FastIC

FastIC ابزاری است که با استفاده از آن می توان با در نظر گرفتن یک توپ فرضی و تعدادی بازیکن اطلاعاتی راجع به زمان رسیدن بازیکن ها به توپ گرفت. از FastIC می شود در جاهای مختلفی مانند پاس و شوت استفاده کرد.

نحوه استفاده از FastIC:

1- برای استفاده از FastIC ابتدا باید یک متغیر از نوع FastIC را بسازید. توجه کنید که از Constructor ای استفاده کنید که یک WorldModel می گیرد.

```
FastIC(const WorldModel * world, bool quarter = false);
```

همانطور که میبینید آرگومان دوم این تابع اختیاری است. این آرگومان مشخص می کند آیا بازیکن های که Quarter هستند هم باید محاسبه بشوند یا نه. در صورتی که مقدار این آرگومان true باشد بازیکن های Quarter هم محاسبه می شوند و در محاسبات حریف در نظر گرفته می شوند.

2- پس از آن شما باید توپی را که می خواهید محاسبات با آن انجام شود با تابع setBall به FastIC بدهید.

```
void setBall(Point ballPos, Point ballVel, int ballDelay = 0, float ballDecay = -9999);
void setBall(Ball _ball, unsigned delay);
```

شما می توانید برای اینکار از هریک از این توابع استفاده کنید.

در تابع اول به جای ballPos باید مکان توپ و به جای ballVel باید سرعت توپ را بدهید.

ballDelay مشخص می کند توپ باید از چه زمانی وارد محاسبات شود. به عنوان مثال اگر شما ballDelay را ۱ بدهید، FastIC در نظر می گیرد که در سایکل اول محاسبات توپ حرکت نمی کند و از سایکل دوم توپ شروع به حرکت می کند.

BallDecay نشان دهنده ی decay توپ است. برای مثال اگر به این آرگومان مقدار 0.5 بدهید FastIC در محاسبات سرعت توپ را هر سایکل نصف می کند. در صورتی که به این آرگومان مقداری ندهید decay توپ مقدار پیش فرض(0.94) در نظر گرفته می شود.

3- در مرحله بعد شما باید بازیکن هایی را که می خواهید در محاسبات حساب شوند را با تابع addPlayer به FastIC بدهید.

```
void addPlayer(const Player * _playerPtr, int _delay = 0, float _controlBuff = 1.0, float _dashMulti = 1.0, int extraDashCycle = 0, bool _plusVel = false);
```

این تابع به عنوان ورودی اول بازیکنی را می گیرد که شما می خواهید در محاسبات FastIC حساب شود. ورودی دوم(delay) مشخص می کند این بازیکن از چه وقتی شروع به حرکت می کند. مثلا شما می توانید با استفاده از این ورودی در محاسبات مشخص کنید که بازیکنان حریف یک سایکل دیرتر شروع به حرکت می کنند.

با استفاده از ورودی سوم می توانید Kickable Area بازیکن مورد نظر خود را کوچک تر در نظر بگیرید. مثلا با دادن مقدار ۲ به این ورودی فرض می شود که Kickable Area بازیکن دو برابر اندازه واقعی است.

ورودی چهارم نیز روی مسافتی که در محاسبات برای بازیکن در نظر گرفته می شود تاثیر می گذارد. مثلا اگر این ورودی ۲ باشد FastIC فرض می کند این بازیکن در هر سایکل دو برابر مقدار اصلی اش می تواند جابجا شود.

بقیه ورودی ها باید در حالت پیش فرض بمانند.

4- انجام کامل محاسبات FastIC وقت گیر است. برای همین توابعی در این کلاس وجود دارد که با استفاده از آن‌ها می‌توانید محسبات FastIC را محدود کنید.

```
void setMaxCount(unsigned max);
```

با استفاده از تابع بالا می‌توانید تعداد بازیکن هایی را که FastIC محاسبات را برای آن‌ها انجام می دهد تعیین کنید. مثلاً اگر به عنوان ورودی به این تابع 1 بدهید، FastIC پس از اینکه اولین بازیکنی که توپ را می‌گیرد پیدا کرد محاسبات را تمام می کند.

```
void setMaxTeammateCount(unsigned max);
```

تابع بالا می‌تواند تعداد بازیکن های هم تیمی را که FastIC محاسبات را برای آن‌ها انجام می دهد تعیین

کنید. مثلاً اگر به عنوان ورودی به این تابع 1 بدهید، FastIC پس از اینکه اولین بازیکنی هم تیمی ای که توپ را می‌گیرد پیدا کرد محاسبات را تمام می کند.

```
void setMaxOpponentCount(unsigned max);
```

این تابع همان کار تابع قبلی را برای حریف ها انجام می دهد. برای مثال در هنگام بررسی سالم بودن یک شوت، دانستن اطلاعات اولین حریفی که توپ را می گیرد برای ما کافی است و لازم نیست بعد از پیدا کردن یک بازیکن محاسبات را ادامه دهیم.

```
void setMaxCycleAfterFirstFastestPlayer(unsigned max);
```

شما می‌توانید با استفاده از تابع بالا مشخص کنید که FastIC چند سایکل بعد از پیدا کردن بازیکن Fastest از ادامه محاسبات صرف نظر کند.

```
void setMaxCycles(unsigned max);
```

با این تابع می‌توانید سایکل های محاسبات را محدود کنید. مثلاً می‌توانید فقط بازیکن هایی را که در کمتر از 50 سایکل به توپ می‌رسند را پیدا کنید.

5- برای انجام محاسبات شما باید توابع refresh و calculate را به ترتیب صدا کنید.

6- بعد از انجام محاسبات شما می‌توانید با استفاده از توابع زیر نتایج آن را دریافت کنید.

```
bool isSelfFastestPlayer() const;
```

این تابع مشخص می‌کند که بازیکن زودتر از همه به توپ می‌رسد یا نه.

```
bool isSelfFastestTeammate() const;
```

این تابع مشخص می‌کند که بازیکن زودتر از بقیه هم تیمی ها به توپ می‌رسد یا نه.

```
bool isOurTeamBallPossessor() const;
```

این تابع مشخص می‌کند که تیم ما زودتر از تیم حریف به توپ می‌رسد یا نه.

```
int getFastestTeammateReachCycle(bool withMe = true) const;
```

این تابع مقدار زمانی که طول می‌کشد تا سریع‌ترین بازیکن هم تیمی به توپ برسد را بر می‌گرداند.
در ورودی این تابع می‌توانید مشخص کنید که خود بازیکن را هم حساب کند یا نه. اگر به این ورودی مقدار ندهید به صورت پیش‌فرض خود بازیکن هم در هم تیمی ها حساب می شود.

```
Point getFastestTeammateReachPoint(bool withMe = true) const;
```

این تابع مکانی که سریع‌ترین بازیکن هم تیمی به توپ می رسد را بر می‌گرداند.
در ورودی این تابع می‌توانید مشخص کنید که خود بازیکن را هم حساب کند یا نه. اگر به این ورودی مقدار ندهید به صورت پیش‌فرض خود بازیکن هم در هم تیمی ها حساب می شود.

```
int getFastestOpponentReachCycle() const;
```

این تابع مقدار زمانی که طول می‌کشد تا سریع‌ترین بازیکن حریف به توپ برسد را بر می‌گرداند.

```
Point getFastestOpponentReachPoint() const;
```

این تابع مکانی که سریع‌ترین بازیکن حریف به توپ می رسد را بر می‌گرداند.

```
int getFastestPlayerReachCycle() const;
```

این تابع مقدار زمانی که طول می‌کشد تا سریع‌ترین بازیکن(بدون در نظر گرفتن تیم) به توپ برسد را بر می‌گرداند.

```
int getSelfReachCycle() const;
```

این تابع مقدار زمانی که طول می‌کشد تا خود بازیکن به توپ برسد را بر می‌گرداند.

```
const Player * getFastestTeammate(bool withMe = true) const;
```

این تابع سریع‌ترین بازیکن هم تیمی ای که به توپ می رسد را به صورت یک اشاره گر بر می‌گرداند.
در ورودی این تابع می‌توانید مشخص کنید که خود بازیکن را هم حساب کند یا نه. اگر به این ورودی مقدار ندهید به صورت پیش‌فرض خود بازیکن هم در هم تیمی ها حساب می شود.
دقت کنید که اگر FastIC کسی را به عنوان سریع‌ترین بازیکن هم تیمی پیدا نکند این تابع Null برمی‌گرداند.

```
const Player * getFastestOpponent() const;
```

این تابع سریع‌ترین بازیکن هم حریفی که به توپ می رسد را به صورت یک اشاره گر بر می‌گرداند.
دقت کنید که اگر FastIC کسی را به عنوان سریع‌ترین بازیکن حریف پیدا نکند این تابع Null برمی‌گرداند.

```
const Player * getFastestPlayer() const;
```

این تابع سریع‌ترین بازیکنی که به توپ می رسد را به صورت یک اشاره گر بر می‌گرداند.

دقت کنید که اگر FastIC کسی را به عنوان سریع‌ترین بازیکن پیدا نکند این تابع Null برمی‌گرداند.

```
FastIC f(worldModel);
Vector ballNewSpeed;
ballNewSpeed.setAsPolar(3.0, 0); // سرعت در زاویه صفر با طول ۳
f.setBall(worldModel->getBall().getPos(), ballNewSpeed, 1); // فرض شود توپ یک سایکل دیرتر حرکت می کند
f.refresh();
f.calculate();
if (f.isOurTeamBallPossessor())
{
	LOG << "EXCELLENT!!! Age bezanam zire toop, toop miofte daste kodemoon!" << endl;
}
```

برای فهمیدن اینکه در حال حاضر با اطلاعات اصلی توپ و بازیکن ها چه بازیکن هایی Fastest هستند، لازم نیست خودتان FastIC بسازید. برای این کار می‌توانید از تابع getGlobalFastIC در کلاس WorldModel که یک FastIC برمی‌گرداند استفاده کنید.

مثال:

File: OffensePlan.cpp Line: 43

```
void Offense::decide(Form &form)
{
	Tackle tackle(worldModel);
	if (worldModel->getGlobalFastIC().isSelfFastestTeammate())
	{
		if (worldModel->isBallKickable())
		{
			Shoot shoot(worldModel);
			Dribble dribble(worldModel);
			Pass pass(worldModel);
			if (shoot.execute())
			{
				command = shoot.getCommand();
			}
			else if(dribble.getValue() >= pass.getValue())
			{
				dribble.decide(form);
				command = dribble.getCommand();
			}
			else
			{
				pass.decide(form);
				command = pass.getCommand();
			}
		}
		else
		{
			if(tackle.execute(form))
			{
				command = tackle.getCommand();
			}
			else
			{
				Intercept intercept = Intercept(worldModel);
				intercept.getValue();
				intercept.execute(form);
```

```
                                        command = intercept.getCommand();
                                }
                        }
                }
        else
        {
                positioning();
        }
}
}
```

در صورتی که سؤالی درباره‌ی FastIC دارید که در این مقاله به آن اشاره نشده است آن را به آدرس mersadbase@gmail.com بفرستید.