مستند پروژه

NASA ANTS Simulation

درس سیستم‌های خود تطبیق

استاد دکتر ناظمی

آموزشیار دکتر یعقوبی

اعضای گروه

الهام استدلالی – بهزاد خسروی فر

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 باشـــد، 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.
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 ...ببرای اینکه بتوانند شکل خود را رسم کنند، خود حاوی اطلاعات مکان و شکل و حرکت هستند و آن را در خود نگه می دارند.
5. در روش قبلی برای پیاده سازی انواع سناریو، یک ساختار Enum بنام Scenario ساخته شده بود که حاوی مقادیر Scenario1, Scenario2, .. Scenario5 بودند. حال نحوه پیاده سازی این کد در شبیه ساز ما با Polymorphism پیاده سازی شده است. بگونه ای که کلاس Container یک Property بنام SimulationScenario از نوع Interface IScenario دارد، که در زمان پیاده سازی به آن مقدار سناریو مورد نظر اختصاص داده می شود که کلاس سناریو مورد نظر باید Interface مذبور را پیاده سازی کرده باشد. مانند:

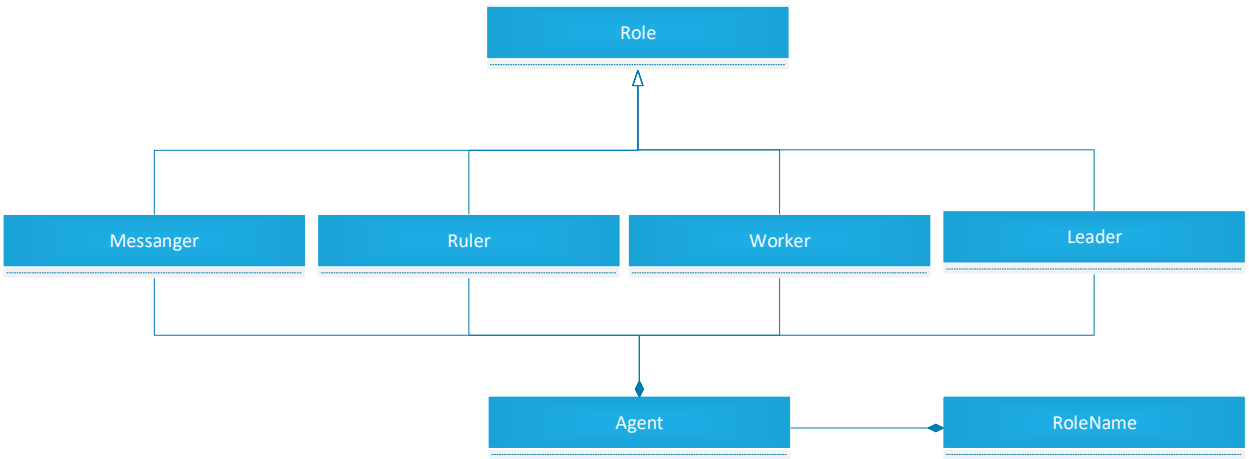
```
public IScenario SimulationScenario { get; } = new SelfHealingScenario1();
```

6. وقتی تیم ها بصورت رندوم در Container قرار می گیرند، ممکن است از تیم n ام به بعد جایگزاری 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. به این دلیل که همیشه فضای هدررفت فضای صفحه وجود دارد، بنابراین مقدار فضای لازم جهت قدار دهی 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.

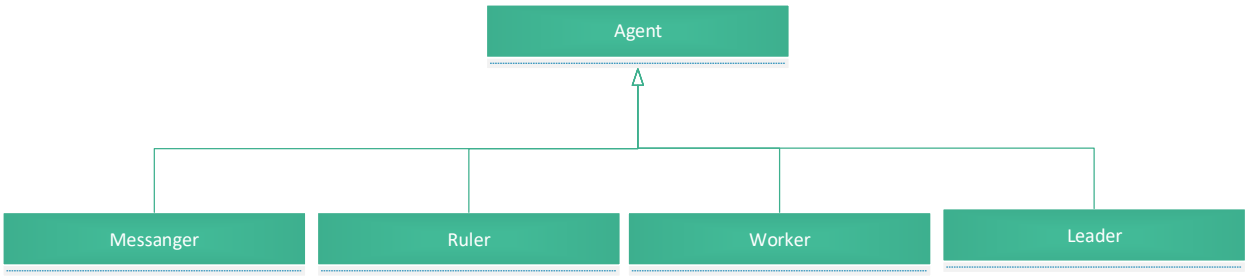
قسمت تخصیص فضای مربع ها داخل MainForm.SetConfiguration انجام شد و قسمت اختصاص مختصات مرکز هر تیم داخل هر مربع در تابع Container.InitialOrgBoundries انجام شد.

7. در کلاس Container بجای درنظر گرفتن اعداد ثابت، بر حسب اندازه صفحه فرم و تعداد تیم ها، Area هر تی رسم می شود.
8. 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.
باهربار زدن این دکمه به تعداد Ruler های دارای وضعیت Failed اضافه می شود.

5) بعد از آن می توان با انتخاب سناریو مورد نظر، 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 مورد نظر، «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 بین 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**:

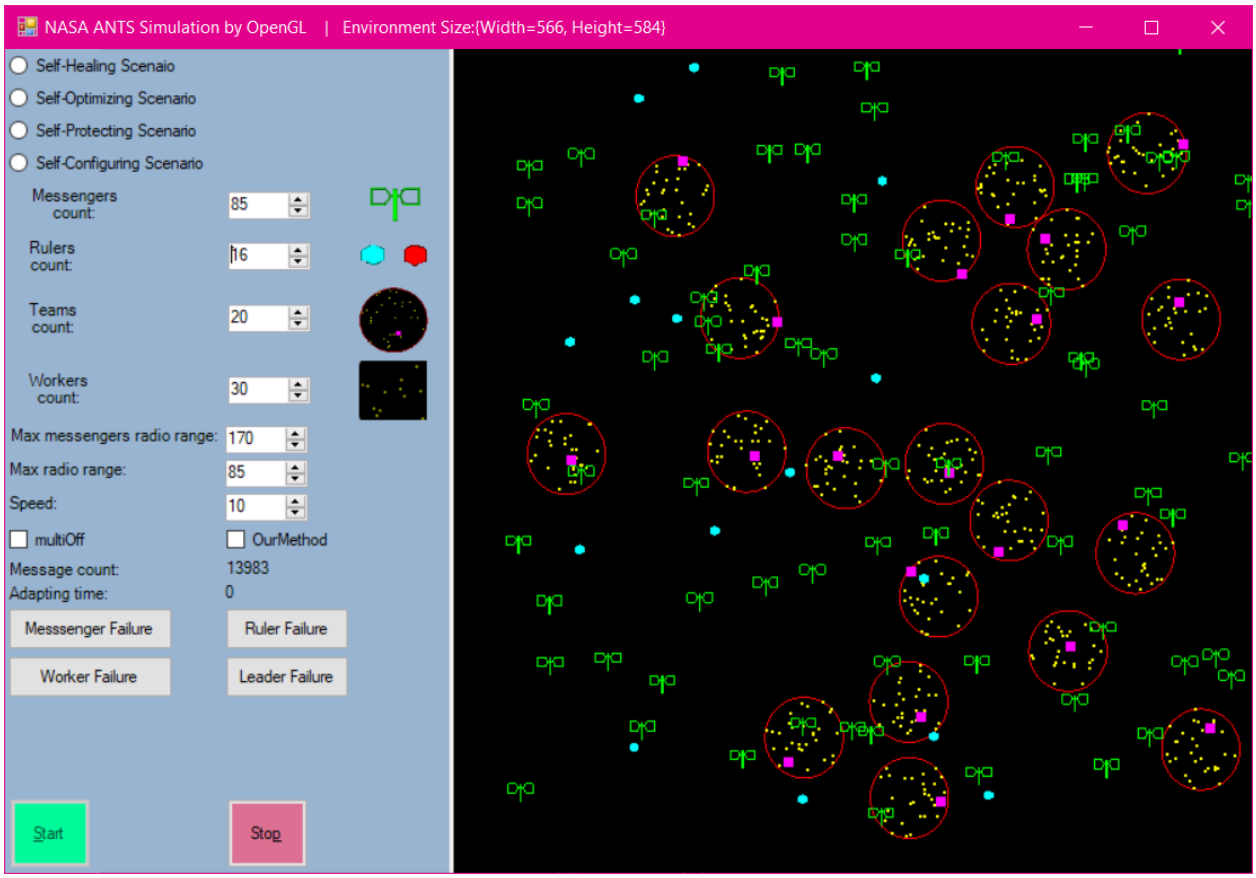
زمانی که یک پیام برای یک Agent ارسال می شود، این تابع در Agent مربوطه فعال می شود.

❁ 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

فرم اصلی برنامه در شکل زیر قابل مشاهده می باشد. با تغییر تنظیمات محیط شبیه ساز و زدن دکمه Start محیط شبیه سازی آماده می شود. سپس با انتخاب سناریو مورد نظر و Failure موردنظر نتیجه کار قابل مشاهده خواهد بود.



❁ سناریوی انهدام سرگروه (Leader)

سناریو انتخابی ما، سناریو انهدام سرگروه می باشد. همانطور که در سناریو شبیه سازی گفته شد، در هر تیم یک عامل کارگر نقش سرگروه (Leader) را بازی می کند. سرگروه نقش درگاه ارتباطی و نماینده گروه برای تبادلات خارج از گروه را بازی میکند. اطلاعات یافته شده توسط 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 سرگروههای دیگر تیم ها است. برای هر عامل راهنما محدوده مشخصی در نظر گرفته شده و هر عامل راهنما با سرگروههای تیمهایی 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 برای ارتباط با کارگرهای تیم 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 اطلاعات را به آن می فرستند. پیام رسان اطلاعات دریافتی را با توجه به اطلاعاتی که از تیم ها دارد به سرگروه آن تیم ارسال می کند. اگر سرگروه بعد از مدتی، جواب پیام رسان را ندهد به احتمال زیاد سرگروه منهدم شده است. درنتیجه دو حالت در پیش داریم:

1) اگر سرگروه پاسخی به پیام رسان نداد، پیام رسان تشخیص می دهد که سرگروه منهدم شده و یکی از کارگر ها را بعنوان سرگروه انتخاب می کند.

2) اگر پیام رسان پاسخی دریافت نکرد، کاری نمی کند و پاسخ کارگر را هم نمی دهد. در نتیجه کارگر بعد از گذشت زمان مشخصی به پیام رسان اعلام می کند که از سرگروه جوابی دریافت نکرده است و سپس پیام رسان برای او سرگروه پیدا می کند.

سناریو انتخابی ما حالت اول را پیاده سازی کرده است.

ادامه دارد...