

A blurred background image of a RoboCup Small Size League match. The scene shows a green field with white lines, several small robots, and a goal. Two robots in the foreground are labeled 'TINO 11' and 'OSPINA 2'.

# Small Size League

A top-down view of a RoboCup Small Size League soccer field. The field is green with white lines. Several small, cylindrical robots are visible. Two robots in the foreground are labeled 'TINO 11' and 'OSPINA 2'. Other robots are scattered across the field, some in motion as indicated by motion blur. A goal is visible in the bottom right corner.

# Small Size League

- Uma das ligas mais antigas da RoboCup Soccer
- Focada em coordenação multi-agent em um ambiente altamente dinâmico
- Existem 3 categorias atualmente (A - 11vs11, B - 6x6 e Entry Level - 3x3 )
- Disputada nacional e internacionalmente
- Desenhada para impulsionar o estado da arte na robótica inteligente
- Livro de regras bem definido

A background image showing a green soccer field with white lines. Several small, cylindrical robots are scattered across the field. Two robots in the foreground are labeled 'TINO 11' and 'OSPINA 2'. The robots are dark-colored with colorful circular markers on top. The field is slightly out of focus, emphasizing the robots.

# Small Size League

Quando foi fundada (em 1997) a missão original era atingir um nível de tecnologia tal que um time de SSL pudesse ganhar do time campeão da **Copa do Mundo** até 2050.

Nosso objetivo é competir no **MUNDIAL** de SSL

# Software Oficial

A categoria fornece um ecossistema de soluções integradas  
que somos obrigados a utilizar para competir



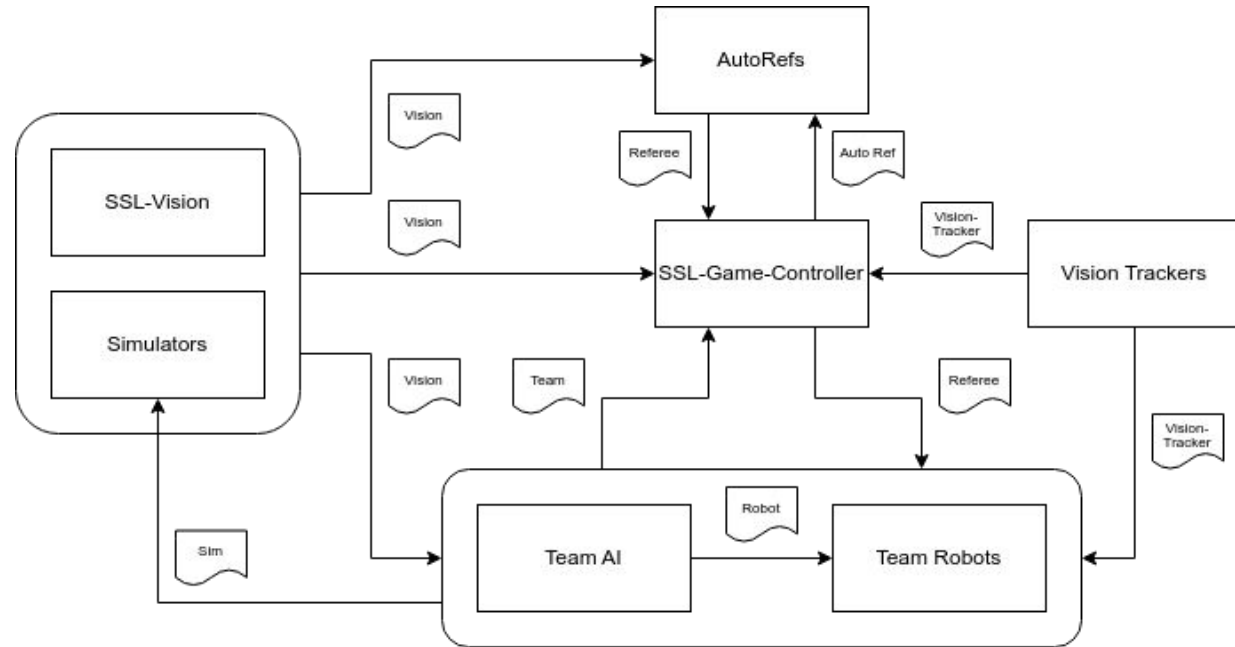
# Software Oficial

## 1. SSL Components

1. SSL-Vision
2. SSL-Game-Controller
3. AutoRefs
4. Simulators
5. Vision Trackers
6. Team AI
7. Team Robots

## 2. Additional League Infrastructure

1. SSL-Log-Tools
2. SSL-Status-Board
3. SSL-Vision-Client
4. SSL-Simulation-Setup
5. SSL-Setup
6. SSL-Match-Stats



## 3. Standard Network Parameters

## 4. Protobuf Definitions

# Software Oficial

SSL-Game-Controller

SSL-Game-Controller

Auto-Referees: 2 | Tracker Sources: 2 ☐ Dark

STARTMATCHPROTOCOLCOMMANDSPLACE BALLGAME EVENTSTEAMSETTINGS

Team Settings

Yellow

Test Team

☐ Timeout request

☐ Substitution request

☐ Emergency request

0 ⇒ 11

Active cards ⇒ max bots

0:00

Next yellow card due

Blue

Test Team

☐ Timeout request

☐ Substitution request

☐ Emergency request


0 ⇒ 11

Active cards ⇒ max bots

0:00

Next yellow card due

☐ Continue automatically (Ctrl+Space)

Press Ctrl +  to continue

Press Esc to Halt

1

BLUE START BALL PLACEMENT

2

HALT

Bot crashed into opponent bot

Blue Bot crashed into opponent bot

Feb, 04 2023 20:54:53,358

Yellow Ball left field via touch line

State: STOP

Stage: Second Half (1:36 left)

Score: Yellow 1 : 0 Blue

Matching duration: 5:10:26

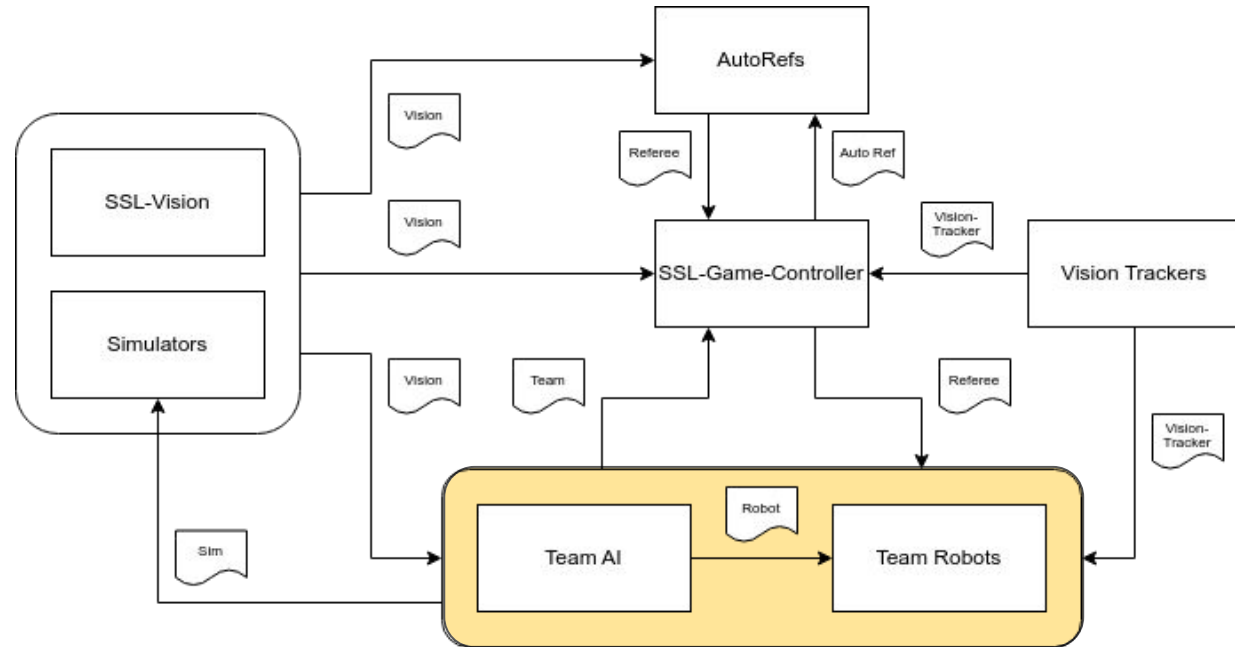
# Software Oficial

## 1. SSL Components

1. SSL-Vision
2. SSL-Game-Controller
3. AutoRefs
4. Simulators
5. Vision Trackers
6. Team AI
7. Team Robots

## 2. Additional League Infrastructure

1. SSL-Log-Tools
2. SSL-Status-Board
3. SSL-Vision-Client
4. SSL-Simulation-Setup
5. SSL-Setup
6. SSL-Match-Stats



# SSL Software



# Sumário

1. Software
2. Simulação
3. Visualização
4. Planejamento de Trajetórias
5. Comportamentos
6. Tomada de Decisão

# Software

Responsável pela comunicação entre todos os módulos.

Utilizando o **ROS 2**, a comunicação entre os módulos é gerenciada pelos **nós**, proporcionando maior **modularidade** ao sistema. Isso torna o sistema **mais rápido e eficiente**, porém, como contrapartida, aumenta a **complexidade** da sua implementação.

# Robot Operating System 2 (ROS2)

## O que é o ROS 2 Humble?

O **ROS 2 Humble Hawksbill** é uma das versões **LTS (Long-Term Support)** do ROS 2, lançada em **maio de 2022**. Como uma versão LTS, ela recebe suporte e correções de bugs por um período mais longo (geralmente **5 anos**), sendo ideal para aplicações de produção e projetos de longo prazo.

# Robot Operating System 2 (ROS2)

## Principais características do ROS 2 Humble:

**Baseado no Ubuntu 22.04 LTS** – Maior compatibilidade e estabilidade.

**Melhorias de desempenho e tempo real** – Ideal para robôs industriais e autônomos.

**Aprimoramento no gerenciamento de pacotes e comunicação** – Maior eficiência na troca de dados entre componentes do robô.

**Suporte a middleware DDS** – Melhor integração e segurança na comunicação distribuída.

**Compatibilidade com versões anteriores** – Facilita a migração de projetos do ROS 2 Foxy.

# Funcionamento de Nós e Tópicos no ROS

## Nós (Nodes)

Os **nós** são processos independentes que executam funções específicas.

- Exemplo: Um nó pode **ler sensores**, outro pode **controlar motores**.

## Tópicos (Topics)

Os **tópicos** são canais de comunicação baseados no modelo **publicador/assinante (publish/subscribe)**.

- **Publicadores** enviam mensagens para um tópico.
- **Assinantes** recebem mensagens desse tópico.

# Funcionamento de Serviços e Parâmetros no ROS 2

## Serviços (Services)

Os **serviços** permitem comunicação **solicitação/resposta** (**request/response**) entre nós.

- Diferente dos tópicos, onde a comunicação é contínua, um serviço só responde quando solicitado.

## Parâmetros (Parameters)

Os **parâmetros** armazenam valores que configuram o comportamento dos nós.

- São usados para definir configurações sem precisar alterar o código.

# Funcionamento de Ações no ROS 2

## Ações (Actions)

As **ações** permitem executar tarefas que **demoram um tempo** para serem concluídas.

- Diferente dos serviços, as ações permitem **feedback contínuo** durante a execução.
- Exemplo: Um nó "braço robótico" pode executar a ação **"mover\_para\_posição"**, enviando atualizações até finalizar o movimento.



<b>Mecanismo</b>	<b>Modelo</b>	<b>Uso</b>
<b>Tópicos</b>	Publicador/Assinante	Enviar dados contínuos
<b>Serviços</b>	Solicitação/Resposta	Pedidos rápidos com retorno imediato
<b>Parâmetros</b>	Configuração	Ajustar valores sem alterar o código
<b>Ações</b>	Solicitação + Feedback	Tarefas demoradas com progresso contínuo

# Simulação

A simulação é um ambiente virtual que recebe os mesmos comandos que os robôs físicos recebem.

Utilizamos o **grSim**, mesmo software utilizado pela Robocup-SSL.

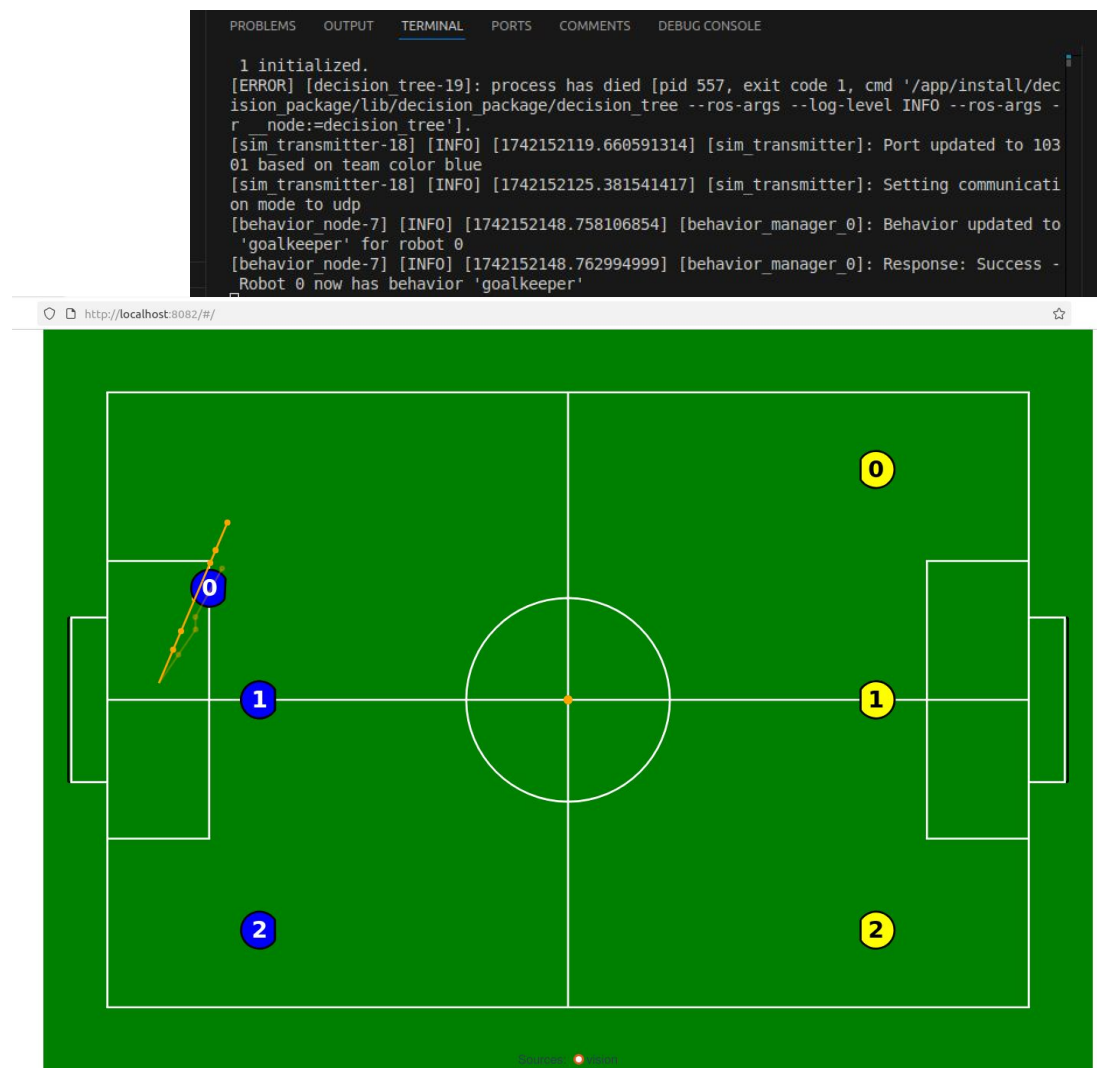
Os comandos são gerados são similares aos gerados pelo SSL-Vision e são transmitidos usando o Google Protobuf.



# Visualização

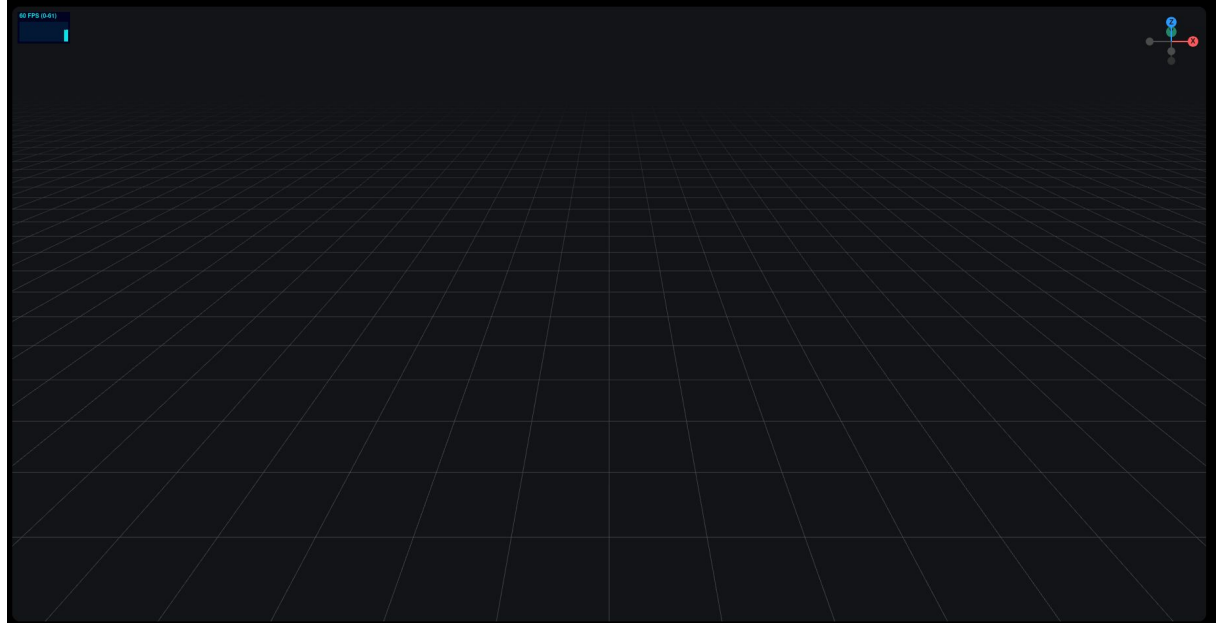
Até então, o meio que utilizamos para a visualização foi o visualizador da própria organização do torneio.

Entretanto, descobrimos que isso foi uma brecha que o Castro encontrou no sistema deles, e pretendemos ter nosso próprio visualizador para este ano.



# Visualização

Este aqui é o visualizador da Robocin, quando não recebe nada da visão.



# Visualização

Este aqui é o visualizador da Robocin, quando recebe as informações da visão usando o software da Robocin.

Nosso objetivo é conseguir usar nosso software para enviar estas informações e utilizar o visualizador.



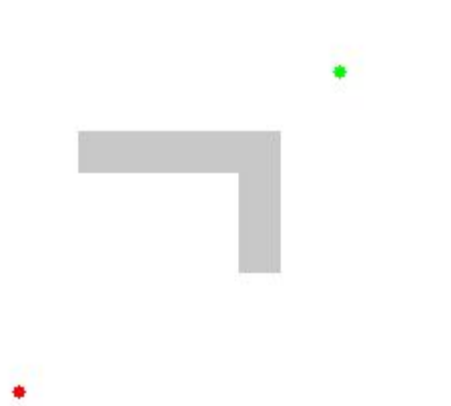
# Planejamento de Trajetórias

Script responsável por encontrar a trajetória entre dois pontos, evitando obstáculos.

- A\*
- RRT / RRT\*
- BangBang (Otimização de Controle)

# A\*

- Utilizado para encontrar o caminho ótimo entre dois pontos.
- Mais eficiente do que busca em profundidade ou largura, considerando o uso de uma boa heurística.
- Custoso em complexidade de tempo de execução





# Rapid-Exploring Random Tree (RRT)

Utiliza uma abordagem baseada em amostras para encontrar um caminho entre dois pontos de forma rápida.

Difícilmente vai encontrar um caminho ótimo, mas é muito eficiente.

# Rapid-Exploring Random Tree (RRT)

## Algorithm BuildRRT

Input: Initial configuration  $q_{init}$ , number of vertices in RRT  $K$ , incremental distance  $\Delta q$

Output: RRT graph  $G$

$G.init(q_{init})$

**for**  $k = 1$  **to**  $K$  **do**

$q_{rand} = \text{RAND\_CONF}()$

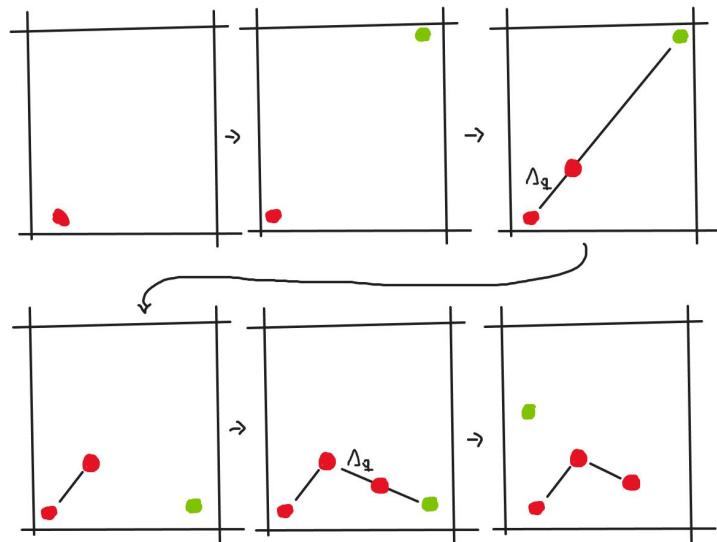
$q_{near} = \text{NEAREST\_VERTEX}(q_{rand}, G)$

$q_{new} = \text{NEW\_CONF}(q_{near}, q_{rand}, \Delta q)$

$G.add\_vertex(q_{new})$

$G.add\_edge(q_{near}, q_{new})$

**return**  $G$



# Rapid-Exploring Random Tree (RRT)

**Algorithm** BuildRRT

Input: Initial configuration  $q_{init}$ , number of vertices in RRT  $K$ ,  
incremental distance  $\Delta q$

Output: RRT graph  $G$

$G.\text{init}(q_{init})$

**for**  $k = 1$  **to**  $K$  **do**

$q_{rand} = \text{RAND\_CONF}()$

$q_{near} = \text{NEAREST\_VERTEX}(q_{rand}, G)$

$q_{new} = \text{NEW\_CONF}(q_{near}, q_{rand}, \Delta q)$

$G.\text{add\_vertex}(q_{new})$

$G.\text{add\_edge}(q_{near}, q_{new})$

**return**  $G$

# RRT\*

Igual ao RRT, porém a cada vértice adicionado, verifica se há um caminho ótimo até a origem

- Otimiza os caminhos criados anteriormente ao longo da execução
- Possui uma complexidade de tempo maior que o RRT devido às revisões dos caminhos já existentes

# Comportamentos

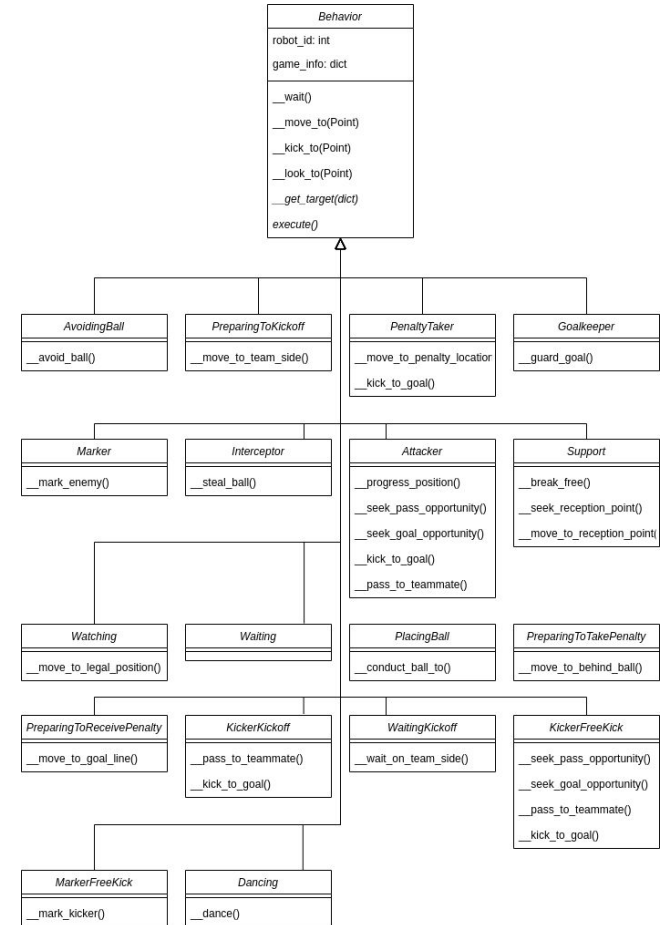
Comportamentos genéricos como atacante, goleiro, marcador, etc.

A idéia era evitar troca de comportamentos dentro de um mesmo estado para tornar a implementação mais simples.

Esse ano queremos, além dos comportamentos genéricos, comportamentos mais específicos que sejam trocados dentro de um mesmo estado. A idéia é que isso possibilite estratégias mais complexas e pontuais como formação de barreira.

# Rule-Based System (RBS)

Todos os comportamentos descendem de uma classe com as funções básicas de um agente. Essas funções básicas são utilizadas para implementar os comportamentos mais complexos.



# Alguns Métodos de Treinamento de Comportamento

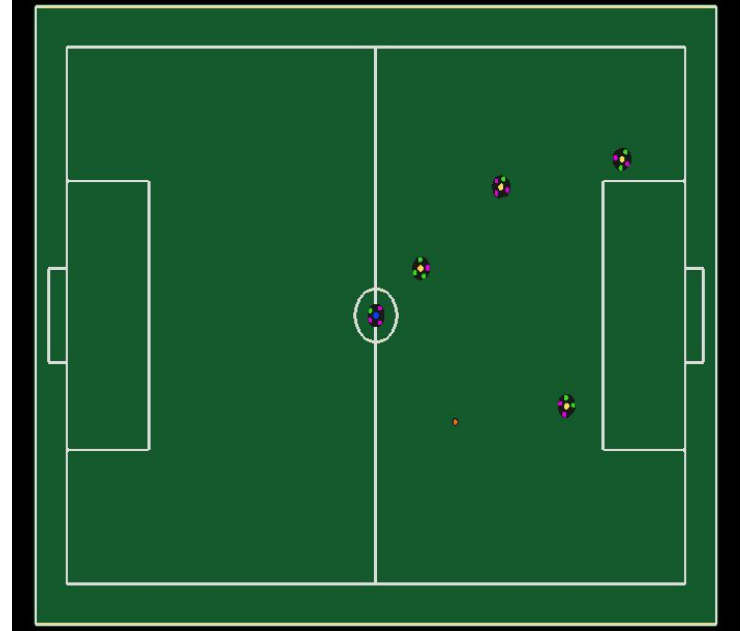
- Proximal Policy Optimization (PPO)
- Soft Actor-Critic (SAC)
- Graph Neural Networks (GNNs)



# Ambiente de Treinamento

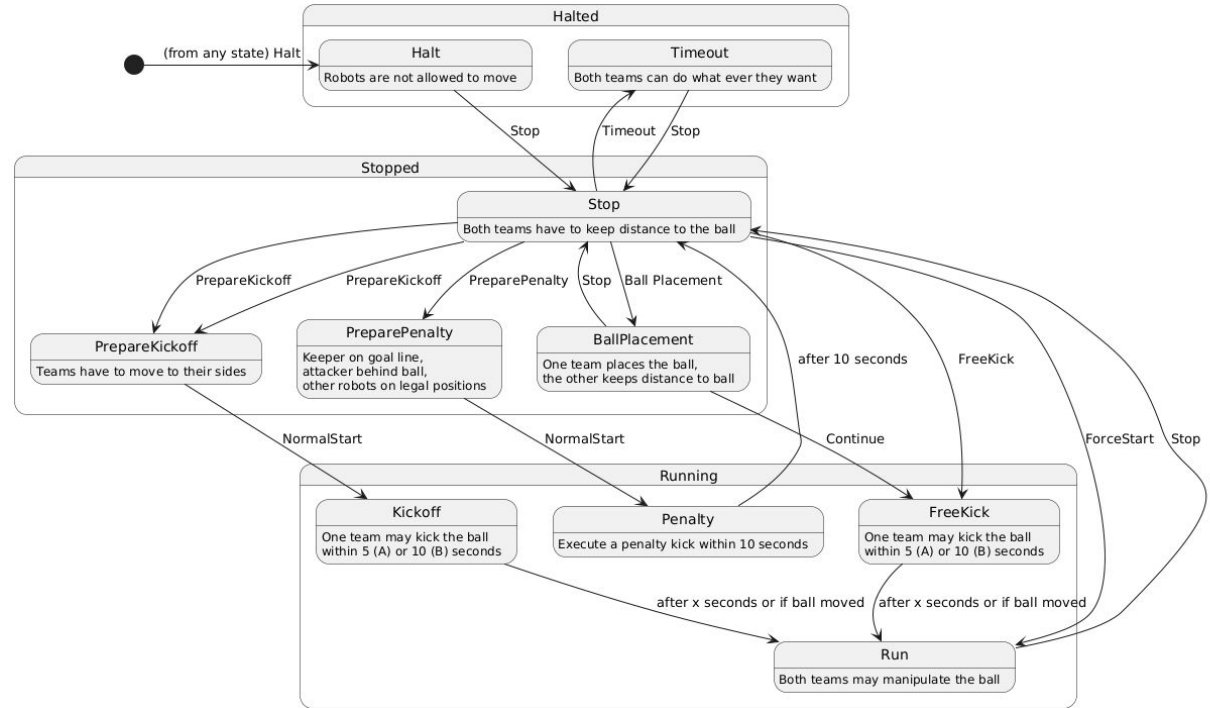
O rSoccer é uma implementação da Robocin em cima da biblioteca gymnasium. Ela adiciona alguns ambientes de treino do SSL.

Esse ano queremos criar mais ambientes de treino para treinar algumas skills básicas e quem sabe até skills mais complexas.



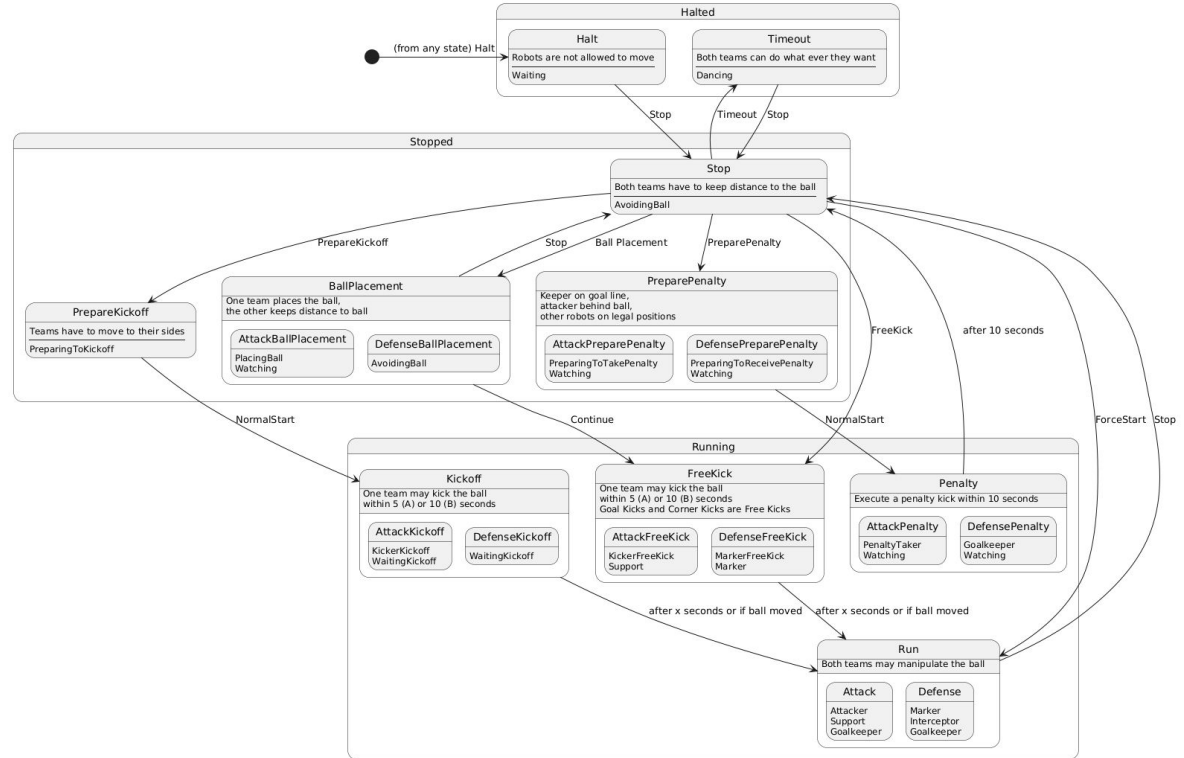
# Tomada de Decisão

Estados fornecidos  
pelo próprio evento



# Decision Tree

Comportamentos utilizados em cada estado



# Outros Métodos de Tomada de Decisão

- Multi-Agent Reinforcement Learning (MARL)
- Hierarchical RL (HRL)

# Case-Based Reasoning (CBR)

Método baseado em armazenar vários casos diferentes. Dada uma situação no presente, o modelo recupera um caso parecido que já tenha vivenciado e aplica a mesma solução adaptada ao presente.

4 R's do CBR:

- Retrieve
- Reuse
- Revise
- Retain

# Mixture of Experts

- Treina vários modelos especialistas em coisas diferentes
- Treina uma *Gating Network* que escolhe quais modelos serão ativados para resolver o problema