# Universidade Federal Fluminense - UFF

# Instituto de Computação

# Projeto de Software - Ticket to Ride

***Grupo 3 - Jansen Alves, Paulo Carrano, Rafael Langsch e Rafael Valverde***

## 1) Descrição do escopo do sistema

### 1.1 Visão geral

Sistema inspirado no jogo de tabuleiro Ticket to Ride, no qual 2 a 5 jogadores competem para ver quem consegue construir mais rotas ferroviárias, ligando cidades em um mapa e completando bilhetes de destino para pontuar.

### 1.2 Objetivos do produto

- Suportar partidas locais para 2 a 5 jogadores.
- Fluxo completo de jogo: setup, turnos, gatilho de final e pontuação final.
- Mapa fixo (ex: América do Norte) com cidades e rotas ligando-as.
- Baralhos: cartas de trem com diferentes cores e bilhetes de destino.
- Ações por turno: comprar cartas, construir uma rota, pegar/descartar bilhetes.
- Regras de pontuação: por tamanho da rota, por número de bilhetes cumpridos/não cumpridos e por maior trilha contínua.
- Persistência: salvar/retomar partida.
- Interface 2D simples com feedback de estado.

### 1.3 Stakeholders e usuários

- Jogadores casuais e entusiastas de board games.
- Equipe de desenvolvimento (produto, devs, QA, UI/UX).
- Docentes (avaliação acadêmica).

### 1.4 Requisitos funcionais

1. Criar sala/partida local e configurar número de jogadores.
2. Embaralhar e distribuir cartas iniciais (trens e bilhetes) conforme regras.
3. Gerenciar turnos e validar ações possíveis.
4. Exibir mão do jogador atual, cartas abertas e rotas disponíveis.
5. Permitir compra de rota com verificação de cor/locomotivas/quantidade.
6. Manter baralhos, descarte e reposição de cartas abertas (3 locomotivas).
7. Detectar gatilho de fim de jogo e encerrar após a última rodada.
8. Calcular pontuações (rotas, bilhetes, penalidades, maior trilha).
9. Salvar/carregar o estado da partida.

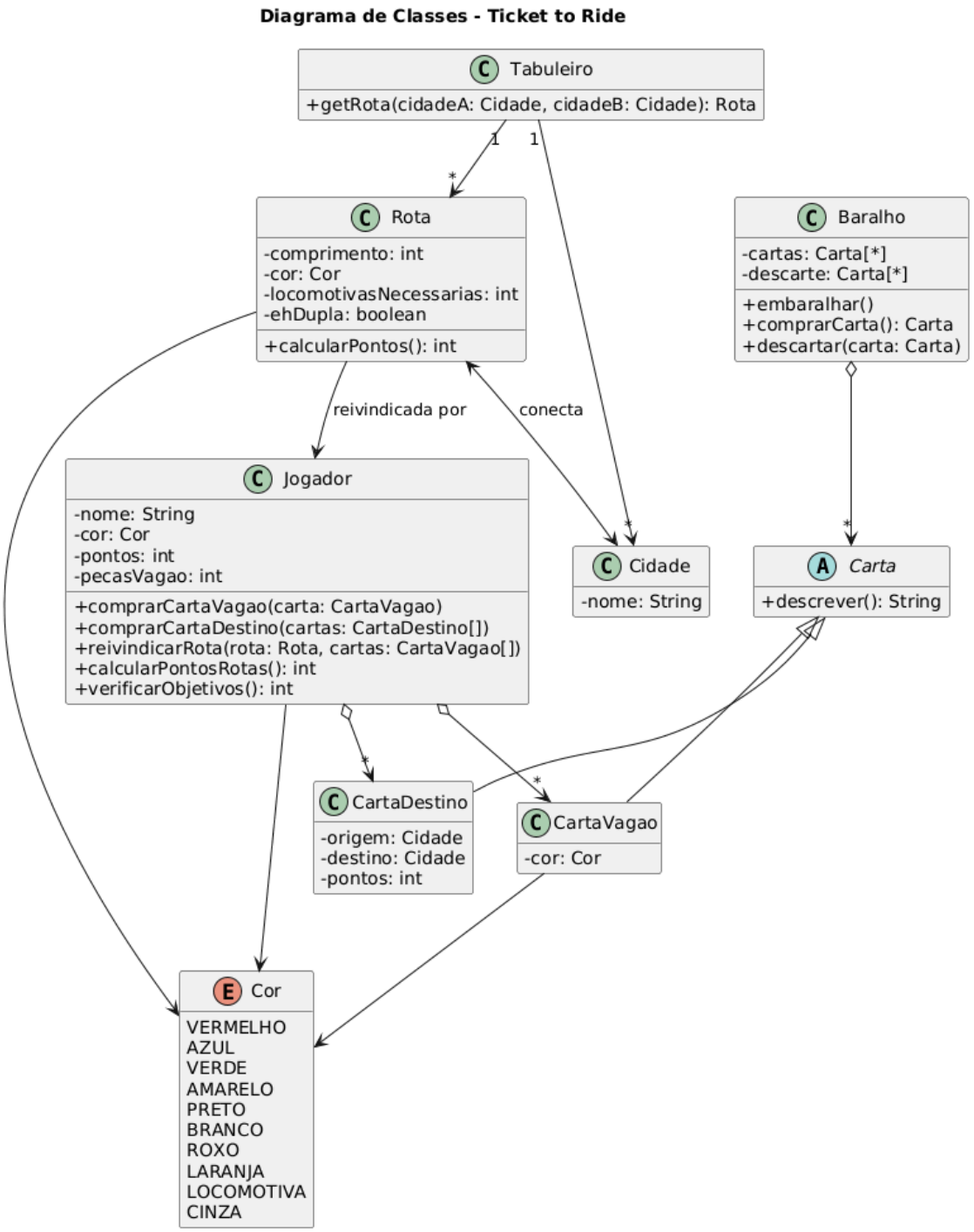
### 1.5 Requisitos não funcionais

- Desempenho: tempo de resposta menor que 100 ms.
- Portabilidade: desktop (Windows/macOS/Linux).
- Manutenção: arquitetura em camadas, com regras desacopladas da UI.
- Confiabilidade: garantir que todas as ações realizadas estejam permitidas nas regras.
- Escalabilidade: suportar de 2 a 5 jogadores por partida.

### 1.6 Restrições e premissas

- Uso de bibliotecas gráficas padrão (ex.: JavaFX/React/Unity).
- Regras baseadas na edição clássica do jogo.

## 2) Diagrama de classes




```
@startuml
skinparam classAttributeIconSize 0
hide empty members
title Diagrama de Classes - Ticket to Ride

enum Cor {
  VERMELHO
  AZUL
  VERDE
  AMARELO
  PRETO
  BRANCO
```

```
  ROXO
  LARANJA
  LOCOMOTIVA
  CINZA
}

class Tabuleiro {
  + getRota(cidadeA: Cidade, cidadeB: Cidade): Rota
}

class Cidade {
  - nome: String
}

class Rota {
  - comprimento: int
  - cor: Cor
  - locomotivasNecessarias: int
  - ehDupla: boolean
  + calcularPontos(): int
}

class Jogador {
  - nome: String
  - cor: Cor
  - pontos: int
  - pecasVagao: int
  + comprarCartaVagao(carta: CartaVagao)
  + comprarCartaDestino(cartas: CartaDestino[])
  + reivindicarRota(rota: Rota, cartas: CartaVagao[])
  + calcularPontosRotas(): int
  + verificarObjetivos(): int
}

abstract class Carta {
  + descrever(): String
}

class CartaVagao extends Carta {
  - cor: Cor
}

class CartaDestino extends Carta {
  - origem: Cidade
  - destino: Cidade
  - pontos: int
}

class Baralho {
  - cartas: Carta[*]
  - descarte: Carta[*]
  + embaralhar()
  + comprarCarta(): Carta
  + descartar(carta: Carta)
}

' Relacionamentos
Tabuleiro "1" --> "*" Cidade
Tabuleiro "1" --> "*" Rota
Rota <--> Cidade : conecta
```

```
Rota --> Jogador : reivindicada por
Jogador o--> "*" CartaVagao
Jogador o--> "*" CartaDestino
Baralho o--> "*" Carta
Jogador --> Cor
Rota --> Cor
CartaVagao --> Cor
@enduml
```