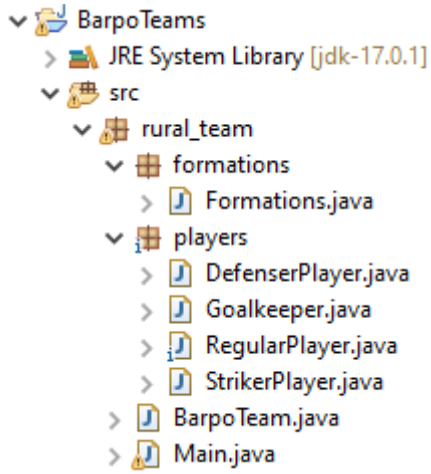
Aluno: Vinicius Barbosa Polito Lopes
Projeto: Github

1. Estrutura e Introdução

O projeto está estruturado da seguinte forma:



A classe **Main** é onde os times são instanciados e a partida começa.

```java
package rural_team;

import java.net.UnknownHostException;

public class Main {
    public static void main(String[] args) throws UnknownHostException {
        BarpoTeam team1 = new BarpoTeam("Barpo_T1", 7, true);
        BarpoTeam team2 = new BarpoTeam("Barpo_T2", 7, true);

        team1.launchTeamAndServer();
        team2.launchTeam();
    }
}
```

A classe **BarpoTeam** é onde a formação inicial do time é escolhida, e os jogadores são instanciados.

```java
package rural_team;

import easy_soccer_lib.AbstractTeam;

public class BarpoTeam extends AbstractTeam {

    public BarpoTeam(String suffix, int numberOfPlayers, boolean goalkeeper ) {
        super(suffix, numberOfPlayers, goalkeeper);
    }

    @Override
    protected void launchPlayer(int uniform, PlayerCommander commander) {

        RegularPlayer pl;
        Formations formation = new Formations();
        Vector2D form;
        form = formation.initalHexaFormation(commander);

        // se o time estiver no lado direito, faz o ajuste
        if (commander.getMyFieldSide() == EFieldSide.RIGHT) {
            form.setX(-form.getX());
            form.setY(-form.getY());
        }

        System.out.printf("Jogador %d, lado %s -- x %f, y %f    \n", uniform, commander.getMyFieldSide(), form.getX(), form.getY());

        if (uniform >= 2 && uniform <= 4) {
            pl = new DefenserPlayer(commander, form.getX(), form.getY());
        } else if(uniform >= 5) {
            pl = new StrikerPlayer(commander, form.getX(), form.getY());
        }
        else {
            pl = new Goalkeeper(commander, form.getX(), form.getY());
        }
        pl.start();
    }
}
```

No pacote **formations**, está a classe Formations. Esta classe contém todas as 4 diferentes formações.

Posição Inicial

public Vector2D initalHexaFormation(PlayerCommander commander);

Posição Neutra

public Vector2D neutralFormation(PlayerCommander commander);

Posição Ataque

public Vector2D attackingFormation(PlayerCommander commander);

Posição Defesa

public Vector2D defendingFormation(PlayerCommander commander);

A formação Inicial é utilizada antes do começar.
A posição Neutra é no caso do jogo estar empatado.
A posição de ataque é no caso do time estar perdendo.
A posição defensiva é no caso do time estar ganhando.

No pacote **players**, estão as classes **RegularPlayer**, **StrikerPlayer**, **DefenserPlayer** e **GoalkeeperPlayer**.

Toda a implementação está na classe **RegularPlayer**.
As outras classes, **StrikerPlayer**, **DefenserPlayer** e **GoalkeeperPlayer**, herdam as características de **RegularPlayer**, mas diferem entre si em alguns comportamentos.

2. Requisitos do projeto (25/03)

*Vocês devem fazer um time baseado em **máquinas de estados** ou **behavior trees** (ponto extra) assim:*

- *6 jogadores "de linha" e 1 goleiro (7 no total)*
- *capaz de jogar nos dois lados do campo*
- *cobra pelo menos o "kick off" corretamente (com 1 jogador tocando para o outro)*
- *que seja capaz de jogar com um **bom comportamento coletivo***

Para avaliar a qualidade do comportamento do time, eu quero que vocês implementem intencionalmente diferentes jogadas e movimentos e expliquem em um relatório. Exemplos de jogadas e ações que são consideradas "boas":

- *se o time inicia com formação organizada (posicionamento coletivo "organizado")*
- *se o time mantém algum formação durante a partida*
- *se algum jogador corre com a bola*
- *se o jogador faz passes (toca a bola) de forma intencional para um companheiro*
- *se o jogador sem bola faz alguma movimentação inteligente (de marcação, ou de posicionamento ofensivo, etc)*
- *se o time muda de formação durante a partida dependendo da posse de bola (quando está ou não com a bola)*
- *se o time muda de formação durante a partida dependendo do resultado*
- *se o time faz a cobrança das bolas paradas (kick in, free kick, etc)*
- *se o goleiro se movimenta sem a bola*
- *se o goleiro consegue defender/rebater (dica: ao invés de "catch", rebata com o "kick")*
- *se você tem mais de um tipo de jogador de linha (ex.: defensores e atacantes)*
- *outros comportamentos que você tenha implementado e que tenha visto no material (aulas e capítulos do livro)*

*Não precisa implementar todas as ideias listadas, mas precisa garantir um bom resultado coletivo. Um critério adicional para saber se o seu time está jogando bem é se ele vence o **Time de Bruno**. Você vai ter que fazer um **relatório** (em Word ou pdf), **listando as jogadas/ações** e explicando quais as linhas do código ou estados da máquina de estados que garantem esses comportamentos. Vou usar esse relatório para analisar o projeto.*

Os trechos em verde são os requisitos do projeto que foram satisfeitos.

3. Comprovação dos Requisitos

- *6 jogadores "de linha" e 1 goleiro (7 no total)*

Número de jogadores é declarado na Main e a diferenciação entre eles é declarada na classe BarpoTeam.

- *capaz de jogar nos dois lados do campo*

Todos os métodos e aspectos do código foram adaptados para os dois lados, é complicado dizer um ponto específico, logo que todo o código foi adaptado para esse aspecto.

- *cobra pelo menos o "kick off" corretamente (com 1 jogador tocando para o outro)*

Esse aspecto é tratado na classe RegularPlayer.

```
142         case KICK_OFF_LEFT:
143             _printf_once("KICK_OFF_LEFT");
144             kickOffLeft(7, 6);
145             break;
146         case KICK_OFF_RIGHT:
147             _printf_once("KICK_OFF_RIGHT");
148             kickOffRight(6, 7);
149             break;
```

- *se o time inicia com formação organizada (posicionamento coletivo "organizado")*

A posição inicial é declarada na classe BarpoTeam.

```
23          Formations formation = new Formations();
24          Vector2D form;
25          form = formation.initalHexaFormation(commander);
```

- *se o time mantém algum formação durante a partida*

Mantém, está declarado na classe RegularPlayer

```
77              if (matchInfo.getTeamScore(EFieldSide.LEFT) > matchInfo.getTeamScore(EFieldSide.RIGHT)) {
78                  if (selfInfo.getFieldSide() == EFieldSide.LEFT) {
79                      homebase = formation.defendingFormation(commander);
80                  } else {
81                      homebase = formation.attackingFormation(commander);
82                  }
83
84              } else if (matchInfo.getTeamScore(EFieldSide.LEFT) < matchInfo.getTeamScore(EFieldSide.RIGHT)) {
85                  if (selfInfo.getFieldSide() == EFieldSide.LEFT) {
86                      homebase = formation.attackingFormation(commander);
87                  } else {
88                      homebase = formation.defendingFormation(commander);
89                  }
90
91              } else {
92                  homebase = formation.neutralFormation(commander);
93              }
94
```

- *se o jogador faz passes (toca a bola) de forma intencional para um companheiro*

Está declarado em RegularPlayer. Ele encontra o jogador mais próximo e faz um passe.

```
437     public void autoPassing() {
438         if (closestToTheBall()) {
439             // Retorna percepção do jogador que receberá o passe, o jogador mais proximo
440             // dele
441             PlayerPerception p = fieldInfo.getTeamPlayer(selfInfo.getFieldSide(),
442                     closestTeamPlayer(selfInfo.getUniformNumber()));
443             // Chuta para posição do jogador que receberá o passe
444             commander.doKickToPoint(70.0d, p.getPosition());
445         }
446     }
```

- *se o time muda de formação durante a partida dependendo do resultado*

Está declarado em RegularPlayer.

```
77              if (matchInfo.getTeamScore(EFieldSide.LEFT) > matchInfo.getTeamScore(EFieldSide.RIGHT)) {
78                  if (selfInfo.getFieldSide() == EFieldSide.LEFT) {
79                      homebase = formation.defendingFormation(commander);
80                  } else {
81                      homebase = formation.attackingFormation(commander);
82                  }
83
84              } else if (matchInfo.getTeamScore(EFieldSide.LEFT) < matchInfo.getTeamScore(EFieldSide.RIGHT)) {
85                  if (selfInfo.getFieldSide() == EFieldSide.LEFT) {
86                      homebase = formation.attackingFormation(commander);
87                  } else {
88                      homebase = formation.defendingFormation(commander);
89                  }
90
91              } else {
92                  homebase = formation.neutralFormation(commander);
93              }
94
```

- *se o time faz a cobrança das bolas paradas (kick in, free kick, etc)*

Está declarado em RegularPlayer. Lá são tratadas a maioria dos casos de bola parada.

```
131    protected void matchStates(MatchPerception matchInfo) {
132
133        switch (matchInfo.getState()) {
134        case BEFORE_KICK_OFF:
135            _printf_once("BEFORE_KICK_OFF");
136            // Não faz nada
137            break;
138        case TIME_OVER:
139            _printf_once("TIME_OVER");
140            this.stateReturnToHomeBase();
141            break;
142        case KICK_OFF_LEFT:
143            _printf_once("KICK_OFF_LEFT");
144            kickOffLeft(7, 6);
145            break;
146        case KICK_OFF_RIGHT:
147            _printf_once("KICK_OFF_RIGHT");
148            kickOffRight(6, 7);
149            break;
150        case KICK_IN_LEFT:
151            _printf_once("KICK_IN_LEFT");
152            kickInLeft();
153            break;
154        case KICK_IN_RIGHT:
155            _printf_once("KICK_IN_RIGHT");
156            kickInRight();
157            break;
158        case FREE_KICK_LEFT:
159            _printf_once("FREE_KICK_LEFT");
160            freeKickLeft();
161            break;
162        case FREE_KICK_RIGHT:
163            _printf_once("FREE_KICK_RIGHT");
164            freeKickRight();
165            break;
166        case CORNER_KICK_LEFT:
167            _printf_once("CORNER_KICK_LEFT");
```

- *se o goleiro consegue defender/rebater (dica: ao invés de "catch", rebata com o "kick")*

É declarado na classe Goalkeeper.

```
    protected void stateDefendingGoal() {
        if (!closestToTheBall()) {
            state = PState.RETURN_TO_HOME;
            return;
        }

        Vector2D ballPosition = fieldInfo.getBall().getPosition();

        if (arrivedAt(ballPosition)) {
            if (selfInfo.getFieldSide() == EFieldSide.LEFT) {
                _printf_once("GK: Catching the ball...");
                //commander.doCatch(MAX_PRIORITY);
                commander.doKickToPoint(100.0d, new Vector2D(52.0d, 0));//Rebater
            } else {
                _printf_once("GK: Catching the ball (right side)...");
                //commander.doCatch(MAX_PRIORITY);
                commander.doKickToPoint(100.0d, new Vector2D(52.0d, 0));//Rebater
            }

        } else {
            if (isAlignedTo(ballPosition)) {
                _printf_once("GK: Running to the ball...");
                commander.doDashBlocking(100.0d);
            } else {
                _printf("GK: Turning...");
                commander.doTurnToPointBlocking(ballPosition);
            }
        }

    }
```

- *se você tem mais de um tipo de jogador de linha (ex.: defensores e atacantes)*

São declarados em BarpoTeam.

```
        if (uniform >= 2 && uniform <= 4) {
            pl = new DefenserPlayer(commander, form.getX(), form.getY());
        } else if(uniform >= 5) {
            pl = new StrikerPlayer(commander, form.getX(), form.getY());
        }
        else {
            pl = new Goalkeeper(commander, form.getX(), form.getY());
        }
        pl.start();
    }
```

*Um critério adicional para saber se o seu time está jogando bem é se ele vence o **Time de Bruno**.*