Project Neon 2020 Very Small Size Soccer

Alexsandro Francisco dos Santos[1], Cesar Seiji Maruyama[2], Gabriel Mendes de Lima[3],
Ivan Seidel Gomes[4], Joao Pedro V. B. Silva[5], Marcus Vinícius H. de Lima[6],
Jessica Bibiano Lopes da Silva[7], Aline Peixoto de Menezes [8], Amanda Mendes Sampaio[9],

***Resumo*— Esse Artigo irá demonstrar o desenvolvimento de um novo sistema de tomada de decisão para o torneio IEEE Very Small Size Soccer da LARC 2020. O Torneio deste ano consiste numa disputa online em um ambiente virtual simulado através do Software FIRASim. Esse projeto tem como objetivo reconstruir o que já foi feito em um projeto anterior (NoPlan) de forma mais modular afim de aproveitar os algoritmos que já tínhamos desenvolvido, além de novos algoritmos adaptados ao novo ambiente de competição.**

I. INTRODUÇÃO

A Equipe foi fundada em 2015 por alunos da da Universidade Federal do ABC e é atualmente composta por 20 membros de diferentes áreas do conhecimento.

Um dos pilares mais importantes adotados pelo time é o conceito de *open-source*: todo trabalho desenvolvido é aberto, seja hardware ou software, para uso de qualquer um, permitindo uma rede de colaboração e aprendizado aberta.

Nas condições inesperadas do ano de 2020, e considerando que a equipe historicamente tem um foco mais relacionado a atividades de hardware e eletrônica, a decisão de embarcar no desafio de reescrever o software usado desde a LARC 2018 foi difícil de ser tomada inicialmente, mas com o novo objetivo em vista, começamos esse projeto com preocupação em desenvolver um software mais modular, criado desde o inicio com uma documentação didática, para que seja não apenas um software que possa ser competitivo dentro das modalidades, mas que também seja uma plataforma de ensino para que membros, veteranos e novatos, possam se debruçar sobre tópicos como engenharia de software, algoritmos de controle e inteligência artificial.

II. SOFTWARE

Desde as iterações anteriores já havia o desejo da equipe de mudar o software de maneira que ele pudesse comportar uma série de funcionalidades. Como:

- Ser mais modular em relação ao recebimento e envio de dados, de forma que possamos competir em outras modalidades semelhantes.
- Desenvolver o software com uma documentação ampla desde seu inicio, facilitando no treinamento de novos membros da equipe, e servindo de guia para novas contribuições.
- Permitir que o software possa tanto trabalhar com robôs que façam seu PID internamente no Firmware (como no modelo 2018), quanto robôs que necessitando de um algoritmo de controle externo.
- Facilitar a implementação de algoritmos de tomada de decisão diferentes para comparar suas performances.

Com esses itens em mente, chegamos a conclusão de redesenhar o software como descrito nas próximas subseções.

A. Módulos

O NeonFC foi desenvolvido para ser um software altamente desacoplável. Para atingir esse objetivo decidimos separar o software em módulos que possam ser trocados dependendo das condições da partida. Esses módulos são:

- Match: modulo principal, que encapsula a iteração do jogo e orquestra os modulos *Entities*.
- Vision: módulo responsável pelo recebimento dos dados de visão.
- Comm: módulo responsável pelo envio dos comandos para os robôs.
- Entities: módulo responsável por representar as diferentes entidades dentro de campo, sejam elas entidades reais (bola, robôs e robôs adversários) ou virtuais (técnico).

Dessa forma, a lógica da partida se mantém intacta no módulo principal *Match* enquanto as formas de envio e recebimento do ambiente do jogo podem alternar entre uma partida física usando visão computacional, ou uma partida virtual, usando simuladores. O módulo das entidades também é desacoplável pensando na possibilidade de robôs com capacidades diferentes (sistemas de controles diferentes, dimensões velocidades distintas, etc.).

O Conceito de um software modularizado também nos ajudou a validar o funcionamento de diferentes algoritmos usados tanto para tomada de decisão do robô, quanto para o sistema de controle, nos possibilitando uma testagem rápida de diferentes algoritmos. Com ajuda de conceitos de programação orientada a objetos pudemos testar diferentes sistemas de controles atrelados a diferentes sistemas de tomada de decisão, que serão descritos na subseção *Fluxo das Entidades*.

[1] alexsandro2134@gmail.com
[2] cesarseiji95@gmail.com
[3] mendes.gabriel95@gmail.com
[4] ivanseidel@gmail.com
[5] joaopedrovbs@gmail.com
[6] marcus.vinicius.99@hotmail.com
[7] je.jessika@hotmail.com
[8] alinezoaxerechan@gmail.com
[9] amandamendesk2@gmail.com

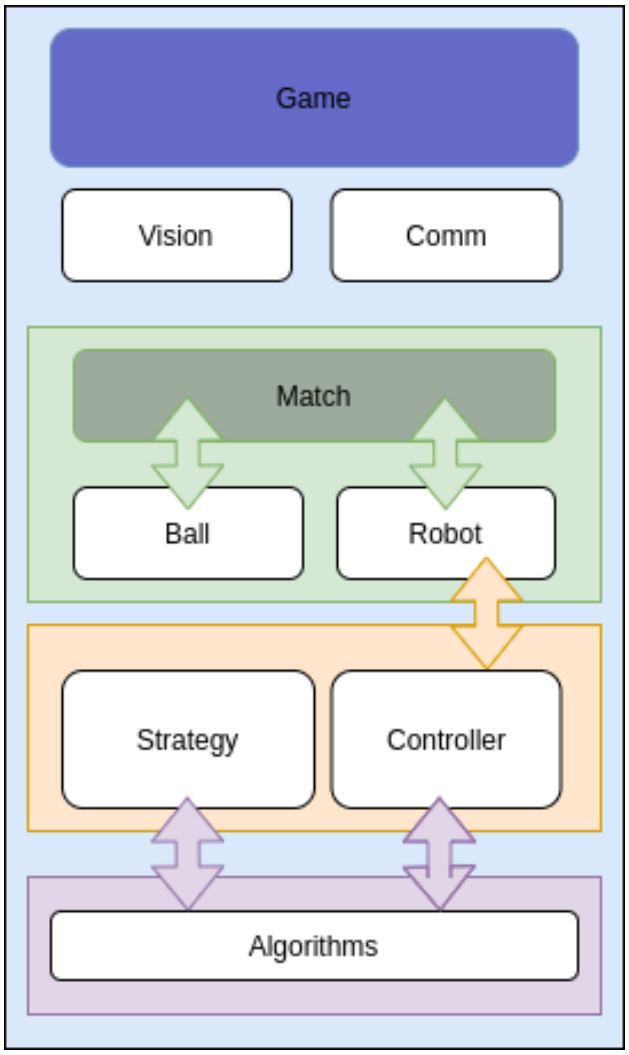

Fig. 1: Todos os Módulos contidos no NeonFC, bem como suas relações (setas bidirecionais representam que o módulo superior é implementado com os módulos inferiores)

Por fim, a facilidade de desacoplar os módulos, também permitiu que membros da equipe com menos familiaridade a esses algoritmos pudessem testar configurações novas e desmembrar as diferentes etapas do fluxo de tomada de decisão do NeonFC. Isso se demonstrou uma decisão acertada dado que diversos algoritmos validados dentro do contexto do nosso software pudessem ser facilmente portados para novos projetos e robôs.

B. Fluxo da Partida

A *Match* é nosso módulo principal, que encapsula toda a lógica da partida, bem como suas *entidades*. Esse módulo tem duas funções principais: a) fazer a ponte entre o módulo *Game* e as entidades. b) ser o responsável pela iteração principal como orquestrador das entidades em jogo.

O módulo Game é o primeiro objeto a ser inicializado na execução do NeonFC e tem como responsabilidade tratar e armazenar informações que precedem a execução do jogo, mais precisamente quais módulos de visão e comunicação

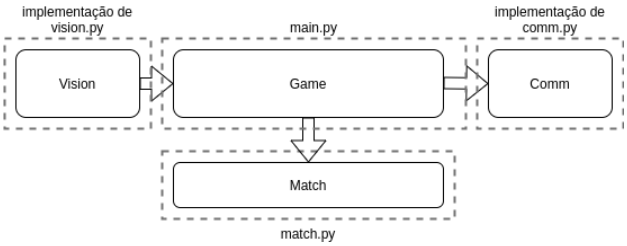

Fig. 2: Iteração principal do NeonFC com seus módulos de visão e comunicação

instanciar, quais são seus arquivos de configuração, quantos robôs inicialmente estarão em campo e qual a cor inicial do time (amarelo ou azul). Por fim, essas informações são passadas para os modulo *Vision*, *Comm* e *Match* para que sejam devidamente inicializados.

O módulo *Vision* nada mais é do que uma implementação da classe threading, thread do Python3, e que por padrão irá rodar com um loop infinito a fim de receber informações do ambiente. Para cada recebimento de informações do campo, uma iteração da Match será executada. O próprio *Vision* também trata potenciais ruídos que possam vir da fonte bem como cálculos de frames por segundo para controle do ambiente.

O módulo *Comm* apenas encapsula as funções necessárias para o envio do dado para o robô. Sendo que a estrutura da mensagem a ser enviada fica a cargo da entidade robot.

C. Fluxo das Entidades

Durante a inicialização do módulo *Match*, através do arquivo de configuração, são instanciados diferentes objetos que encapsularão entidades importantes da partida, essas são chamadas de *Entities*. Cada *Entity* herda de uma classe abstrata que exige a implementação de três métodos importantes: start, update e execute.

1) start: Método de inicialização da entidade, inicializa qualquer estrutura necessária para o objeto que essa entidade representa funcione, por exemplo a inicialização de algoritmos de decisão.

2) update: Método usado para receber quais informações externas a entidade para atualizar suas variáveis. Principal dados da posição dessa entidade no campo mas não unicamente isso, podendo receber outras informações como dados de um arbitro automatizado ou feedback do firmware do robô como status das baterias.

O método update também é responsável por calcular informações derivadas aquelas que recebe, como qualquer velocidade dessa entidade através da sua posição ou status de estar em disputa de bola através da informação dos robôs adversários.

3) decide: Método responsável por executar qualquer tomada de decisão que aquela entidade possa fazer, no caso das entidades robot é usado para saber o próximo comando a ser enviado para o ambiente, no caso do coach para saber quais módulos de estrategia serão atribuídos a quais robôs.

Um decisão de design de software que tomamos para as entidades é que mesmo elementos totalmente alheios a

comando do software (como a bola ou robôs adversários) também são tratados como entidades. Essas entidades não tem seus métodos de decide executados mas atualizam seus dados através do update. Como consequência, essa decisão torna fácil no futuro usarmos do mesmo software para tomar decisão pelas duas equipes, facilitando testes mais amplos onde queremos analisar o comportamento contra uma equipe de estrategia semelhante.

D. Coach, Strategy e Algorithms

Para que as entidades tomem decisões, seguimos como inspiração algumas ideias do algoritmos de tomada de decisão da equipe CMDragons da Canargie Mellon University[4]. Utilizando uma entidade chamada coach. O Coach é a entidade virtual que tem como objetivo centralizar a tomada de decisão do comportamento das demais entidades. O Coach faz isso tento uma lista de estrategias e uma série de regras. Para cada estrategia existe uma regra que deve ser atendida e uma prioridade, a entidade mais apta a receber aquela estrategia passa a desempenha-la.

A entidade Estrategia (Strategy) é uma entidade que encapsula um ou mais algoritmos de tomada de decisão de maneira a abstrair como é calculado e se focar naquilo que efetivamente aquele algoritmo faz. Um exemplo seria a estrategia FollowBall, que implementa um algoritmo de campo potencial.

Dessa maneira podemos dividir não apenas nosso código, mas os membros da equipe que estão desenvolvendo o projeto em criadores de estrategia e implementadores dos algoritmos.

III. ESTRATÉGIA

Nosso objetivo desenvolvendo o NeonFC era principalmente ampliar as possibilidades de algoritmos que poderíamos adicionar as estrategias já disponíveis no antigo software NoPlan [1]. Dessa forma além de optarmos pelo desenvolvimento mais orientado a objeto visando a desacoplação também decidimos desenvolver na linguagem Python, essa decisão foi tomada pensando no Python como uma linguagem muito usada dentro do meio cientifico e acadêmico além da sua facilidade de ser aprendido. Graças a sua adesão dentro da comunidade acadêmica muitos algoritmos já foram implementados e bem testados nessa linguagem e bibliotecas poderosas como o scipy e o numpy nos ajudam com a parte matemática.

A. Coach

Nessa categoria de competição, com um campo pequeno sendo usado nas partidas, os times são bem reduzidos, o que torna necessário que todos os jogadores sejam bem versáteis. Nessa situação, onde a troca de funções é algo constante, como por exemplo quando o atacante vira defensor para melhorar a capacidade defensiva do time, transições físicas são ineficientes em relação a tempo, então fazer os robôs simplesmente assumirem novas funções a nível de software é mais adequado.

A função do *Coach* é organizar as funções dos jogadores no campo durante a partida, para que estes possam trabalhar juntos efetivamente no cumprimento do objetivo, ganhar. O *Coach* decide o número de goleiros, atacantes e defensores durante a partida. O número de robôs atribuídos para cada função reflete o nível de agressividade desejado para a situação atual do jogo. A divisão das N funções é baseada em variáveis que descrevem as circunstâncias atuais de jogo, como possessão de bola, posição no campo e a agressividade do time adversário.

O *Coach* vai ser responsável por decidir as melhores estratégias de movimentos para cada jogador, assim como também, as excepcionais. Por exemplo, se a bola passa entre um jogador e o gol, com um espaço aberto a frente, o *Coach* vai atribuir a esse jogador uma função de ataque. Nesse caso, a função de ataque vai dar ”fortes incentivos”ao robô, para que este vai na direção de um certo ponto (ou curva), e ao mesmo tempo, tentar repelir o robô para impedir que este bata nos outros enquanto tenta concluir a ação. [4].

AGRADECIMENTOS

Nós agradecemos aos fundadores da nossa equipe, pelo esforço que eles colocaram nesse projeto, e na organização da equipe em geral, criando um ambiente de aprendizado amigável e convidativo para todos. Não podemos deixar de agradecer também alguns dos nosso parceiros chave, Ivan Seidel, Rodrigo Hausen um amigo próximo da equipe, e nossos patrocinadores, Robocore, Tenda Digital e Engenhoteca, que sempre estiveram com a equipe nos ajudando a colocar nossos projetos no mundo, providenciando as melhores peças e serviços para nossos robôs. Sem eles, nada disso seria possível.

REFERÊNCIAS

[1] NoPlan code available in: https://github.com/Project-Neon/NoPlan
[2] NeonFC code available in: https://github.com/Project-Neon/NeonFC
[3] CMDragons paper. Coordinate players layer, 2015. Available in: http ://goo.gl/mGYlOs.
[4] CMDragons paper. Dynamic Passing and Strategy. Available in: http: //goo.gl/1yYYko.