# مستند پروژه

# NASA ANTS Simulation

## سیستم‌های خود تطبیق

استاد دکتر ناظمی

آموزشیار دکتر یعقوبی


**اعضای گروه**

الهام استدلالی – بهزاد خسروی فر


(مسیر گیت پروژه: https://github.com/bezzad/NasaAntsSimulation.git)

بهمن سال ۱۳۹۸

## ❁ کتابخانه مورد استفاده

در راستای پیاده سازی شبیه ساز پروژه خود تطبیقی تصمیم بر آن شد که از همان شبیه ساز تست شده در محیط .NET. استفاده شود. لذا تمام اصول شی گرایی را در آن رعایت کردیم و درچندین سیستم مختلف آن را تست کردیم. بدلیل استفاده پروژه تست از کتابخانه Tao Framework و منقضی شدن این کتابخانه توسط شرکت OpenTK، تصمیم گرفتیم آن را با کتابخانه جایگزین شرکت OpenTK عوض کنیم.

نام کتابخانه های جایگزین OpenTK و OpenTK.GLControl می باشد، که می توان از مدیریت پکیج NuGet آنها را دریافت و نصب نمود. این پکیج ها با ورژن 3.1.0 بصورت پیش فرض در پروژه اضافه شده اند و درصورت کامپایل پروژه بصورت خودکار دانلود خواهند شد.

## ❁ برای رعایت شی گرایی در شبیه ساز تغییرات زیر انجام شد:

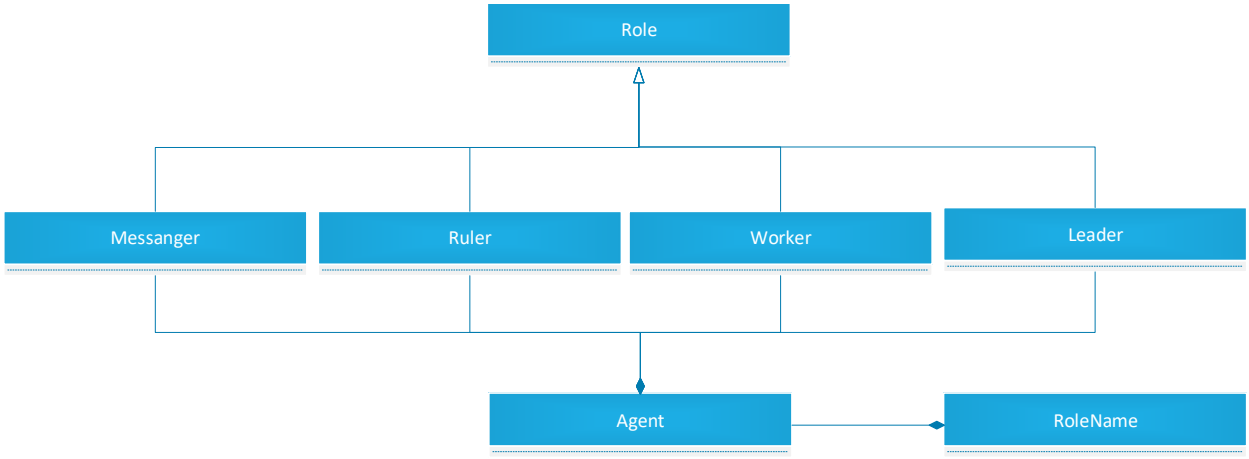
1. دسترسی لایه های بالایی (Program و MainForm) به لایه های پایینی (مانند کلاس Agent) و عدم دسترسی انواع داده ای لایه های پایینی به کلاس های لایه های بالایی اعمال شد.
2. در کلاس GUI فقط رسم شکل های پایه ای مانند دایره و مثلث، خط و نقطه انجام خواهد شد و نه روال کار شبیه ساز.
3. اطلاعات پیکربندی محیط شبیه ساز بجای آنکه از کلاس Program گرفته شود و لایه های پایینی مانند GUI و Agent ها مستقیما با صدا زدن Program به آن پیکربندی دسترسی داشته باشند، نوع داده ای از Config برای نگه داشتن اطلاعات پیکربندی ایجاد می شود و به تمام کلاس هایی که به آن نیاز دارند فرستاده خواهد شد.
4. هر نوع کارگر (Worker) مانند Agent، Messenger و ... برای آنکه بتوانند شکل خود را رسم کنند، خود حاوی اطلاعات مکان و شکل و حرکت هستند و آن را در خود نگه می دارند.
5. وقتی تیم ها بصورت رندوم در Container قرار می گیرند، ممکن است از تیم n ام به بعد جایگذاری تیم ها بقدری مشکل باشد که در حلقه بینهایت گیر بیفتد. از آنجایی که مختصات مرکز تیم در صفحه بصورت تصادفی انتخاب می شود، حالت بهینه ای برای آن وجود ندارد. بدلیل اینکه همیشه فضای هدررفت فضای صفحه وجود دارد، بنابراین مقدار فضای لازم جهت قراردهی تیم ها همیشه بیشتر از مقدار اندازه داده شده فضا، توسط کاربر می باشد. برای حل این مشکل از روش تقسیم صفحه به مربع های بزرگتر از یک تیم استفاده کردیم، بطوری که طول هر ضلع مربع برای هر تیم، 4 برابر شعاع آن تیم

است. در ضمن تقسیم بندی صفحه به تعداد مربع های کنارهم قرار گرفته، باعث جلوگیری از Overlapping تیم ها خواهد شد.

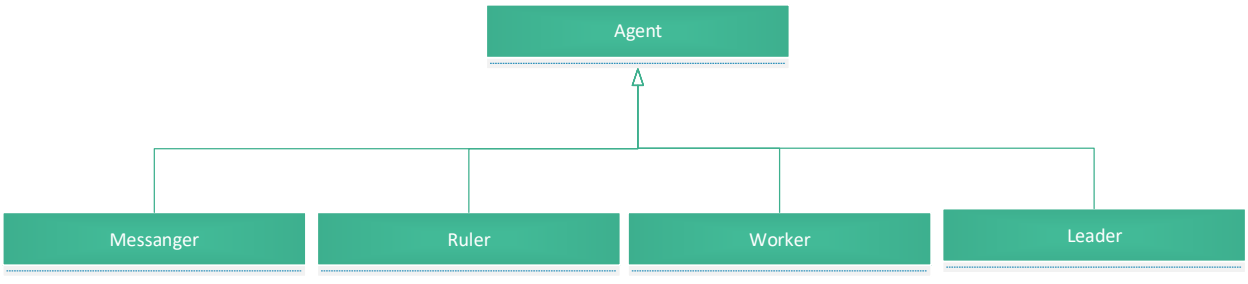
قسمت تخصیص فضای مربع ها داخل MainForm.SetConfiguration انجام شد و قسمت اختصاص مختصات مرکز هر تیم داخل هر مربع در تابع Container.InitialOrgBoundries انجام شد.

6. در کلاس Container بجای درنظر گرفتن اعداد ثابت، بر حسب اندازه صفحه فرم و تعداد تیم ها، Area هر تیم رسم می شود.
7. کلاس Role و RoleName می توانند حذف شود، زیرا تمام مشخصات آن با کلاس Agent یکی می باشد. بنابراین همه از کلاس Agent ارث بری خواهد کرد.

ساختار کلاس برنامه قبل از تغییر:




ساختار کلاس برنامه بعد از تغییر:



## ❁ کلاس Agent:

- دستورات مربوط به حرکت Agent ها (تابع Movement و تابع رسم شکل Draw) همگی به داخل کلاس مربوطه Agent انتقال یافته و شکل و نحوه حرکت آنها بهبود یافت، بطوریکه حرکت پرشی ندارند.
- **متد FindNearestMessenger**:

در کلاس Agent متدی بنام FindNearestMessenger تعبیه شده است تا هم در کلاس های فرزند قابل دسترس باشد و هم بتواند قابلیت دسترسی به نزدیکترین Messenger را برای همه فراهم کند.

- **پیاده سازی Agent های غیرفعال یا شکست خورده:**

1) کلاس جدیدی بنام Fault Generator در Core برنامه ایجاد شد که حاوی چهار تابع بنام های Ruler Failure، Messenger Failure، Leader Failure و Worker Failure است.
2) هر کلاس مربوط به Agent دارای خصوصیتی بنام Status هستند که وضعیت را در دو حالت Stable و Failed نشان می دهد. با استفاده از توابع Failure، وضعیت Agent تغییر می کند و در لحظه ارسال پیام فقط Agent دارای وضعیت Stable قادر به پاسخ گویی است.
3) همچنین در هنگام Fail شدن هر یک از Agent ها بواسطه تابع Draw هریک، رنگ آن Agent به رنگ قرمز در می آید.
4) در فرم اصلی برنامه برای هریک از توابع Failure دکمه ای درنظر گرفته شده است. بدین صورت می توان در هر زمان که خواستیم بصورت تصادفی یکی از عامل ها را منهدم کنیم. بعنوان مثال با انتخاب دکمه Ruler Failure توسط کاربر، در همان لحظه یکی از Ruler ها دچار شکست شده و دارای وضعیت Failed می شود و با شکل ظاهری قرمز می شود. با هربار زدن این دکمه به تعداد Ruler های دارای وضعیت Failed اضافه می شود.
5) با توجه به سناریو پیاده سازی شده، نتیجه کار را مشاهده نمود.

- **تابع PreProcess**:

در هر یک از توابع Fault Generator ، پیش پردازشی انجام می شود تا «زمان شبیه ساز در لحظه شکست» و «تعداد پیام های ارسال شده تا لحظه شکست» را ثبت کند. دلیل انجام این پیش پردازش این است تا بتوان بعد از خود تطبیقی توسط سناریو پیاده سازی شده، «مدت زمان تا لحظه خودتطبیقی» و «تعداد پیام های ردو بدل شده تا لحظه خودتطبیقی» را بدست آوریم.

- **تابع WorkerFailure**:

هر تیم دارای یک لیست از عامل های Worker می باشد. بنابراین ابتدا از لیست تیم ها، یک تیم بصورت تصادفی انتخاب می شود. سپس از بین Worker های تیم انتخابی یک Worker بصورت تصادفی انتخاب می شود و وضعیت آن به Failed تغییر پیدا می کند.

- **تابع LeaderFailure**:

بعد از انجام عمل پیش پردازش، از لیست تیم ها یکی بصورت تصادفی انتخاب می شود و وضعیت leader آن تیم به Failed تغییر پیدا می کند.

- **تابع RulerFailure**:

بعد از انجام عمل پیش پردازش، از لیست Ruler ها بصورت تصادفی یک Ruler انتخاب می شود به شرطی که آن Ruler حداقل یک Leader داشته باشد. سپس وضعیت Ruler انتخاب شده به Failed تغییر پیدا می کند.

- **تابع MessengerFailure**:

بعد از انجام عمل پیش پردازش، از لیست Messenger **ها**، یکی بصورت تصادفی انتخاب شده و وضعیت آن به Failed تغییر پیدا می کند.

- **تابع SendMessage**:

این تابع بصورت Protected پیاده سازی شده است و قابل استفاده در تمامی کلاس های فرزند (Worker, Messenger, ...) می باشد. این تابع جهت ارسال پیام از یک Agent به Agent مقصد استفاده می شود.

- **تابع UpdateOneMilisecond**:

این تابع بصورت Overridable پیاده سازی شده است و در هر کلاس فرزند علاوه بر کد نوشته شده در کلاس Agent، دستورات مربوط به خود را در این تابع اجرا می کنند.

- **تابع OnMessege**:

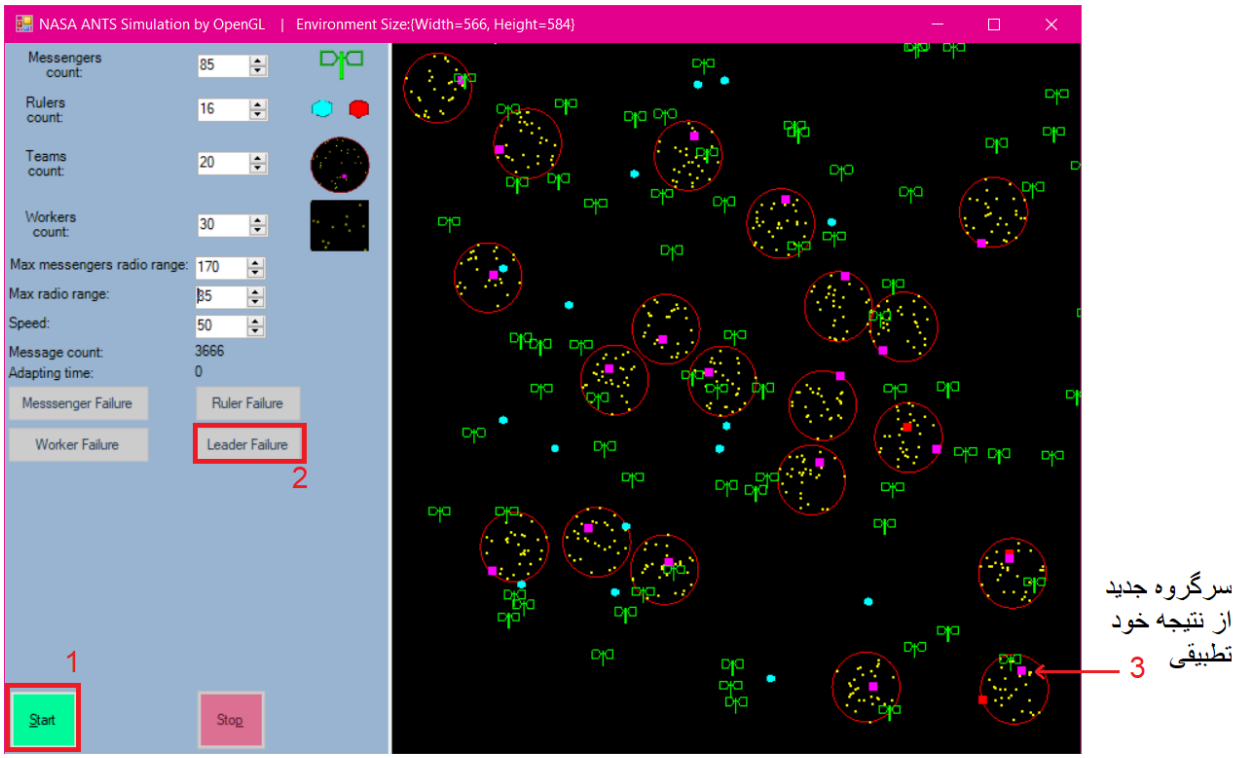
زمانی که یک پیام برای یک Agent ارسال می شود، این تابع در Agent مربوطه فعال می شود.

## ❁ کلاس Team:

تیم های محیط شبیه سازی شده را می سازد. زمانی که یک تیم ساخته می شود، اطلاعات آن تیم به سرگروه آن تیم ارجاع داده می شود. بنابراین یک سرگروه، کارگرهای تیم خود را می شناسد و همچنین زمانی که کارگرها ساخته می شوند، اطلاعات سرگروه تیم به آنها داده می شود.

## ❁ فرم اصلی برنامه

در این فرم با تغییر تنظیمات محیط شبیه ساز و زدن دکمه Start محیط شبیه سازی شروع به کار می‌کند. سپس با انتخاب Leader Failure نتیجه کار قابل مشاهده خواهد بود.




*شکل 1: فرم اصلی برنامه*

## ❁ سناریوی انهدام سرگروه و Self-Healing

سناریو پیاده سازی شده، انهدام سرگروه و عمل خودتطبیقی Self-Healing می باشد. همانطور که گفته شد، در هر تیم یک عامل کارگر نقش سرگروه (Leader) را بازی می کند. سرگروه نقش نماینده گروه برای تبادلات خارج از گروه را بازی می کند. اطلاعات تمامی عاملهای کارگر عضو گروه توسط این عامل تجمیع شده و اطلاعات خارجی نیز تا حد امکان به این عامل ارجاع می شوند. هر سرگروه قادر به تبادل اطلاعات با راهنمای مربوط به خود، پیام رسانها و سرگروه های دیگر تیم ها است. برای هر عامل راهنما محدوده مشخصی در نظر گرفته شده و هر عامل راهنما با سرگروه های تیم هایی که در محدوده وی هستند ارتباط برقرار می کند. پیام رسانها قادر به تبادل اطلاعات با تمام عاملها هستند.

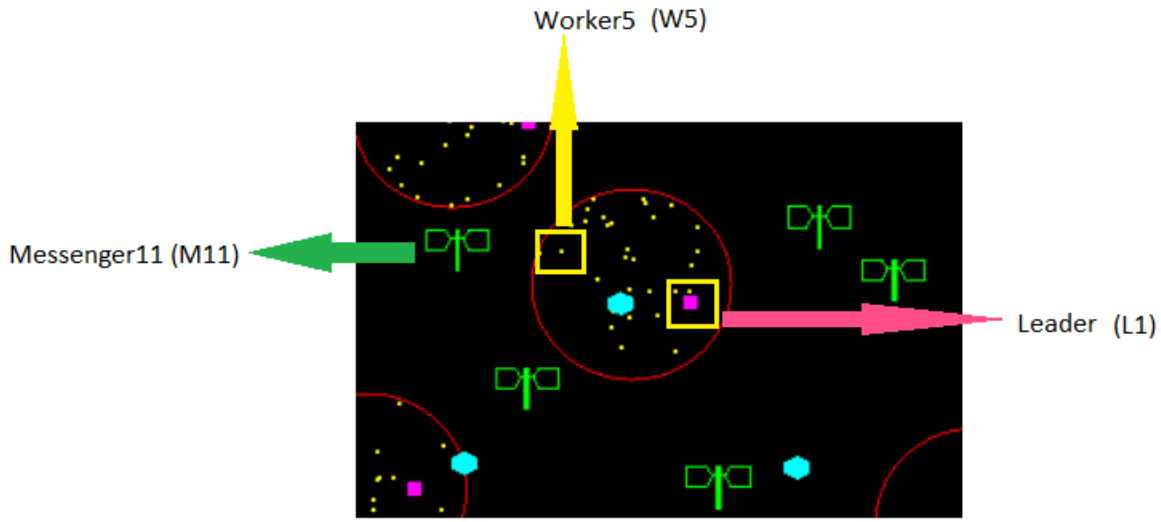
با توجه به توضیحات داده شده، سرگروه ها و کارگرهای تیم برای ارتباط با یکدیگر از پیام رسان ها استفاده می کنند. سرگروه، کارگرهای تیم خود را می شناسد و کارگرها نیز سرگروه خود را می شناسند. اگر کارگری بخواهد اطلاعاتی را به سرگروه خود بفرستد، ابتدا نزدیکترین پیام رسان به خود را پیدا می کند و پیام را به آن می فرستند. پیام رسان، با توجه به اطلاعاتی که از تیم ها دارد پیام دریافتی را به سرگروه آن تیم ارسال می کند. اگر سرگروه بعد از مدتی، جواب پیام رسان را ندهد به احتمال زیاد سرگروه گم شده است.

- ممکن است این پیام برای رسیدن به مقصد از چند پیام رسان میانی عبور کند. در واقع مسیریابی یک پیام ممکن است چند مرحله ای باشد. در هر مرحله پیام رسانی انتخاب می شود که مجموع فاصله اش به فرستنده و گیرنده کمتر از دیگر پیام رسانها باشد.
- اگر در تبادل پیام، یک عامل نتواند پیام رسانی را در محدوده خود پیدا کند، به دلیل اینکه محدوده
- عامل ها تغییر می کند، صبر می کند تا پیام رسانی در محدوده رادیویی اش قرار گیرد.

## نحوه انجام مراحل خودتطبیقی:

کارگرها در بازه های زمانی مشخص، سرگروه خود را از طریق نزدیکترین پیام رسان به خود Ping می کنند و هریک پیام مربوط به Ping را در لیست ReplyWaitingList مربوط به خود درج می کنند.

در شکل زیر یک Worker (W5) با سرگروه خود (L1) درارتباط است:




W5 برای ارسال پیام، نزدیکترین پیام رسان به خود را پیدا کرده (M11) و پیام اولیه ای با ساختار زیر می سازد و L1 را در لیستی که نام عامل هایی که به آن ها پیام از نوع ping فرستاده است ذخیره می کند. درصورتی

که بعنوان مثال بعد از 50ms جواب این پیام به W5 برنگشت، W5 متوجه می شود که احتمالا سرگروه L1 دچار مشکل شده است.

| Ping: L1 | From: W5 | Interface Receiver: M11 | Final Receiver: L1 |
|---|---|---|---|

توسط مکانیزمی این لیست در بازه های زمانی مشخص چک می شود تا اگر بعد از این مدت جواب ping به کارگر نیامد، عدم پاسخ Ping پدیدار شود. بعد از وقوع این اتفاق، کارگر دوباره درخواست خود را به پیام رسان ارسال می کند. این بار محتوای پیام کارگر «گم شدن سرگروه» می باشد.

پیام رسان با دریافت این پیام دوباره سرگروه مربوطه را ping می کند تا از گم شدن آن مطمئن شود. همچنین این پیام ping جدید را در لیست ReplyWaitingList خود نگه می دارد تا درصورت عدم دریافت پاسخ از سرگروه، با قطعیت اعلام کند که سرگروه گم شده است. بدین صورت کارگر و و پیام رسان وظیفه پایش را عهده دار می شوند.

درصورتی که پیام رسان پیامی از سرگروه دریافت نکند، پاسخ کارگر را با محتوای اینکه سرگروه جدید همان کارگر خواهد یود، به همراه اطلاعات تیم آن کارگر ارسال می کند و منتظر جواب کارگر می ماند. کارگر مربوطه با گرفتن این پیام، خود را سرگروه اعلام کرده و اطلاعات تیم را هم خواهد داشت. سپس برای اعلام سرگروه جدید به بقیه اعضا تیم، کارگر مربوطه پیامی به پیام رسان می فرستد تا بصورت Broadcast به تمامی اعضا تیم ارسال شود. بدین صورت تمام کارگران، سرگروه جدید را خواهند شناخت. همچنین این کارگر، خود را از لیست کارگران تیم حذف می کند.

در این روش عامل ها با پذیرش نقش تکمیلی خودتطبیق در قالب سازمانهای خودتطبیق عملیات مورد نظر را انجام می دهند و وضعیت سیستم به حالت عادی باز می گردد.