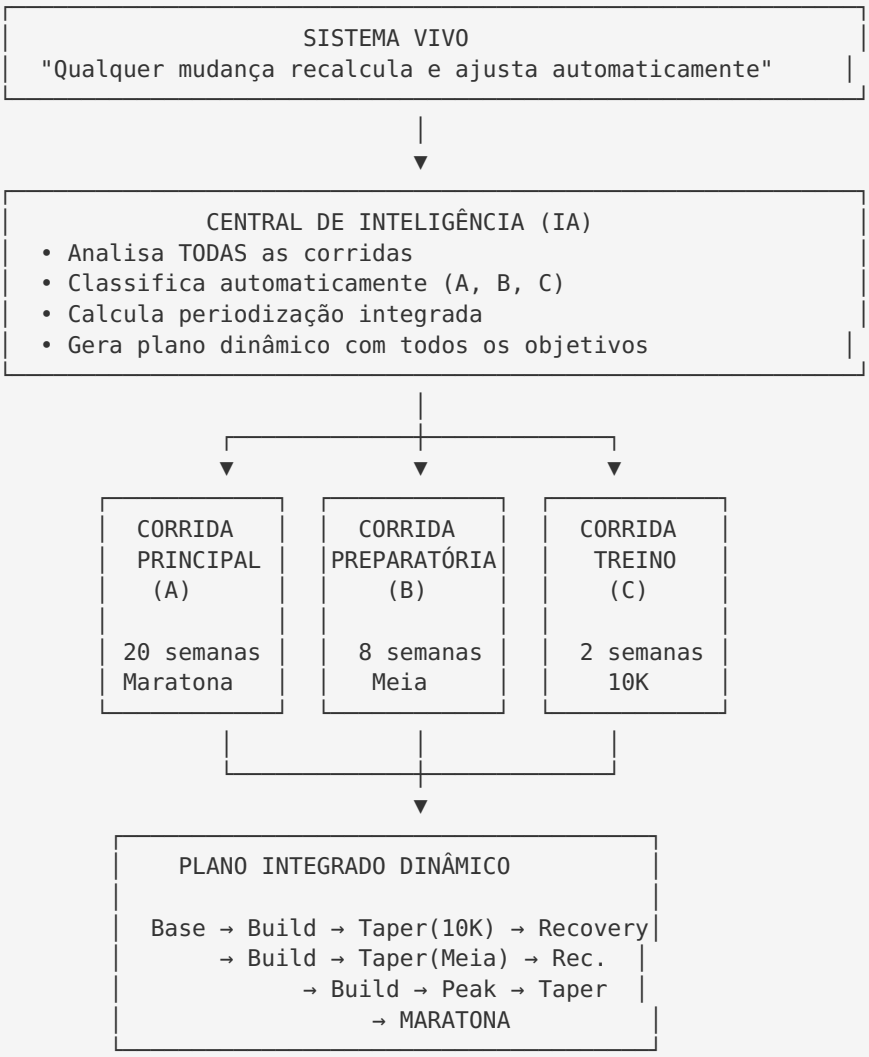
🚨 GAP #4: Falta de Classificação e Priorização

Base Científica:
- Corridas A: 1-3/ano, objetivo principal, tapering completo
- Corridas B: 4-6/ano, preparatórias, tapering moderado
- Corridas C: 7+/ano, treino, tapering mínimo

Problema Atual:
- Todas as corridas têm apenas flag `isPrimary` (binário)
- Não há conceito de “corrida preparatória”
- Não há estratégia de uso de corridas menores

🎯 Proposta de Solução Integrada

Arquitetura do Sistema “Vivo”



🔧 Implementação Técnica

1. Extensão do Schema Prisma

```
model RaceGoal {
  id              Int      @id @default(autoincrement())
  athleteId       Int

  // Detalhes da corrida
  raceName        String
  distance        String
  raceDate        DateTime
  targetTime      String?
  location        String?

  // Status
  status          String   @default("active")

  // ✨ NOVO: Sistema de Classificação Inteligente
  raceType        String   @default("auto") // "A", "B", "C", "auto"
  importance      Int      @default(5) // 1-10 (1=treino, 10=objetivo vida)

  // ✨ NOVO: Estratégia de Tapering
  taperingWeeks   Float?   // Calculado pela IA baseado em importância
  recoveryDays    Int?     // Dias de recovery pós-corrida

  // ✨ NOVO: Relação com o Plano
  integratedInPlan Boolean @default(false) // Se está no plano atual
  planPhase       String?  // Fase do plano onde está inserida

  // ✨ NOVO: Resultado e Análise
  actualTime      String?
  placement       Int?
  notes           String?  @db.Text
  aiAnalysis      String?  @db.Text // Análise pós-corrida pela IA

  createdAt       DateTime @default(now())
  updatedAt       DateTime @updatedAt

  athlete         AthleteProfile @relation(fields: [athleteId], references: [id], onDe
lete: Cascade)

  @@index([athleteId, status])
  @@index([athleteId, raceDate]) // ✨ NOVO: Index para buscar por data
  @@index([athleteId, raceType]) // ✨ NOVO: Index para buscar por tipo
  @@map("race_goals")
}

// ✨ NOVO: Histórico de Mudanças no Plano
model PlanRevision {
  id              Int      @id @default(autoincrement())
  planId          Int
  revisionNumber  Int      // 1, 2, 3...

  reason          String   // "new_race_added", "race_removed", "perform-
ance_adjustment", "injury"
  description     String   @db.Text

  // Snapshot do plano anterior (JSON)
  previousState   Json?

  // Mudanças aplicadas
  changes         Json     // Lista de mudanças específicas

  createdAt       DateTime @default(now())
```

```
  plan              CustomTrainingPlan @relation(fields: [planId], references: [id], o
nDelete: Cascade)

  @@map("plan_revisions")
}
```

---

2. Enhanced AI Plan Generator

Novo Tipo: `MultiRacePlanInput`

```typescript
export interface MultiRacePlanInput {
  // Perfil do atleta (existente)
  athleteProfile: AIUserProfile;

  // ✨ NOVO: Todas as corridas ativas
  races: Array<{
    id: number;
    raceName: string;
    distance: string;
    raceDate: Date;
    targetTime?: string;
    importance: number; // 1-10
    raceType?: 'A' | 'B' | 'C' | 'auto';
  }>;

  // ✨ NOVO: Contexto adicional
  currentDate: Date;
  existingPlanId?: number; // Se está re-planejando
}
```

Novo Fluxo: `generateMultiRaceAIPlan()`

```
/**
 * Gera plano integrado considerando TODAS as corridas do atleta
 *
 * Baseado em:
 * - Non-Linear Periodization para múltiplos objetivos
 * - Block Periodization para peaks específicos
 * - Tapering escalonado por importância
 */
export async function generateMultiRaceAIPlan(
  input: MultiRacePlanInput
): Promise<MultiRaceAIPlan> {

  // ETAPA 1: Classificação Inteligente de Corridas
  const classifiedRaces = await classifyRaces(input.races);

  // ETAPA 2: Calcular Cronograma de Fases
  const phaseSchedule = calculatePhaseSchedule(
    classifiedRaces,
    input.currentDate
  );

  // ETAPA 3: Gerar Estratégia com IA
  const strategy = await generateMultiRaceStrategy({
    profile: input.athleteProfile,
    races: classifiedRaces,
    phaseSchedule,
  });

  // ETAPA 4: Expandir para plano detalhado
  const plan = expandMultiRaceStrategy(strategy);

  // ETAPA 5: Validar periodização
  validateMultiRacePeriodization(plan);

  return plan;
}
```

Sistema de Classificação Automática

```
/**
 * Classifica corridas automaticamente baseado em:
 * - Distância
 * - Tempo até a corrida
 * - Proximidade com outras corridas
 * - Importância definida pelo usuário
 */
function classifyRaces(races: RaceInput[]): ClassifiedRace[] {
  // Ordenar por data
  const sorted = races.sort((a, b) =>
    a.raceDate.getTime() - b.raceDate.getTime()
  );

  const classified: ClassifiedRace[] = [];

  for (const race of sorted) {
    const weeksUntilRace = getWeeksUntil(race.raceDate);

    // REGRA 1: Corrida mais distante e longa = Provável A
    const isLongestDistance = race.distance === 'marathon' ||
                              race.distance === 'half_marathon';

    // REGRA 2: Usuário marcou importância alta
    const highImportance = race.importance >= 8;

    // REGRA 3: Muito próxima de outra corrida = Provável C
    const hasCloseRace = classified.some(r =>
      Math.abs(getWeeksUntil(r.raceDate) - weeksUntilRace) < 3
    );

    let raceType: 'A' | 'B' | 'C';

    if (highImportance && isLongestDistance) {
      raceType = 'A'; // Objetivo principal
    } else if (hasCloseRace || weeksUntilRace < 4) {
      raceType = 'C'; // Treino / Shakeout
    } else {
      raceType = 'B'; // Preparatória / Tune-up
    }

    // Calcular tapering baseado no tipo
    const taperingWeeks = {
      'A': race.distance === 'marathon' ? 3 : 2,
      'B': 1,
      'C': 0.5, // Apenas redução nos últimos 3 dias
    }[raceType];

    // Calcular recovery baseado no tipo e distância
    const recoveryDays = calculateRecoveryDays(race.distance, raceType);

    classified.push({
      ...race,
      raceType,
      taperingWeeks,
      recoveryDays,
      classification: {
        reason: `Classificada como ${raceType} por: ${
          highImportance ? 'alta importância, ' : ''
        }${
          isLongestDistance ? 'distância longa, ' : ''
        }${
          hasCloseRace ? 'proximidade com outra corrida' : ''
```

```
        }`,
      },
    });
  }

  return classified;
}
```

Cálculo de Cronograma de Fases

```
/**
 * Divide o tempo total em fases considerando TODAS as corridas
 *
 * Exemplo:
 * Hoje → 10K (2 sem) → Meia (8 sem) → Maratona (20 sem)
 *
 * Fases:
 * 1. Base: Semana 1-4 (construção geral)
 * 2. Build + Taper 10K: Semana 5-6
 * 3. Recovery 10K: Semana 7
 * 4. Build para Meia: Semana 8-12
 * 5. Taper Meia: Semana 13
 * 6. Recovery Meia: Semana 14
 * 7. Build para Maratona: Semana 15-18
 * 8. Peak: Semana 19
 * 9. Taper: Semana 20-21
 * 10. Race Week: Semana 22
 */
function calculatePhaseSchedule(
  races: ClassifiedRace[],
  startDate: Date
): PhaseSchedule {
  const phases: Phase[] = [];
  let currentWeek = 1;

  for (let i = 0; i < races.length; i++) {
    const race = races[i];
    const isLastRace = i === races.length - 1;
    const nextRace = races[i + 1];

    const weeksUntilRace = getWeeksFromDate(startDate, race.raceDate);
    const weeksBetweenRaces = nextRace
      ? getWeeksFromDate(race.raceDate, nextRace.raceDate)
      : 0;

    // FASE 1: BUILD (até começar taper)
    const buildWeeks = weeksUntilRace - currentWeek - race.taperingWeeks;

    if (buildWeeks > 0) {
      phases.push({
        type: i === 0 ? 'base' : 'build',
        name: i === 0
          ? 'Base Aeróbica'
          : `Construção para ${race.raceName}`,
        startWeek: currentWeek,
        endWeek: currentWeek + buildWeeks - 1,
        weeks: buildWeeks,
        focus: getFocusForDistance(race.distance),
        targetRace: race.raceName,
      });

      currentWeek += buildWeeks;
    }

    // FASE 2: TAPER
    if (race.taperingWeeks > 0) {
      phases.push({
        type: 'taper',
        name: `Polimento para ${race.raceName}`,
        startWeek: currentWeek,
        endWeek: currentWeek + Math.ceil(race.taperingWeeks) - 1,
        weeks: Math.ceil(race.taperingWeeks),
```

```
        focus: 'Redução de volume, manutenção de intensidade',
        targetRace: race.raceName,
        volumeReduction: race.raceType === 'A' ? 0.4 : 0.25, // A: -40%, B/C: -25%
      });

      currentWeek += Math.ceil(race.taperingWeeks);
    }

    // FASE 3: RACE WEEK
    phases.push({
      type: 'race',
      name: `Semana de Prova - ${race.raceName}`,
      startWeek: currentWeek,
      endWeek: currentWeek,
      weeks: 1,
      focus: 'Prova + ativação',
      targetRace: race.raceName,
    });

    currentWeek++;

    // FASE 4: RECOVERY (se não for última corrida)
    if (!isLastRace && race.recoveryDays > 0) {
      const recoveryWeeks = Math.ceil(race.recoveryDays / 7);

      phases.push({
        type: 'recovery',
        name: `Recuperação pós ${race.raceName}`,
        startWeek: currentWeek,
        endWeek: currentWeek + recoveryWeeks - 1,
        weeks: recoveryWeeks,
        focus: 'Recuperação ativa, volume reduzido',
        volumeReduction: 0.3, // -30%
      });

      currentWeek += recoveryWeeks;
    }
  }

  return {
    phases,
    totalWeeks: currentWeek - 1,
  };
}
```

3. Sistema de Re-planejamento Automático

Trigger: Nova corrida adicionada

```
// app/api/race-goals/route.ts - POST
export async function POST(req: NextRequest) {
  // ... código existente ...

  const raceGoal = await prisma.raceGoal.create({ ... });

  // ✨ NOVO: Trigger de re-planejamento
  if (profile.hasCustomPlan && profile.customPlanId) {
    // Verificar se corrida está dentro do período do plano
    const plan = await prisma.customTrainingPlan.findUnique({
      where: { id: profile.customPlanId },
      include: { weeks: true }
    });

    const raceIsInPlan = isDateInRange(
      raceGoal.raceDate,
      plan.startDate,
      plan.targetRaceDate
    );

    if (raceIsInPlan) {
      // REPLANEAR automaticamente
      await replanWithNewRace(profile.id, raceGoal);

      return NextResponse.json({
        raceGoal,
        message: '✨ Plano ajustado automaticamente para incluir esta corrida!',
        planAdjusted: true
      }, { status: 201 });
    }
  }

  return NextResponse.json({ raceGoal }, { status: 201 });
}
```

Função de Re-planejamento Inteligente

```
async function replanWithNewRace(
  athleteId: number,
  newRace: RaceGoal
) {
  console.log(`[REPLAN] Nova corrida adicionada: ${newRace.raceName}`);

  // 1. Buscar perfil e todas as corridas
  const profile = await prisma.athleteProfile.findUnique({
    where: { id: athleteId },
    include: {
      raceGoals: {
        where: { status: 'active' },
        orderBy: { raceDate: 'asc' }
      },
      customPlan: {
        include: { weeks: { include: { workouts: true } } }
      }
    }
  });

  // 2. Criar snapshot do plano atual
  const snapshot = createPlanSnapshot(profile.customPlan);

  // 3. Gerar novo plano integrado
  const newPlan = await generateMultiRaceAIPlan({
    athleteProfile: mapProfileToAI(profile),
    races: profile.raceGoals,
    currentDate: new Date(),
    existingPlanId: profile.customPlanId,
  });

  // 4. Preservar treinos JÁ COMPLETADOS
  const completedWorkouts = await prisma.completedWorkout.findMany({
    where: { athleteId: profile.id }
  });

  // 5. Substituir plano mantendo ID (para não quebrar referências)
  await replaceExistingPlan(
    profile.customPlanId,
    newPlan,
    completedWorkouts
  );

  // 6. Criar revisão do plano
  await prisma.planRevision.create({
    data: {
      planId: profile.customPlanId,
      revisionNumber: (await getLastRevisionNumber(profile.customPlanId)) + 1,
      reason: 'new_race_added',
      description: `Corrida "${newRace.raceName}" (${newRace.distance}) adicionada
para ${newRace.raceDate.toLocaleDateString()}. Plano ajustado para incluir periodiza-
ção adequada.`,
      previousState: snapshot,
      changes: calculateChanges(snapshot, newPlan),
    }
  });

  // 7. Notificar atleta
  await sendPlanAdjustmentNotification(profile.userId, {
    reason: 'new_race',
    raceName: newRace.raceName,
    adjustments: ['Tapering adicionado', 'Recovery programado', 'Volume ajustado'],
```

```
  });

  console.log(`[REPLAN] Plano ajustado com sucesso!`);
}
```

---

4. Prompt Melhorado para IA

```
const systemPrompt = `Você é um treinador expert em periodização para MÚLTIPLAS CORRI-
DAS.

CONTEXTO CIENTÍFICO:
- Non-Linear Periodization: Ideal para atletas com múltiplas corridas no ano
- Block Periodization: Blocos curtos de 2-4 semanas para objectives específicos
- Tapering baseado em importância:
  * Corrida A (objetivo principal): 2-3 semanas, redução 40-60%
  * Corrida B (preparatória/tune-up): 1 semana, redução 25-30%
  * Corrida C (treino): 3-5 dias, redução 15-20%
- Recovery pós-corrida:
  * Maratona: 2-3 semanas easy running
  * Meia Maratona: 1-2 semanas
  * 10K: 3-7 dias
  * 5K: 2-4 dias

REGRAS CRÍTICAS:
1. SEMPRE considerar TODAS as corridas cadastradas
2. NUNCA sobrepor fases de taper de corridas diferentes
3. SEMPRE incluir recovery adequado pós-corrida
4. Ajustar volume baseado em densidade de corridas
5. Usar corridas B como "simulação" para corrida A

ESTRATÉGIAS:
- Se múltiplas corridas em curto prazo: Periodização ondulada
- Se corridas espaçadas (>6 semanas): Blocos independentes
- Se corrida próxima (<3 semanas): Iniciar imediatamente com taper
- Corridas B devem simular intensidade de corrida A sem desgaste excessivo`;

const userPrompt = `${userContext}

# CORRIDAS DO ATLETA

O atleta possui as seguintes corridas cadastradas (ordenadas por data):

${races.map((race, i) => `
## Corrida ${i + 1}: ${race.raceName}
- Distância: ${race.distance}
- Data: ${race.raceDate.toLocaleDateString('pt-BR')}
- Tipo: ${race.raceType} (${getRaceTypeDescription(race.raceType)})
- Importância: ${race.importance}/10
- Meta de Tempo: ${race.targetTime || 'Não definida'}
- Semanas até a prova: ${getWeeksUntil(race.raceDate)}
`).join('\n')}

# CRONOGRAMA DE FASES CALCULADO

${phaseSchedule.phases.map(phase => `
- Semanas ${phase.startWeek}-${phase.endWeek}: ${phase.name}
  * Tipo: ${phase.type}
  * Foco: ${phase.focus}
  ${phase.volumeReduction ? `* Redução de volume: ${phase.volumeReduction * 100}%` : '
'}
`).join('\n')}

# TAREFA

Crie uma ESTRATÉGIA DE TREINAMENTO INTEGRADA que:
1. Prepare o atleta para TODAS as corridas listadas
2. Use corridas B como simulação/preparação para corrida A
3. Implemente tapering apropriado para cada corrida baseado no tipo
4. Inclua recovery adequado após cada corrida
```

```
5. Mantenha volume equilibrado considerando densidade de corridas
6. Progrida de forma científica e sustentável

ATENÇÃO: Este é um plano para MÚLTIPLAS corridas. Cada corrida tem papel específico:
- Corridas C: Treino de ritmo de prova
- Corridas B: Simulação e ajuste de estratégia
- Corrida A: Objetivo principal, peak de performance

Responda com JSON detalhado seguindo o formato especificado.`;
```

📊 Exemplo Prático

Cenário: Usuário com 3 Corridas

Corridas Cadastradas:
1. **10K Local** - 2 semanas (Importância: 4/10)
2. **Meia Maratona XYZ** - 10 semanas (Importância: 7/10)
3. **Maratona de São Paulo** - 20 semanas (Importância: 10/10)

Classificação Automática:
1. 10K → Tipo C (treino, muito próxima)
2. Meia → Tipo B (preparatória, importante mas não principal)
3. Maratona → Tipo A (objetivo principal)

Cronograma Gerado:

```
Semana 1: Base Aeróbica
Semana 2: Mini-taper 10K (3 dias)
Semana 3: CORRIDA 10K + Recovery (3 dias)
Semana 4-9: Build para Meia Maratona
Semana 10: Taper Meia (7 dias)
Semana 11: CORRIDA MEIA + Recovery
Semana 12-13: Recovery completo
Semana 14-17: Build para Maratona (Base)
Semana 18-19: Peak (Intensidade)
Semana 20-21: Taper Maratona
Semana 22: MARATONA
```

Plano Gerado pela IA:
- ✅ 10K usada como “shakeout race” para testar ritmo
- ✅ Meia usada como simulação de ritmo de maratona + 30s/km
- ✅ Volume progressivo: 35km → 50km → 45km (taper 10K) → 60km → 50km (taper Meia) → 70km → 80km (peak) → 50km (taper)
- ✅ Recovery adequado entre corridas
- ✅ Cada corrida tem objetivo específico no contexto maior

🚀 Roadmap de Implementação

Fase 1: Fundação (1-2 dias)

- [ ] Atualizar schema Prisma com novos campos

- [ ] Criar `MultiRacePlanInput` e tipos relacionados
- [ ] Implementar `classifyRaces()`
- [ ] Implementar `calculatePhaseSchedule()`

Fase 2: Geração Integrada (2-3 dias)

- [ ] Criar `generateMultiRaceAIPlan()`
- [ ] Atualizar prompts da IA
- [ ] Implementar validação de periodização
- [ ] Testar com cenários múltiplos

Fase 3: Re-planejamento (1-2 dias)

- [ ] Implementar `replanWithNewRace()`
- [ ] Criar sistema de snapshots
- [ ] Implementar `PlanRevision` model
- [ ] Adicionar triggers em race-goals API

Fase 4: Interface (1-2 dias)

- [ ] Melhorar UI de gestão de corridas
- [ ] Mostrar classificação automática (A/B/C)
- [ ] Dashboard de periodização visual
- [ ] Notificações de ajustes no plano

Fase 5: Inteligência Avançada (2-3 dias)

- [ ] Análise pós-corrida automática
- [ ] Ajuste de VDOT baseado em performances
- [ ] Sugestões de corridas preparatórias
- [ ] Otimização de calendário de corridas

📖 Referências Científicas

1. **Bosquet et al. (2007)** - “Effects of Tapering on Performance” - Meta-análise sobre tapering
2. **Mujika & Padilla (2003)** - “Scientific Bases for Precompetition Tapering”
3. **Daniels’ Running Formula** - Sistema VDOT e periodização
4. **Pfitzinger & Douglas** - “Advanced Marathoning” - Multiple race preparation
5. **Verkhoshansky & Issurin** - Block Periodization
6. **Seiler (2010)** - “What is best practice for training intensity?” - Polarized training

---

💡 Benefícios do Sistema Integrado

Para o Atleta

✅ Planos que fazem sentido com TODAS suas corridas
✅ Preparação adequada para cada objetivo
✅ Confidence em seguir o plano (sabe que o sistema está “vendo” tudo)

✅ Flexibilidade para adicionar corridas sem perder o plano
✅ Aprendizado: entende o papel de cada corrida

Para o Negócio

✅ Diferencial competitivo único
✅ Sistema “vivo” aumenta engagement
✅ Redução de churn (planos sempre relevantes)
✅ Base científica sólida
✅ Escalabilidade (IA faz o trabalho pesado)

Para a Ciência

✅ Aplicação prática de periodização não-linear
✅ Coleta de dados sobre efetividade
✅ Validação de estratégias de múltiplas corridas
✅ Contribuição para o conhecimento do treinamento

🎯 Próximos Passos

1. **Revisar este documento** com o time
2. **Validar abordagem** científica e técnica
3. **Priorizar fases** de implementação
4. **Implementar Fase 1** como MVP
5. **Testar com usuários** reais
6. **Iterar baseado** em feedback
7. **Expandir funcionalidades** avançadas

Autor: Sistema DeepAgent
Data: 25 de Outubro de 2025
Versão: 1.0
Status: Proposta para Revisão