



پایان نامه مقطع کارشناسی مهندسی کامپیوتر

پیاده سازی آزمون‌های مورد نیاز شبیه ساز دوبعدی فوتبال مبتنی

بر یکپارچه سازی مداوم

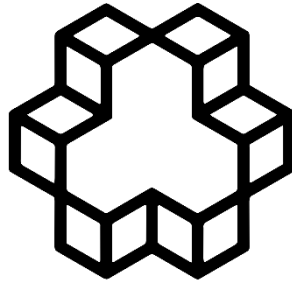
مجتبی موذن

استاد راهنما:

دکتر سعید صدیقیان کاشی

شهریور ۹۹

السلام الرحيم الرحيم



دانشگاه صنعتی خواجه نصیرالدین طوسی
دانشکده مهندسی کامپیوتر

تأییدیه هیات داوران

هیات داوران پس از مطالعه پایان نامه و شرکت در جلسه دفاع از پایان نامه تهیه شده تحت عنوان:

پیاده سازی آزمون های مورد نیاز شبیه ساز دوبعدی فوتبال مبتنی بر یکپارچه
سازی مداوم

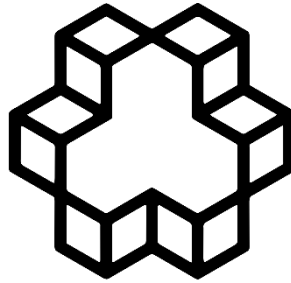
توسط آقای مجتبی موذن صحت و کفایت تحقیق انجام شده را برای اخذ درجه کارشناسی
در رشته مهندسی کامپیوتر مورد تأیید قرار می دهند.

امضاء

استاد راهنما آقای دکتر سعید صدیقیان کاشی

امضاء

استاد ارزیاب آقای دکتر مهدی اثنی عشری



دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده مهندسی کامپیوتر

اظهارنامه دانشجو

اینجانب مجتبی مؤذن دانشجوی مقطع کارشناسی رشته مهندسی کامپیوتر گواهی می‌نمایم که تحقیقات ارائه شده در پروژه با عنوان:

پیاده سازی آزمون های مورد نیاز شبیه ساز دوبعدی فوتبال مبتنی بر یکپارچه سازی مداوم

با راهنمایی استاد محترم جناب آقای دکتر سعید صدیقیان کاشی توسط شخص اینجانب انجام شده است. صحت و اصالت مطالب نگارش شده در این پروژه مورد تأیید می‌باشد و در تدوین متن پروژه چارچوب (فرمت) مصوب دانشگاه را به طور کامل رعایت کرده‌ام.

امضاء دانشجو:

تاریخ: ۹۹/۷/۳۰

حق طبع، نشر و مالکیت نتایج

- ۱- حق چاپ و تکثیر این پروژه متعلق به نویسنده و استاد راهنمای آن می‌باشد. هرگونه تصویربرداری از کل یا بخشی از پروژه تنها با موافقت نویسنده یا استاد راهنما یا کتابخانه دانشکده مهندسی برق و کامپیوتر دانشگاه صنعتی خواجه‌نصیرالدین طوسی مجاز می‌باشد.
- ۲- کلیه حقوق معنوی این اثر متعلق به دانشگاه صنعتی خواجه‌نصیرالدین طوسی می‌باشد و بدون اجازه کتبی دانشگاه به شخص ثالث قابل‌واگذاری نیست.
- ۳- استفاده از اطلاعات و نتایج موجود پروژه بدون ذکر مرجع مجاز نمی‌باشد.

تقدیم به :

تقدیم به پدر و مادر عزیزم که با مهربانی و صبر مرا به این نقطه از زندگی رسانده‌اند.

تشکر و قدردانی :

از خانواده‌ی عزیزم ، اساتید گرامی، دکتر سعید صدیقیان کاشی ، دکتر مهدی اثنی عشری و تمامی دوستانم که در مراحل مختلفی از زندگی همراهم بوده‌اند تشکر میکنم.

چکیده

شبیه‌سازی دوبعدی فوتبال ، فضایی شبیه سازی شده از فوتبال واقعی است که به دوبعد محدود است. این محیط شبیه‌سازی شده یکی از بسترهای مسابقاتی مسابقات روبوکاپ است که هر ساله در سطح بین‌المللی برگزار می‌شود [۱]. در این فضا هر تیم دارای ۱۱ بازیکن و یک مربی است . هر بازیکن دارای دید مستقل و محدود و دارای خطا^۱ می‌باشد ، اما دید مربی کاملاً بدون خطا است [۱]. یکی از چالش‌های تیم‌های شبیه سازی دوبعدی فوتبال بررسی روند رشد تیم با استفاده از آزمون‌های متفاوتی است که هر تیم برای سنجش میزان بهبودی خود می‌تواند اجرا کند . از طرفی اجرای خودکار آزمون‌های متفاوت در تیم‌ها باید به گونه‌ای باشد که اعضای تیم بتوانند به صورتی دقیق و کاربردی روند نزولی و یا صعودی معیارهای مهم بازی‌ها را مشاهده کنند . آنچه که در این پروژه در نهایت پیاده‌سازی شده است پیاده‌سازی آزمون‌های خودکار و در نهایت بررسی آزمون‌ها با استفاده از رابط کاربری مناسب برای هر کدام از اعضای یک تیم شبیه‌سازی دوبعدی است .

کلید واژه: مهندسی کامپیوتر، مهندسی نرم‌افزار، یکپارچه‌سازی مداوم، توسعه نرم‌افزار

^۱ Noise

فهرست مطالب

۵.....	فصل اول : مقدمه.....	۱
۶.....	۱-۱ پیشگفتار.....	
۷.....	۲-۱ تاریخچه.....	
۷.....	۳-۱ کد پایه.....	
۸.....	Agent۲D..... ۱-۳-۱	
۹.....	LibRCSC..... ۲-۳-۱	
۹.....	Server..... ۳-۳-۱	
۱۰.....	۴-۱ سنجش بهبود عملکرد یک تیم شبیه سازی دوبعدی فوتبال.....	
۱۱.....	فصل دوم : معرفی مفاهیم کلی.....	۲
۱۲.....	۱-۲ مقدمه.....	
۱۲.....	۲-۲ نگاه کلی به روند اجرای آزمون در یک تیم شبیه سازی دوبعدی فوتبال.....	
۱۲.....	۱-۲-۲ روند اجرای آزمون در بستر شبیه سازی دوبعدی فوتبال.....	
۲۰.....	فصل سوم : شرح ویژگیهای ابزار آزمون و مشاهده ی نتایج در شبیه سازی دوبعدی فوتبال.....	۳
۲۱.....	۱-۳ مقدمه.....	
۲۱.....	۲-۳ آزمون ها.....	
۲۲.....	۱-۲-۳ آزمون AutoTest.....	
۲۲.....	۲-۲-۳ آزمون AutoTest formation.....	
۲۵.....	۳-۳ روند اجرای آزمونها به صورت خودکار.....	
۲۶.....	خط لوله..... ۱-۳-۳	
۲۷.....	گیتلب..... ۲-۳-۳	
۳۴.....	۴-۳ مراحل اجرای خط لوله و راه اندازی آن و دریافت نتایج.....	
۳۶.....	۱-۴-۳ فایل YML. و توضیحات مربوط به پیکربندی خط لوله.....	
۳۷.....	۲-۴-۳ مانیتورینگ نتایج.....	
۴۵.....	فصل چهارم : جزئیات پیاده سازی خط لوله و ابزار مشاهده نتایج.....	۴
۴۶.....	پیاده سازی و پیکربندی خط لوله..... ۱-۴	

۴۸ Stages ۱-۱-۴ کلمه ی کلیدی
۴۸ پیکربندی کار ۲-۱-۴
۵۲ پیاده سازی آزمون ها ۲-۴
۵۲ AutoTest آزمون ۱-۲-۴
۵۲ AutotestFormation آزمون ۲-۲-۴
۵۵ پیاده سازی و ساخت فایل تصویر داکری ۳-۴
۵۷ پیاده سازی ابزار مقایسه نتایج ۴-۴
۵۷ سمت کاربر ۱-۴-۴
۵۹ سمت سرور ۲-۴-۴
۶۲ سورس کنترل ۵-۴
۶۳ ۵ - نتیجه گیری و پیشنهادات
۶۴ ۱-۵ پیشنهادات
۶۴ پیاده سازی آزمونهای متنوع دیگر ۱-۱-۵
۶۵ پیاده سازی مشاهده ی نتایج در ابزار مانیتورینگ نتایج ۲-۵
۶۷ فهرست مرجع ها

۱ فصل اول : مقدمه

۱-۱ پیشگفتار

در دنیای امروز هوش مصنوعی^۱ به عنوان یکی از ارکان علم کامپیوتر به حساب می آید [۲] و همواره بسترهای متفاوتی برای اجرای الگوریتم های مختلف و همچنین ایده های متفاوت این علم پیاده سازی و اجرا شده است. شبیه ساز دوبعدی فوتبال به عنوان یک بستر^۲ هوش مصنوعی از سال ۱۹۹۵ در مسابقات روبوکاپ^۳ به کار گرفته شده است. یکی از اهداف مهم این لیگ پیاده سازی الگوریتم هایی در هوش مصنوعی است که در نهایت بتواند در سال ۲۰۵۰ مقابل یک تیم از انسان های فوتبالیست، به پیروزی دست پیدا کند.

در شبیه سازی دوبعدی فوتبال همه ی بازیکن های داخل زمین دید^۴ مستقل و محدودی دارند که برای نزدیک تر شدن هر چه بیشتر این لیگ به واقعیت دنیای فوتبال، دید بازیکنان داخل زمین دارای خطا است [۳] و تنها شخصی که می تواند به صورت دقیق و بدون خطا بازی را دنبال کند مربی تیم است که می تواند اطلاعات خود را به صورت محدود در اختیار بازیکنان قرار دهد. بازیکنان در شبیه ساز دوبعدی فوتبال درست همانند فوتبال واقعی امکان انجام اعمالی نظیر شوت و تکل را در هر زاویه دلخواه دارند و در صورت انجام فعالیت بیش از حد انرژی^۵ خود را از دست می دهند و حرکت آن ها کند می شود.

^۱ Artificial Intelligence

^۲ platform

^۳ RoboCup

^۴ View

^۵ Stamina



تصویر ۱- نمایی از محیط بصری شبیه‌سازی دوبعدی فوتبال

۲-۱ تاریخچه

اولین بار در سال ۱۹۹۰ آقای مک‌ورث ایده‌ی برگزاری مسابقات فوتبال ربات‌ها را داد ولی در ابتدا به این ایده توجه زیادی نشد تا این که در سال ۱۹۹۳ تعدادی از محققان هوش مصنوعی نیاز به وجود بستری برای اجرای الگوریتم‌های پیشرفته را احساس کردند و در سال ۱۹۹۵ اولین دوره مسابقات شبیه‌سازی دوبعدی فوتبال تحت عنوان مسابقات روبوکاپ برگزار شد و پس از آن لیگ‌های متفاوتی از جمله لیگ ربات‌های امدادگر و تیم فوتبال‌یست‌های ساینز کوچک و بزرگ و متوسط و تعدادی لیگ دیگر به این مسابقات اضافه شد. مهم‌ترین هدف در این لیگ حل مسائل هوش مصنوعی و موضوعات مرتبط با آن بوده است. پس از برگزاری مسابقات در سال ۱۹۹۵ کمیته‌ی جهانی روبوکاپ تشکیل شد که هدف آن ساماندهی مسابقات در جهت پیشرفت هر چه بهتر هوش مصنوعی عنوان گردید. [۴]

۳-۱ کد پایه

در سال ۱۹۹۳ اولین سرور^۱ شبیه‌سازی دوبعدی فوتبال با محدودیت‌های گرافیکی عرضه شد. این سرور

^۱ Server

اولیه بازیکن ها را به شکل کاراکتر^۱ در صفحه ی کنسول^۲ نشان می داد اولین نسخه ی سرور که پایه اصلی سرورهای کنونی شبیه سازی دوبعدی فوتبال است در سال ۱۹۹۴ عرضه شد [۳] و در دسترس علاقه مندان به این لیگ قرار گرفت . پس از آن سعی شد تا تغییراتی در سرور مسابقات ایجاد شود که باعث نزدیک تر شدن هر چه بیشتر این لیگ به فوتبال واقعی شود . آخرین نسخه ی سرور شبیه سازی دوبعدی فوتبال که نسخه ۱۴ آن می باشد در سال ۲۰۱۸ با قابلیت دادن اظهارهای کارت قرمز و زرد ارائه شد که این نسخه نیز به صورت متن باز^۳ در اختیار علاقه مندان قرار گرفت.

شبیه سازی دوبعدی فوتبال به طور کلی شامل سه بخش مهم است : [۳]

۱. Agent^۴D

۲. LibRCSC

۳. Server

۱-۳-۱ Agent^۴D

کد شبیه سازی دوبعدی شامل یک بخش به نام کد پایه است که در واقع پیاده سازی بخش های مختلف و الگوریتم های اصلی و ایده های متفاوت بازی در داخل همین بخش انجام می شود . تیم های بسیاری تا به حال کد پایه شامل این بخش را به صورت عمومی در دسترس سایر تیم ها قرار داده اند که از جمله اینها تیم های راییتیگل^۴ و هلیوس^۵ می باشند . در سال ۲۰۱۰ تیم هلیوس کد پایه^۶ خود را به صورت عمومی که با زبان ++C نوشته شده بود را به صورت متن باز در اختیار سایر تیم ها قرار داد و تیم های نوپای زیادی با استفاده از این کد پایه توانستند پا به عرصه مسابقات بین المللی بگذارند .

^۱ character

^۲ console

^۳ Open Source

^۴ WrightEagle

^۵ HELIOS

^۶ Agent base code

LibRCSC ۲-۳-۱

کد پایه‌ی تیم هلیوس دارای یک کتابخانه بود که در آن رفتارهای پایه‌ای محیط فوتبال پیاده‌سازی شده است. این رفتارهای پایه‌ای شامل دویدن، شوت زدن، تکل زدن، پاس دادن و رفتارهایی نظیر نگاه‌داشتن توپ بود که برای تفکیک بهتر تحت عنوان یک کتابخانه^۱ در اختیار قرار گرفت.

این رفتارها خود به سطوح مختلفی تقسیم میشوند که در زیر به تعدادی از این رفتارها اشاره میکنیم:

- رفتارهای سطح صفر شامل شوت زدن و پاس دادن و دویدن و یک سری از حرکات بسیار اساسی
- رفتارهای سطح یک شامل رفتن به نقطه‌ای مشخص از زمین و نگاه‌داشتن توپ
- رفتارهای سطح دو مثل سد کردن توپ حریف و یا مارک^۲ کردن بازیکنان حریف

لازم به ذکر است، رفتارهای سطح دوم در کد منبع پیاده‌سازی نشده است ولی باقی این دستورات شامل رفتارهای سطح یک و صفر در کتابخانه و کد پایه پیاده‌سازی شده است و برنامه نویسان برای پیاده سازی ایده‌های خود می‌توانند از آن‌ها استفاده کنند.

Server ۳-۳-۱

سرور مسابقات برنامه‌ای است که زمین فوتبال و تمام عوامل دخیل در یک مسابقه فوتبال واقعی مانند وزن توپ، وزن بازیکنان، سرعت بازیکنان و تعداد گل‌ها و همچنین جهت وزش باد در بازی را کنترل می‌کند. اگر از نگاه شیء‌گرایی^۳ به این نرم افزار نگاه کنیم مشخصاتی همچون وزش باد و وزن توپ، جزو صفات زمین به حساب می‌آید و توابعی مثل محاسبه‌ی مختصات توپ و بازیکن جزو رفتارهای زمین فوتبال به حساب می‌آیند. لازم به ذکر است زمان در بازی‌های فوتبال در این بستر شبیه‌سازی، به قطعات جدا از هم به نام سیکل^۴ تقسیم شده است که هر سیکل یک دهم ثانیه است و زمان بازی‌ها ۶۰۰۰ سیکل است. همچنین

^۱ library

^۲ mark

^۳ Object Oriented

^۴ cycle

مثل دنیای واقعی فوتبال در این بستر ، وقت اضافه در صورت مساوی شدن تیم ها وجود دارد که ۱۵۰۰ سیکل می باشد و در صورت مساوی شدن در وقت اضافه ، بازی به ضربات پنالتی کشیده خواهد شد . تمام بازیکنان در این بستر برای دریافت اطلاعات می توانند به سرور مسابقات درخواست داده و اطلاعات مهمی که مورد نیاز است را از آن دریافت کنند و هر کدام تجزیه و تحلیل های متفاوتی را روی این اطلاعات اعمال کنند . [۵]

۱-۴ سنجش بهبود عملکرد یک تیم شبیه سازی دوبعدی فوتبال

در دنیای شبیه سازی دوبعدی فوتبال همواره معیارهای متفاوتی برای سنجش بهبود عملکرد کلی تیم پس از پیاده سازی ایده ها توسط اعضا وجود دارد . امروز تیم های فعال ، هر یک با ایده های متفاوت که توسط اعضای تیم پیاده سازی می شوند سعی در ارتقای سطح فنی خود دارند . معیارهای کلی برای سنجش بهبود عملکرد تیم ها وجود دارد که اکنون اعضای یک تیم از آن ها برای ارتقای سطح فنی خود استفاده می کنند . این معیارها می تواند شامل تعداد گل های خورده و یا زده شده و یا تعداد برد و باخت های تیم باشد . این معیار ها خود به چند دسته متفاوت تقسیم می شوند ، برخی از آن ها به صورت کلی وضعیت تیم مورد نظر را مورد بررسی قرار می دهند (مثل تعداد گل زده شده) و برخی از آن ها سعی دارند به صورت آزمون های واحد^۱ عمل کنند مثل سنجش عملکرد دروازه بان در برابر موقعیت های تک به تک که به صورت واحد به عملکرد دروازه بان و کدی که برای آن نوشته شده می پردازد. در این پروژه قصد داریم به پیاده سازی بستری برای انجام آزمون ها و همچنین مشاهده ی نتایج آن ها و در نتیجه مانیتور^۲ کردن کد یک تیم شبیه سازی دوبعدی فوتبال بپردازیم .

^۱ Unit Tests

^۲ monitor

۲ فصل دوم : معرفی مفاهیم کلی

۱-۲ مقدمه

در این فصل به معرفی ابزار آزمون و مانیتور^۱ در شبیه سازی دوبعدی فوتبال به عنوان یک بستر برای پیاده سازی آزمون ها و مشاهده روند تیم های فعال در این لیگ می پردازیم .

۲-۲ نگاه کلی به روند اجرای آزمون در یک تیم شبیه سازی دوبعدی فوتبال

در یک تیم شبیه سازی دوبعدی فوتبال جهت بهبود عملکرد تیم ، ایده های متفاوت توسط اعضای یک تیم قابل پیاده سازی است که هر کدام از اعضا برای مشاهده روند بهبودی و یا نزولی کد پس از پیاده سازی با توجه به ایده ی مورد نظر ، ممکن است آزمون های متفاوتی را به صورت غیر خودکار بر روی تیم خود پیاده سازی کنند و از نتایج آن برای بهبود ایده ها و جلوگیری از روند نزولی تیم استفاده کنند . می توان گفت یکی از ملزومات هر تیم نرم افزاری با توجه به دخالت هر یک از اعضا در بخشی از آن ، ایجاد یک سامانه یکپارچه و منظم جهت انجام آزمون های مورد نیاز و مشاهده ی نتایج آن است .

همواره یکی از مشکلات تیم های شبیه سازی دوبعدی ، نبود این بستر و در نهایت انجام آزمون ها به صورت غیر خودکار و نامنظم بوده است که باعث کندشدن روند تولید ایده ها می شود. از طرفی نبود بستری برای مشاهده نتایج و انجام عمل مانیتورینگ در یک تیم ، باعث نادیده گرفته شدن نقاط ضعف و قوتی می شود که در طول اجرا و پیاده سازی ایده ها وجود دارند.

۱-۲-۲ روند اجرای آزمون در بستر شبیه سازی دوبعدی فوتبال

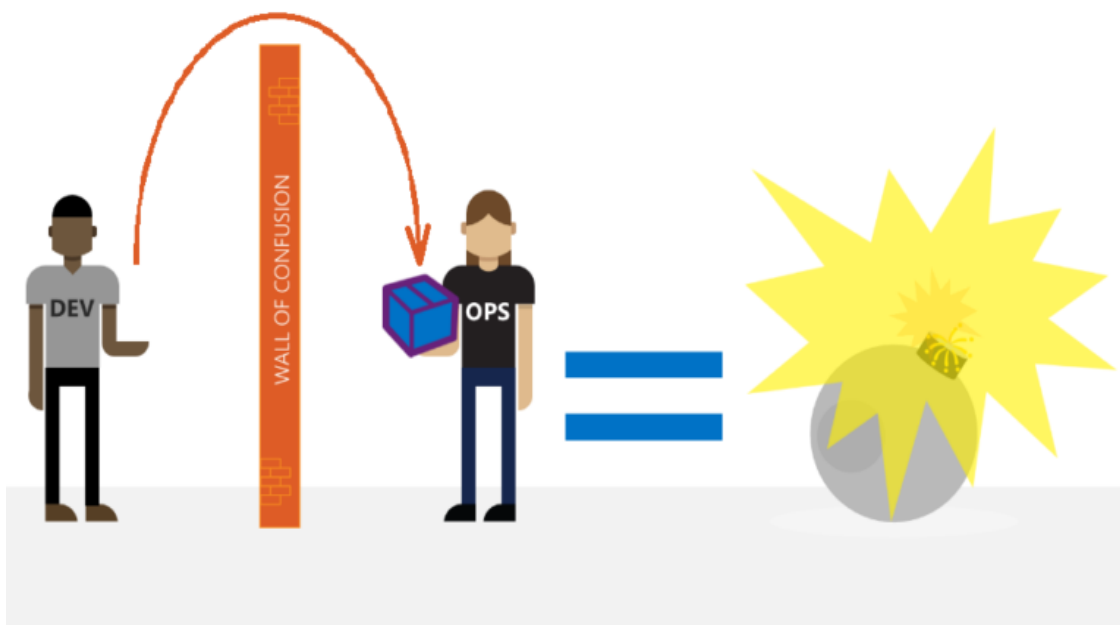
در این پروژه هدف اول پیاده سازی بستری مناسب مبتنی بر یکپارچه سازی مداوم بوده است که افراد داخل هر تیم با استفاده از آن ، روال انجام آزمون ها را به صورت خودکار دنبال کنند و بتوانند بدون دخالت در ایده های یک دیگر آزمون های گوناگونی را روی ایده های متفاوت پیاده سازی شده، اجرا کنند. این عمل

^۱ Monitor

باعث می‌شود تا اعضای هر تیم گزارشی صحیح از عملکرد خود در تیم داشته باشد و آن را به رهبران تیم انتقال دهند.

۲-۱-۲-۱ توسعه عملیات

توسعه عملیات یا دواپس^۱ مجموعه‌ای از روش‌ها و فرآیندهایی است که با تمرکز بر ارتباطات و همکاری و همچنین یکپارچگی بین تیم‌های توسعه نرم‌افزار^۲ و عملیات^۳ نرم‌افزارهای تولید شده را به صورت سریع و مداوم به مشتریان نهایی می‌رسانند. از نظر تاریخی، سابقاً در تیم‌های نرم‌افزاری دو تیم مستقل از هم تحت عنوان تیم توسعه و تیم نرم‌افزار وجود داشته است که به مرور زمان و با گذشت و توسعه نرم‌افزارهای بزرگتر جای خود را به تیم‌های چند تخصصی با مهارت‌ها و روش‌های متفاوت داده‌اند. [۶]



تصویر ۲- دیوار موجود بین تیم توسعه و عملیات باعث روند کند تولید محصول میشود [7]

یکی از دلایلی که باعث شد مفاهیم این دو تیم یعنی تیم توسعه و تیم نرم‌افزار با هم درآمیزد، محدودیت

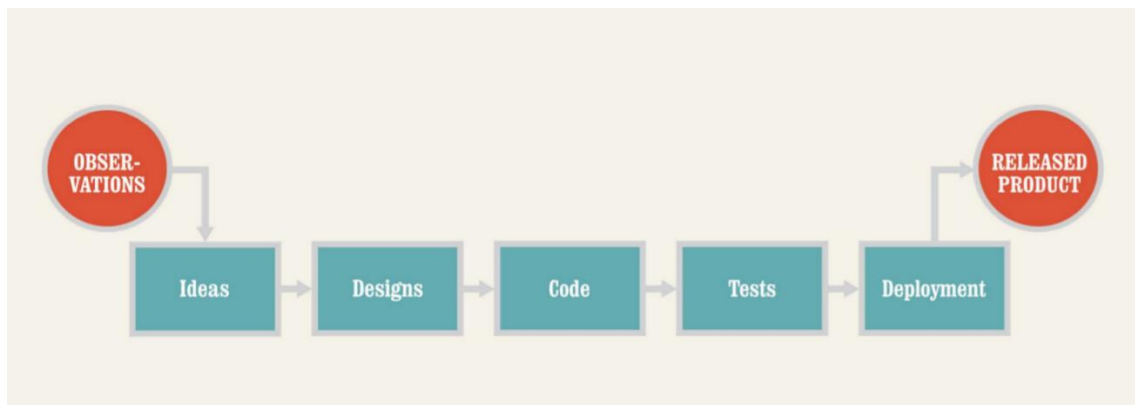
^۱ DevOps

^۲ Software Developer

^۳ Operation Team

هایی بود که باعث می شد در یک تیم نرم افزاری که با روش های توسعه نرم افزار چابک^۱، کار می کنند، از سرعت تولید نرم افزار کاسته شده و نتوانند با یکدیگر فعالیت و ارتباط خوبی برقرار کنند. به دنبال این محدودیت ها، مفهوم توسعه عملیات یا همان دواپس مطرح شد که باعث شد دیوار بین دو تیم توسعه و عملیات تا حدودی برداشته شده و روند تولید نرم افزار با تمرکز بر افزایش تعاملات بین تیمی، بهبود فایل توجهی یابد.

در یک تیم توسعه و عملیات مراحل مختلفی از ایده تا تولید محصول نهایی اجرا میشود که در شکل زیر به طور خلاصه این مراحل آورده شده است:



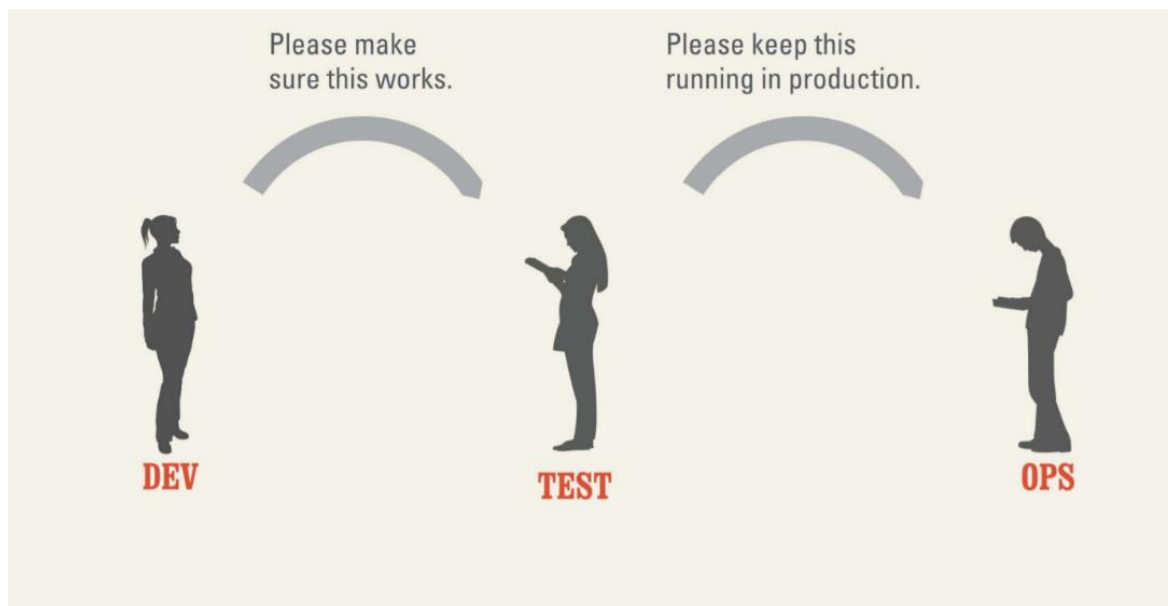
تصویر ۳- مراحل تولید یک محصول در تیم توسعه عملیات [8]

۱. مرحله ای ایده و طراحی: در یک تیم شبیه سازی دوبعدی این مرحله شامل ایده های تئوری و نحوه ی پیاده سازی آنها و همچنین الگوریتم هایی است که از آنها برای بهبود عملکرد تیم استفاده می شود.

۲. مرحله پیاده سازی: در این مرحله هر یک از اعضا ایده های خود را به صورت عملی پیاده سازی می کنند، تا در نهایت خروجی مورد نظر مشاهده شود. پیاده سازی این بخش به همراه بخش

^۱ Agile

بعدی تحت عنوان یکپارچه سازی مداوم^۱ شناخته می شود که شامل پیاده سازی و انجام آزمون - های متفاوتی است که روی این قسمت انجام میشود.



تصویر ۴- روابط موجود بین توسعه دهنده ، تیم آزمون و همچنین تیم عملیات را مشاهده می کنید ، پس از توسعه توسط تیم برنامه نویس و پشت سر گذاشتن آزمون هایی توسط تیم دیگر ، وظیفه ی تیم عملیات اطمینان از اجرایی بودن نرم افزار بر روی بسترها و سیستم عامل های متفاوت است . [4]

۳. مرحله انجام آزمون: پیش از آن که مفهوم توسعه و عملیات با یکدیگر ترکیب شود انجام آزمون بر روی ایده های پیاده سازی شده ، توسط یک تیم جدا از افرادی انجام می شد که وظیفه - ی آنها مورد آزمون قراردادن بخش های مختلفی بود که توسط تیم توسعه ، نوشته شده بود . در یکپارچه سازی مداوم یکی از وظایف تیم دواپس^۲ پیکربندی روالی است که بتوان این مرحله را به صورت خودکار انجام داد و روال توسعه را سریع تر کند. در یک تیم شبیه سازی دوبعدی فوتبال، این بخش شامل آزمون های متفاوتی است که روی ایده ها انجام می شود و می توان معیارهای گوناگونی برای آن ، مثل گل خورده و یا گل زده شده به حریف ، در نظر گرفت .

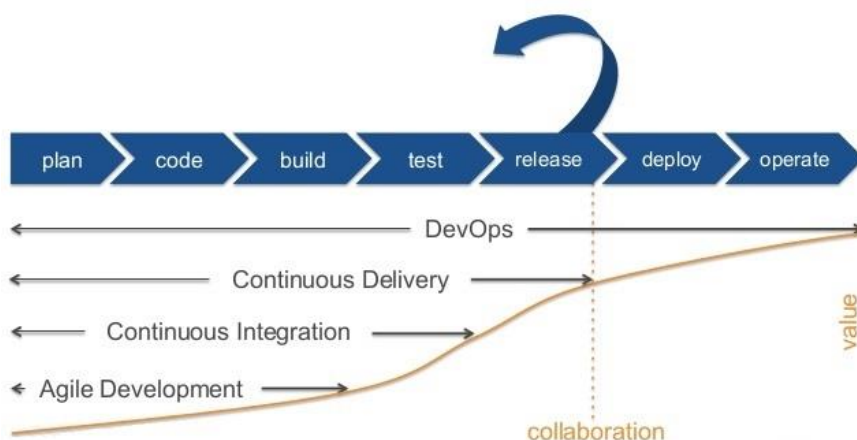
۴. مرحله تولید نرم افزار : در این مرحله نرم افزار مورد نظر ، پس از عبور از آزمون های متفاوت و

^۱ Continuous integration

^۲ DevOps

اطمینان پذیری^۱ بالا ، آماده انتشار میشود .

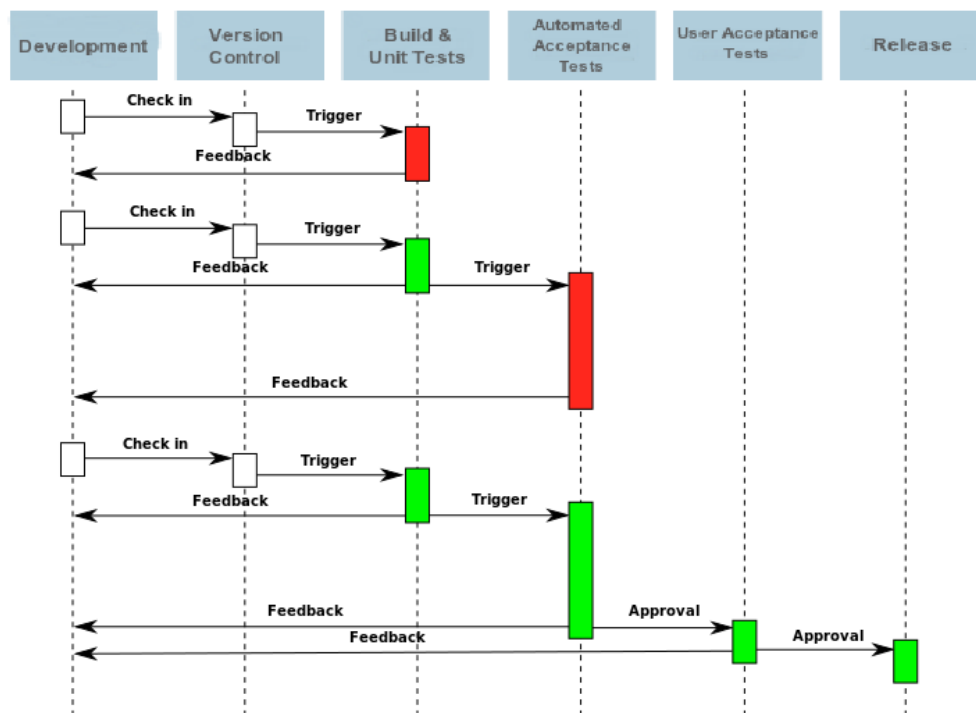
به رویکردی که در آن یک تیم نرم افزاری ، قادر است نرم افزار تولیدشده را به روشی سریع و مطمئن برای انتشار و تحویل آماده کند Continuous Delivery می گویند ، که در آن از لحظه ای که کد مورد نظر تغییر می کند تا زمان انتشار به صورت نرم افزار ، اعمالی نظیر بیلدشدن ، مورد آزمون قرارگرفتن ، پیکربندی و انتشار در محیطها و سیستم عامل های مختلف ، روی کد مورد نظر انجام می شود . یکپارچه سازی مداوم یک بخش کوچک از Continuous Delivery است که بازخوردهای متفاوتی را تا انتشار نرم افزار به تیم توسعه دهنده ارائه می کند.



تصویر ۵

در تصویر به صورت دقیق تر ، دریافت بازخوردها تا زمان انتشار نرم افزار را مشاهده می کنیم .

^۱Software reliability



تصویر ۶- برنامه نویسان با استفاده از بازخوردهایی که در طول یکپارچه سازی مداوم از ایده های خود دریافت می کنند و تکرار این مراحل ، میتوانند از قابلیت اطمینان بالای ایده های خود مطمئن شوند. [9]

همانطور که در شکل بالا ملاحظه میفرمایید ، هدف از یکپارچه سازی مداوم ، اجرای خودکار^۱ و مداوم آزمون های مورد نظر بر روی ایده ها و دریافت بازخورد^۲ از آن ها و در نهایت پیشرفت یک تیم نرم افزاری است.

۲-۱-۲-۲ آزمون های مورد نیاز یک تیم شبیه سازی دوبعدی فوتبال

در یک تیم شبیه سازی دوبعدی فوتبال ، برای مسائل گوناگون ، ایده های متفاوتی مطرح میشود که عموماً این ایده ها در دوبرخش دفاعی و حمله ، خلاصه میشود . در بخش دفاعی برای مثال ایده های سد کردن توپ توسط یک بازیکن خط دفاع و بهبود عملکرد دروازه بان و همچنین بهبود وضعیت مکانی یک بازیکن در خط دفاع و جلوگیری از ضدحمله ، از جمله الزامات یک تیم خوب و موفق به شمار می رود . دربخش حمله ی تیم هم ایده هایی نظیر بهبود وضعیت قرارگیری بازیکنان موجود در خط حمله ، از جمله ویژگی های یک تیم

^۱ Automatic testing

^۲ Feedback

خوب است .



تصویر ۷- پیاده سازی الگوریتم ورونوی^۱ توسط تیم رباتیک KN2C جهت بهبود وضعیت دفاعی

در این پروژه به طور کلی دو نوع آزمون مورد نیاز به صورت خودکار پیاده سازی شده است :

۱. آزمون AutoTest : هدف از این آزمون اجرای بازی های متعدد است ، همانگونه که در ابتدای

فصل گذشته اشاره شد ، در یک بازی فوتبال در شبیه سازی دوبعدی ، به جهت هرچه نزدیک تر

شدن به دنیای واقعی فوتبال ، اطلاعات دریافتی دارای کمی خطا می باشد ، که این خطا توسط

سرور مسابقات (که بازیکن برای دریافت اطلاعات به آن درخواست می دهد) ایجاد می شود .

مقدار این خطا توسط سرور مسابقات ، محدودی ، به صورت عادلانه بین هر دو تیم مسابقه

تقسیم می شود ، ولی تاثیر این مقدار خطا در یک بازی ممکن است با یک بازی دیگر متفاوت

باشد و دلیل این تفاوت به دلیل تاثیر گذاری در موقعیت های متفاوت است . به همین جهت ،

^۱ voronoi algorithm

در یک بازی شبیه‌سازی شده ، یک بازی واحد ، تعیین کننده‌ی برآیند کلی تیم نیست و می‌تواند نتایج بسیار خوب و یا نتایج بسیار بد آن ، به جهت تاثیرگذاری خطا در آن بازی باشد. از این رو ، این آزمون پیاده‌سازی شده است تا بتوان با استفاده از آن برآیند خوبی از وضعیت تیم داشت و روند صعودی یا نزولی تیم را مقایسه کرد.

۲. آزمون AutoTest Formation : هر تیم فوتبال در دنیای واقعی با یک سیستم چینش به زمین فوتبال می‌رود که تاثیر این سیستم چینش در بازی فوتبال ، بسیار زیاد است . خصوصا در شبیه‌سازی دوبعدی فوتبال ، تاثیر چینش بازیکنان در خطوط دفاعی و حمله ، بسیار زیاد است. در این آزمون تعدادی بازی با استفاده از هر کدام از سیستم‌های چینش بازیکنان اجرا می‌شود و نتایج آن به صورت‌های متفاوت در اختیار اعضای تیم قرار می‌گیرد .

۲-۱-۲-۳ انجام آزمون‌ها به صورت خودکار

همانگونه که ذکر شد در بحث یکپارچه‌سازی مداوم و اجرای آزمون‌های مرتبط با یک تیم شبیه‌سازی دوبعدی ، انجام آزمون‌ها به صورت خودکار از اهمیت بالایی برخوردار است . در این بستر شبیه‌سازی برای مورد آزمون قرار گرفتن ایده‌ها ، آزمون‌هایی طراحی و پیاده‌سازی شده است که با استفاده از اجرای آن‌ها و مشاهده نتایج ، می‌توان به بهبود وضعیت کلی تیم کمک کرد.

در فصل بعد به چالش‌های موجود در روند پیاده‌سازی این ابزار و همچنین پیاده‌سازی آزمون‌های مورد نیاز به صورت دقیق‌تر می‌پردازیم .

۳ فصل سوم : شرح ویژگی‌های ابزار آزمون و مشاهده‌ی نتایج در

شبیه‌سازی دوبعدی فوتبال

۳-۱ مقدمه

در فصول قبل به معرفی اجمالی و اهداف پروژه پرداختیم. همانطور که ذکر شد اهداف این پروژه پیاده سازی یک بستر مناسب برای هر تیم شبیه سازی دوبعدی فوتبال است که با استفاده از آن بتوان اجرای آزمون های مورد نیاز را به صورت خودکار انجام داد و همچنین با استفاده از یک ابزار ثانویه، نتایج را به صورت دیداری مشاهده کرد و آن ها را مورد مقایسه انجام داد.

این پروژه به صورت کلی از سه بخش بسیار مهم تشکیل شده است:

۱. بخش اول شامل آزمون هایی است که برای این منظور نوشته شده است که در انتهای فصل

قبل به صورت اجمالی به آن ها اشاره شد.

۲. بخش دوم شامل پیکربندی روند اجرای آزمون ها به صورت خودکار است که ابزارها و

تکنولوژی های مورد استفاده در این بخش به صورت کامل در ادامه توضیح داده خواهد شد.

۳. بخش سوم شامل پیاده سازی ابزار مانیتورینگ است که در آن هر یک از اعضای یک تیم

می توانند به صورت کامل نتایج آزمون های خود را با آزمون های دیگر مقایسه کنند.

در ادامه در این فصل به بررسی چالش های پیاده سازی و پیکربندی این سه بخش می پردازیم.

۳-۲ آزمون ها

همواره یکی از مشکلات تیم های شبیه سازی دوبعدی فوتبال، پیاده سازی یک آزمون خوب برای بررسی

روند صعودی یا نزولی تیم ها بوده است. در این پروژه دو آزمون مهم را که نشان دهنده وضعیت کلی یک

تیم فوتبال است در دو بخش پیاده سازی و پیکربندی شده است.

۳-۲-۱ آزمون AutoTest

این آزمون که به صورت متن باز در اختیار تیم های شبیه ساز دوبعدی فوتبال قرار گرفته است تعدادی بازی فوتبال (به انتخاب کاربر) در مقابل یکی از تیم های حریف اجرا کرده و در آخر و پس از تمام شدن کامل آن خروجی آن نمایش داده میشود. اطلاعاتی که پس از تمام شدن این آزمون نمایش داده می شود شامل :

- تعداد بازی های اجرا شده
- تعداد گل خورده و میانگین تعداد گل خورده در هر بازی مقابل حریف (برای سنجش خط دفاع)
- تعداد گل زده شده و میانگین تعداد گل زده مقابل حریف (برای سنجش خط حمله)
- درصد برد و باخت مقابل تیم حریف
- تفاضل گل های موجود در بازی های متفاوت بر مبنای تعداد بازی ها

نحوه ی کارکرد این آزمون به این شکل است که در هر بار اجرا شدن آن ، بسته به سخت افزاری که بازی ها روی آن اجرا می شود می توان به صورت گسسته و پشت سر هم تعدادی بازی اجرا کرد . برای مثال اگر کاربر قصد دارد ۲۰۰ بازی را اجرا کند می تواند آن را در ۴۰ دوره ی زمانی که در هر دوره ۵ بازی به صورت موازی اجرا می شود اجرا کند . این آزمون به صورت پایه ای توسط تیم رایتینگل نوشته شده و در دسترس عموم قرار داده شده است ولی به منظور خودکار سازی آزمون و استخراج اطلاعات بیشتر در این پروژه مورد بازبینی و پیاده سازی مجدد قرار گرفته است .

۳-۲-۲ آزمون AutoTest formation

یکی دیگر از آزمون های مورد نظر ، آزمونی است که در آن با استفاده از هر کدام از سیستم های چینش بازیکنان در زمین ، تعدادی بازی فوتبال با استفاده از سیستم^۱ مورد نظر اجرا می شود و نتایج بازی های

^۱ Formation

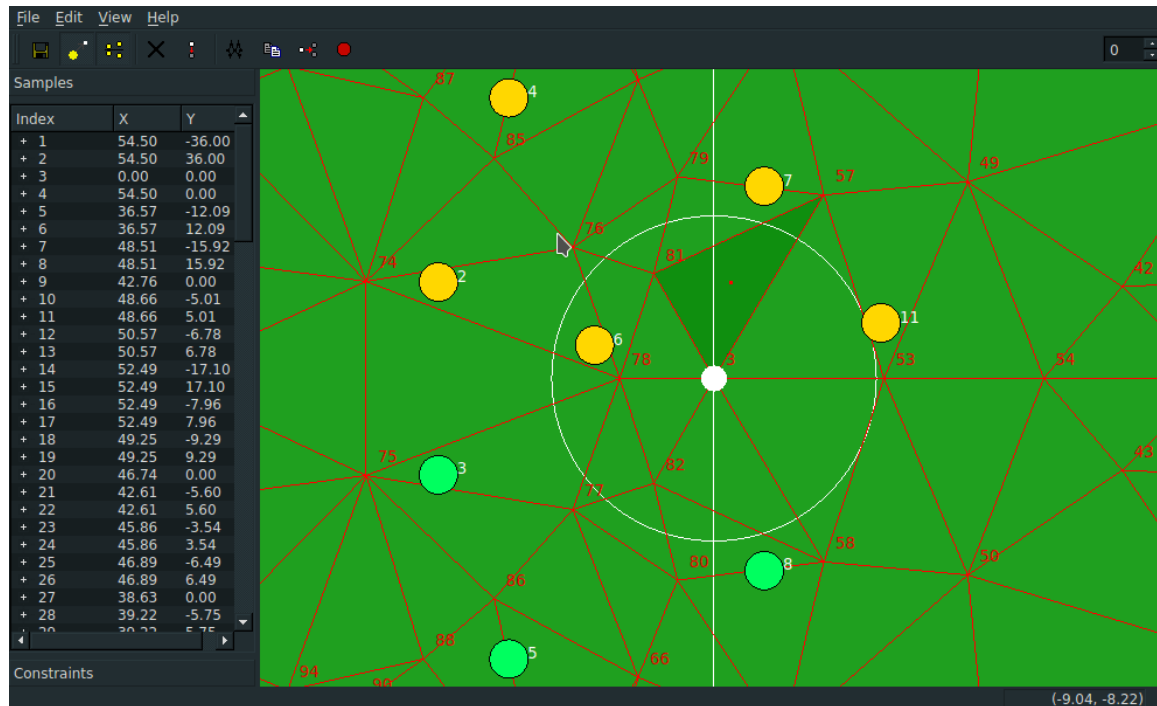
اجرا شده به عنوان خروجی مورد تجزیه و تحلیل قرار می‌گیرد. هدف از این آزمون این است که به طور دقیق‌تر یک تیم شبیه‌سازی دوبعدی فوتبال را با سیستم‌های متفاوت بسنجیم و پی ببریم که کدام سیستم چینش بازیکنان در زمین بیشتر مناسب تیم مورد نظر است. یکی از چالش‌های مهم در شبیه‌سازی فوتبال، مقدار انرژی بازیکنان در طول بازیکن و محدود بودن آن است، به طوری که ممکن است یک سیستم چینش، باعث حرکت و جابه‌جایی بیش از حد بازیکنان در زمین فوتبال و در نهایت باعث از دست رفتن انرژی آن‌ها در طول بازی شود. در صورت رخداد این اتفاق، بازیکنان خط دفاعی در معرض ضدحمله‌های حریف قرار خواهند گرفت که خصوصاً در پایان دو نیمه‌ی بازی می‌تواند در نتایج بازی تاثیرگذار باشد. با استفاده از این آزمون سرعت رسیدن یک تیم به سیستم چینش موردنظر و مطلوب خود، افزایش داده شده است. از آن جهت که ایده‌های متفاوتی را می‌توان در بحث مکان‌یابی و حرکت بازیکنان در یک سیستم چینش مطرح کرد، وجود داشتن چنین آزمونی یکی از ملزومات مهم برای یک تیم شبیه‌سازی دوبعدی بود.

۳-۲-۱ ساخت سیستم‌های متفاوت چینش بازیکنان

چینش بازیکنان در زمین به این شکل تعریف می‌شود:

"نسبت دادن یک موقعیت مکانی نسبی مجاز به بازیکنان موثر تیم خودی به ازای موقعیت‌های مکانی (قطعی) متفاوت توپ در نقاط اختیاری زمین"

همانگونه که ذکر شد میتوان برای آزمون مورد نظر، از سیستم‌های متفاوت دفاعی و حمله استفاده کرد. برای تولید این سیستم‌ها در این پروژه از ابزاری استفاده شده است که تیم هلیوس در سال ۲۰۱۰ آن را به صورت متن باز در اختیار قرار داده است که نام آن Fedit است. با استفاده از این ابزار میتوان بازیکنان را، نسبت به موقعیت توپ، در موقعیت‌های متفاوت در زمین قرار داد. در شکل زیر نمونه‌ای از طراحی آرایش تیمی را به عنوان نمونه مشاهده می‌کنید.



تصویر ۸- نمونه ای از طراحی یک آرایش تیمی

شاید این سوال برای ما مطرح شود که آرایش تیمی چگونه کار می کند و چگونه می توان از این ابزار به صورت دقیق تر استفاده کرد . لازم به توضیح است الگوریتم ^۱ پیاده سازی شده این بستر شبیه سازی ، برای چینش بازیکنان از ایده های مثلث های دلانی ^۲ استفاده میکند . در این الگوریتم مجموعه ای از گره ها را روی صفحه ای در نظر می گیریم. سپس آن صفحه را به زیر صفحه های کوچکتری تقسیم می کنیم به طوری که هر زیر صفحه تنها یک گره دربر داشته باشد. فاصله ی هر نقطه ی تصادفی در زیر صفحه تا گره آن زیر صفحه از فاصله ی آن نقطه تا تمامی گره ها کمتر است. از این ایده به این شکل استفاده می شود که کافی است ما نقاط مختلفی را برای قرار گرفتن بازیکنان نسبت به توپ در نظر بگیریم و پس از آن با استفاده از این الگوریتم ، در حین بازی ، با توجه به نقاط بین دومتلی که توپ بین آن جابه جا می شود ، بازیکن مورد نظر نیز بین نقاط تعیین شده در دو مثلث مورد نظر حرکت می کند . ایده ی استفاده از مثلث های دلانی در این قسمت ، به این دلیل است که نمی توان مختصات بازیکن ها را برای هر نقطه ، به دلیل نامحدود بودن نقاط ، به بازیکن

^۱ Algorithm

^۲ Delaunay Triangulation

اطلاع داد . [۱۰]

یک آرایش تیمی خوب دارای اصلی بهینگی است و هدف آزمون مورد نظر ذکر شده در بالا ، سنجش اصل بهینگی در حرکت است . حرکت باید بهینه باشد یعنی نباید بیهوده بازیکنان را جابجا کنیم و افرادی که نمی توانند در موقعیت تأثیر مستقیم بگذارند ، بیهوده انرژی صرف نکنند . برای مثال بازیکنی که در خط دفاع است ، در حین حمله معمولاً جایگیری خاصی برایش در نظر گرفته نمی شود و طبق چینش جایگیری می کند چون تغییر مکان او اگر کم باشد ، به دلیل فاصله ی او از توپ و بازیکنان صاحب توپ نمی تواند در موقعیت ها تأثیرگذار باشد و اگر هم تغییر مکان زیادی داشته باشد ، باید فاصله زیادی را پیموده و انرژی زیادی را صرف کند و در صورت از دست رفتن موقعیت هم ، تیم در شرایط نامناسبی قرار می گیرد ، چون جای خالی بازیکن احساس خواهد شد . یکی از راه های سنجش میزان بهینگی حرکت بازیکنان در زمین ، سنجش تاثیرگذاری چینش آن ها در زمین فوتبال است .

به طور کلی ما در ابزار آزمون آرایش تیمی بازیکنان ، از ۱۰ چینش متفاوت که توسط تیم های متفاوت و ۳ چینش که برای سنجش بهتر پیاده سازی شده است استفاده می کنیم .

۳-۳ روند اجرای آزمون ها به صورت خودکار

همانطور که در فصل قبل مشاهده شد روند اجرای آزمون در یک تیم شبیه سازی دویعدی به صورت خودکار ، یکی از ملزومات پیشرفت و همفکری در یک تیم است . در یک تیم دویعدی ممکن است در طول روز چندین مرتبه ایده های متفاوت با پیاده سازی های متفاوت به اجرا درآید و اجرای آزمون های مورد نظر بر روی هر بخش از آن به صورت غیرخودکار وقت زیادی را از اعضای تیم بگیرد .

۳-۳-۱ خط لوله^۱

به طور کلی به کارهایی که از ابتدای تغییرات برروی یک قسمت از کد و کامیت^۲ کردن آن ، بر روی کد مورد نظر انجام میگیرد خط لوله می گویند . در این پروژه خط لوله شامل سه بخش کلی است :

۱. بیلد^۳ شدن کد شبیه سازی دوبعدی فوتبال: باید بدانیم که هر تکه کدی که در زبان سی پلاس پلاس^۴ نوشته شده است برای اجرا شدن به یک فایل اجرایی^۵ نیازمند است که به روال ساخته شدن این فایل بیلد شدن کد می گویند. در این بخش پس از تغییراتی که هر شخص در تیم برروی قسمتی از کد داده است و پس از کامیت کردن آن ، برای آنکه اطمینان حاصل شود که کد مورد نظر از نظر خطای دستوری^۶ فاقد اشتباه است ، کد به صورت خودکار بیلد شده و پس از گذراندن این مرحله وارد مرحله ی بعدی می شود .

۲. انجام آزمون ها : برای آزمودن ایده ها ، آزمون هایی پیاده سازی می شود که در این مرحله به صورت خودکار برروی کدی که در مرحله قبل تبدیل به فایل اجرایی شده است انجام می شود . این مرحله بسته به انتخاب کاربر میتواند شامل آزمون هایی باشد که از قبل نوشته شده اند و یا خود اعضای تیم آن را نوشته اند .

۳. خروجی آزمون ها برای بررسی و مقایسه ی آن ها : در این مرحله خروجی آزمون به روش های گوناگون به اطلاع هر یک از اعضای یک تیم شبیه سازی دوبعدی فوتبال می رسد.

برای پیکربندی آزمون های خودکار در این پروژه ، از امکاناتی که سایت گیتل^۷ب در اختیار کاربران خود

^۱ Pipeline

^۲ commit

^۳ build

^۴ C++

^۵. exe file

^۶ Syntax Error

^۷ Gitlab

قرار داده است استفاده شده که در ادامه به شرح این پیکربندی‌ها و چالش‌های پیش‌روی آن می‌پردازیم.

۲-۳-۳ گیتلب

ابتدا به این مفهوم می‌پردازیم که گیت چیست؟ گیت یک سیستم کنترل نسخه‌های^۱ متفاوت است که توسط لینوس توروالدز^۲ که سازنده‌ی هسته لینوکس^۳ نیز می‌باشد طراحی شده است. هدف اصلی آن فراهم کردن محیط و امکاناتی است که یک تیم برنامه‌نویسی روی یک پروژه با یکدیگر همکاری کنند. به کمک گیت میتوان تغییرات یک برنامه نرم‌افزاری را ذخیره و ردیابی کرد که این تغییرات توسط تیم برنامه نویس اعمال می‌شود. گیت تمام پروژه‌ای که شما تعریف کرده‌اید را در قالب یک مخزن^۴ نگه‌داری می‌کند و به کمک این مخزن شما می‌توانید پروژه‌های خود را با دیگران به اشتراک بگذارید و آن‌ها را مورد آزمون قرار دهید. حال که با مفهوم گیت آشنایی پیدا کردیم، باید بدانیم که گیتلب یک مخزن تحت وب^۵ است که به برنامه نویسان امکانات زیادی را تحت قالب گیت ارائه می‌دهد. سایت گیتلب یک مخزن مورد اعتماد است که توسط چندین شرکت بزرگ مانند IBM، سونی^۶، علی بابا^۷ مورد استفاده قرار می‌گیرد و امکانات زیادی را در دسترس برنامه نویسان قرار می‌دهد. [۱۱]

یکی از ابزارهای رایجی که سایت گیتلب در اختیار افراد قرار می‌دهد امکان راه‌اندازی یک خط لوله مناسب جهت انجام آزمون‌های برنامه نویسی است. ابزار یکپارچه سازی مداوم گیتلب^۸، توسط مخازن گیتلب پشتیبانی می‌شود که در این پروژه از این ابزار استفاده شده است

در ادامه به تعریف چند مفهوم که در این پروژه با استفاده از این ابزار پیاده‌سازی و استفاده شده است

^۱ Version Controll System

^۲ linus torvalds

^۳ Linux

^۴ Repository

^۵ web

^۶ Sony

^۷ Alibaba

^۸ Gitlab CI

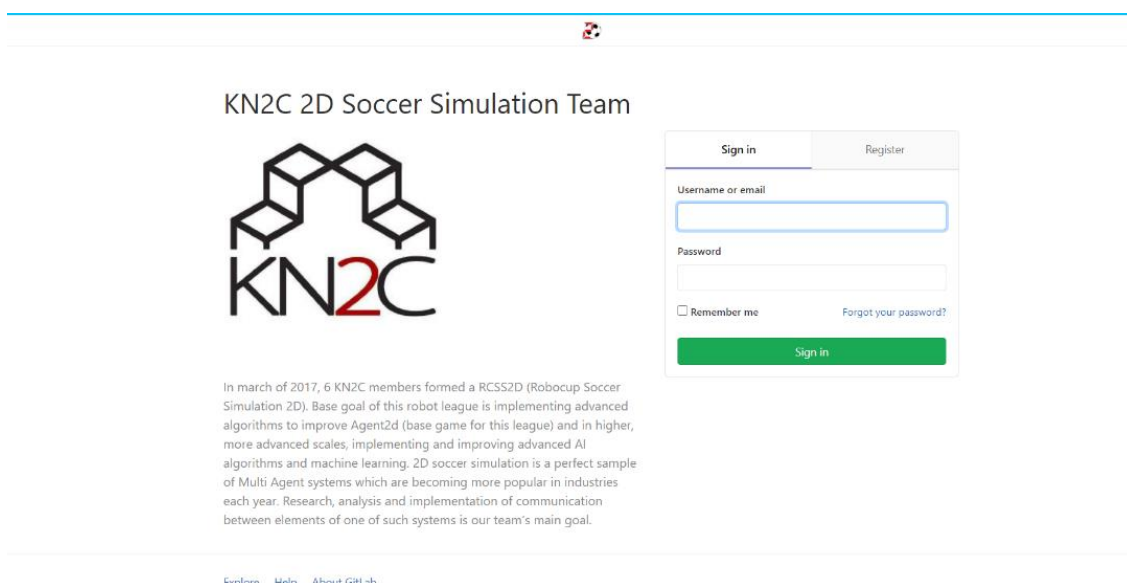
می پردازیم.

۳-۲-۱ نسخه ی محلی گیتلب

یکی از چالش های برنامه نویسان کشورهای مختلف ، تحریم هایی است که توسط برخی کشورها خصوصا در حوزه استفاده از نرم افزارها به این کشورها اعمال می کند . کشور ما نیز از این موضوع مهم مستثنا نبوده و همواره برنامه نویسان داخلی با این مشکل روبه رو بوده اند . علاوه بر آن دسترسی همیشگی و بدون مشکل به امکانات مخازن گیت ، در این پروژه باعث شد که از نسخه ی محلی مخازن گیتلب در یک سرور^۱ محلی استفاده شود تا مشکلات ارتباطی به حداقل مقدار خود برسد . سایت گیتلب ، یک نسخه ی محلی از امکانات خود را در اختیار برنامه نویسانی که نیاز به شخصی سازی خود دارند ارائه کرده است که در این پروژه از این نسخه ی محلی در یک سرور محلی واقع در دانشگاه صنعتی خواجه نصیرالدین طوسی استفاده شده است .

این نسخه دقیقا همان نسخه ای است که در خود سایت از آن استفاده شده است با این تفاوت که

مشکلات تحریم و ارتباط در این نسخه به دلیل محلی بودن آن وجود ندارد. [۱۲]



تصویر ۹- شخصی سازی در نسخه محلی گیتلب آسان تر است

^۱ Server

علاوه بر این موضوع ، شخصی سازی واسط نرم افزای گیتلب در این نسخه امکان پذیر است . می توان اندپوینت^۱ های جدیدی را به این نسخه اضافه کنیم و تا حد ممکن درخواست های خود را بهتر نمایش دهیم.

Gitlab CICD ۲-۲-۳-۳

ابزار یکپارچه سازی مداوم و تحویل مداوم گیتلب امکان درست کردن یک خط لوله مناسب را به کاربران می دهد .

Gitlab Runner ۳-۲-۳-۳

تمامی دستوراتی که در خط لوله اجرا می شوند باید توسط یک یا چند سرور اجرا شود . گیتلب به طور کلی دو نوع اجرا کننده در اختیار قرار می دهد .

۱. سرورهای اشتراکی^۲: برای برخی از کارهای عمومی و توسط خود گیتلب پشتیبانی می شوند و برای اجرای دستوراتی مناسب است که جنبه عمومی دارد . گیتلب برای هر پروژه تعدادی از این سرورهای اشتراکی را در اختیار قرار می دهد .

۲. سرورهای خصوصی^۳: به شما این امکان را می دهد که سرور خصوصی خود را متناسب با کار مورد نظر ثبت کنید . پس از هر تغییر در کد و کامیت شدن آن ، کد به صورت مستقیم در سرور محلی به صورت موقت کش^۴ می شود و پس از اجرا شدن دستورات مورد نظر و انجام آزمون های مورد نیاز حذف می شود .

به دلیل محدودیت هایی که سرورهای اشتراکی خصوصا در بحث شبیه سازی دوبعدی و توان پردازشی آنها ، در این پروژه از سرور خصوصی موجود دانشگاه خواجه نصیرالدین طوسی استفاده شده است .

^۱ Endpoint

^۲ Shared Gitlab Runner

^۳ Specific gitlab Runner

^۴ Cache

۳-۲-۳-۴ سرور خصوصی گیتلب

گیتلب به عنوان یک مخزن ، این امکان را به شما می دهد که بتوانید ارتباط پروژه خود و اجرا شدن دستورات خط لوله را به وسیله یک کلید خصوصی^۱ پشتیبانی کنید . این کلید را خود گیتلب در اختیار شما قرار می دهد . همچنین روش های ارتباطی^۲ زیادی برای ارتباط با سرور خصوصی تحت گیتلب پشتیبانی می شود . شما می توانید دستورات خود را در خط لوله تحت روش های زیر انتقال دهید :

- ssh
- shell
- docker
- kubernetes

در این پروژه برای ارتباط و ارسال اطلاعات از گیتلب به سرور خصوصی از روش Shell executor استفاده شده است . یکی از مزایای مهم استفاده از Shell به عنوان روش ارتباطی ، اجرای آسان تر اسکریپت های bash^۳ بر روی ماشینی است که گیتلب بر روی آن راه اندازی شده است .

^۱ Token

^۲ Gitlab runner executor

^۳ Shell scripts

Group Runners

GitLab Group Runners can execute code for all the projects in this group. They can be managed using the Runners API.

[Disable group Runners](#) for this project

Available group Runners: 1

 c UgVwfZ

2d soccer simulation spec runner (Khosro is here)

#8

 2d

عکس ۱۰ سرورهای خصوصی، قابلیت خدمات به چندین پروژه را دارند

پس از پیکربندی یک سرور خصوصی برای یک پروژه، پس از تغییرات کد توسط یکی از اعضای تیم و ثبت شدن آن، کد مورد نظر در سرور محلی دریافت می‌شود و تمامی دستورات خط‌لوله روی آن اجرا می‌شود.

۳-۲-۵ داکر^۱

در این بخش به توضیح مختصری از داکر و کاربرد آن در خطه لوله‌ی مورد نظر می‌پردازیم.

در حدود سال ۲۰۱۵ ساز و کاری تحت عنوان داکر معرفی شد که به سرعت به محبوبیت بالایی رسید. داکر امکانی را فراهم می‌سازد که نرم‌افزارها به صورت مجزا در محیطی کاملاً بسته، بر روی کرنل^۲ لینوکس راه‌اندازی شود که به این محیط بسته اصطلاحاً کانتینر^۳ می‌گویند. کانتینرها این امکان را برای برنامه‌نویسان و توسعه‌دهندگان نرم‌افزارها فراهم می‌کند تا یک برنامه را با تمام مازول‌ها و کامپوننت‌های

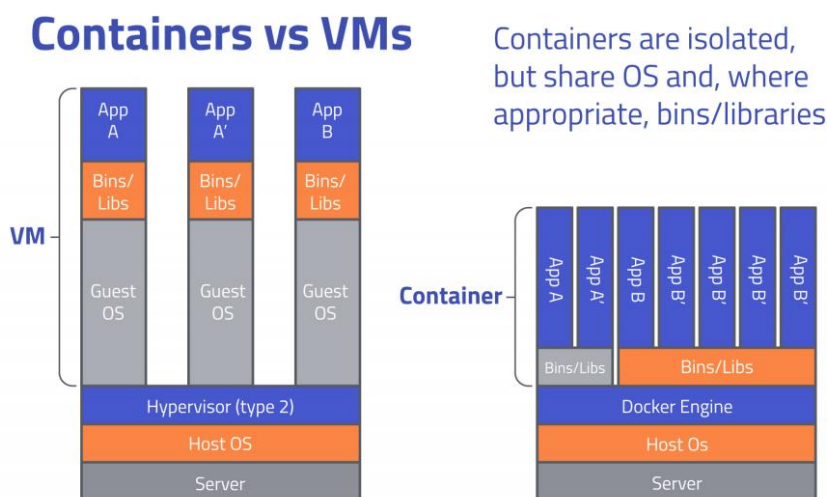
^۱ docker

^۲ kernel

^۳ container

وابسته به آن (مانند کتابخانه ها و توابع) یکی کرده و به صورت یک بسته بندی در اختیار دیگران قرار دهد. یکی از ویژگی های بسیار مناسب داکر ، عدم نگرانی بابت تنظیمات برنامه های داخل یک کانتینر است .

یکی از وظایف داکر ، مدیریت کانتینرها است و تقریباً شباهت خیلی زیادی با یک ماشین مجازی دارد، با این تفاوت که در ماشین مجازی ، باید برای اجرا شدن برنامه ها و دسترسی به کتابخانه ها و توابع به صورت ایزوله و بسته ، ماشین های جدا از هم ایجاد کنیم که این باعث بار پردازشی بیش از حد بر روی سرور می شود ، در حالی که داکر ، نیازی به اجرای بیش از یک ماشین مجازی ندارد . بر روی هر ماشین مجازی جدای از نوع سیستم عامل آن ، می توان ماژول داکر را راه اندازی کرد و سپس با اجرا کردن کانتینترها تحت مدیریت داکر ، از عدم دسترسی برنامه های داخل هر کانتینر به کانتینر دیگر ، اطمینان حاصل کرد . با استفاده از داکر، افزایش حجم بار پردازشی به میزان قابل توجهی کم می شود .

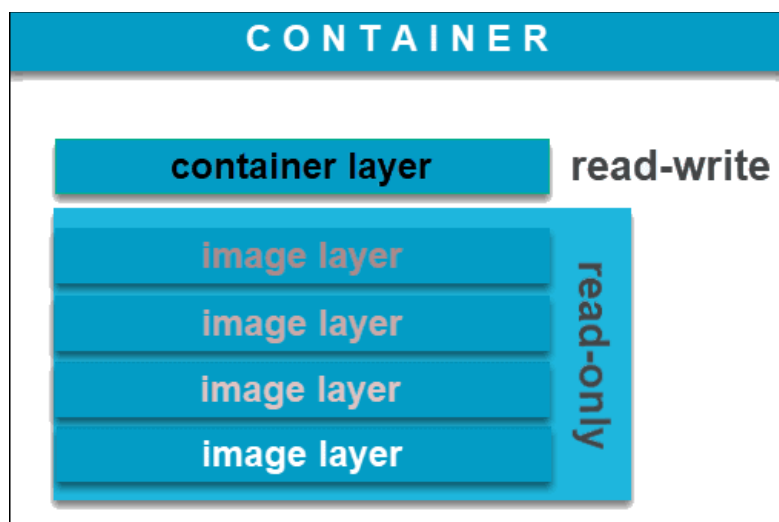


تصویر ۱۱- یکی از مهمترین تفاوت های داکر با ماشین مجازی در میزان بار پردازشی تحمیل شده به سرور است

یکی از چالش های ساخت خط لوله ی مرتبط تحت یکپارچه سازی مداوم ، برای لیگ شبیه ساز دوبعدی فوتبال ، خصوصاً در مراحل بیلد شدن کد و اجرای آزمون های مورد نیاز ، استلزام وجود کتابخانه و وابستگی - هایی است که برای اجرا شدن بازی ها و همچنین اجرای مرحله بیلد شدن کد در مراحل خط لوله لازم است .

برای آن که دید بهتری از داکر و کانتینر داشته باشیم به این مثال توجه کنید . فرض کنیم که ما نیازمندیم که برای سفری طولانی تمامی نرم‌افزارهای خود را به همراه داشته باشیم . برای این کار با استفاده از داکر و ویژگی‌های آن ، میتوان تمام نیازمندی‌ها را داخل یک سبد گذاشت و هر گاه خواستیم بدون آنکه شخص دیگری به این سبد دسترسی داشته باشد از آن استفاده کنیم .

برای ساخت چنین سبدی ما از دستورات مخصوص داکری استفاده میکنیم . و یک داکر فایل^۱ که شامل یک فایل متنی است ، از تمام نیازمندی‌های خود تهیه کرده و پس از آن ، با استفاده از دستورات مخصوص فایل ایمیج^۲ مورد نظر را میسازیم . یک داکر ایمیج در واقع یک فایل است که شامل نیازمندی‌ها و کتابخانه‌های مورد نظر است. این فایل فقط قابلیت خواندن دارد و نمی‌توان آن را تغییر داد. به صورت دقیق‌تر یک کانتینر یک لایه از فایل ایمیجی^۳ است که به حالت اجرا درآمده است که قابلیت نوشتن و خواندن را به کاربر می‌دهد.



تصویر ۱۲- یک کانتینر در واقع یک ایمیج اجرا شده است که میتوان تغییراتی در آن اعمال نمود.

تا به حال مفهوم داکر و قابلیت‌های آن را شرح دادیم و همچنین توضیح دادیم که نیازمندی و چالش

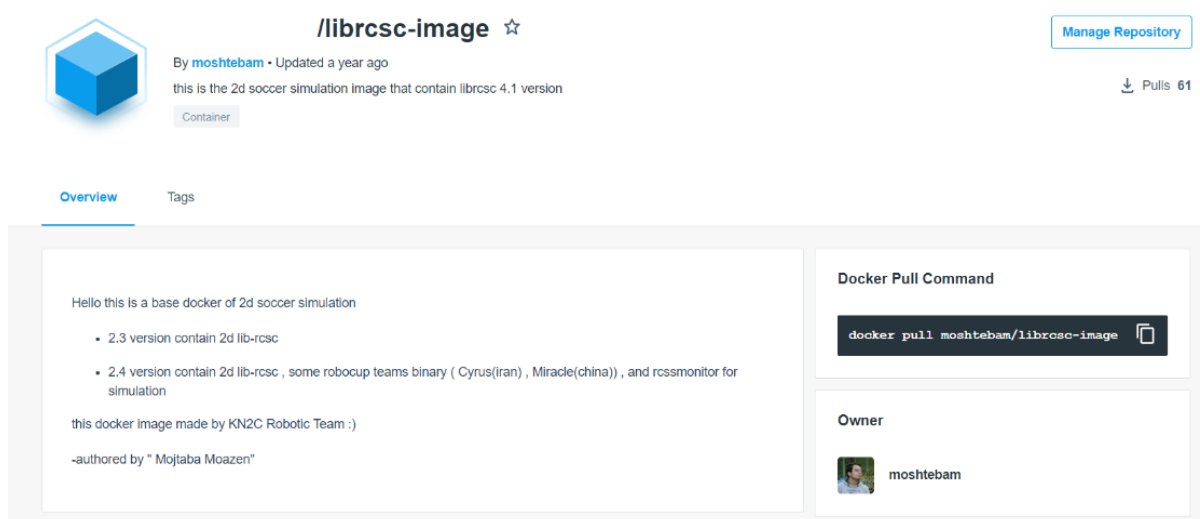
^۱ File

^۲ Docker image

^۳ Image

اصلی در پیاده سازی خط لوله ، کتابخانه ای است که برای اجرا شدن مراحل خط لوله به آن نیاز داریم . راه دیگری برای ارضای این امر وجود دارد و آن فرستادن کل وابستگی ها به همراه هر پروژه به مخزن گیتلب است که این کار باعث افزایش حجم مخزن و طولانی شدن فرآیند خط لوله می شود . برای این پروژه ، یک داکر فایل از تمام وابستگی های مورد نیاز تهیه شد و در نهایت ایمج آن به صورت متن باز در اختیار سایر تیم های شبیه ساز دوبعدی فوتبال قرار گرفت . این داکر ایمج را می توان با استفاده از آدرس و دستور زیر دریافت کرد :

`docker pull moshtebam/librcsc-image`



The screenshot shows the Docker Hub page for the repository `moshtebam/librcsc-image`. It includes a description of the image as a base for 2D soccer simulation, a list of tags (2.3 and 2.4), the Docker pull command, and the owner's name (moshtebam).

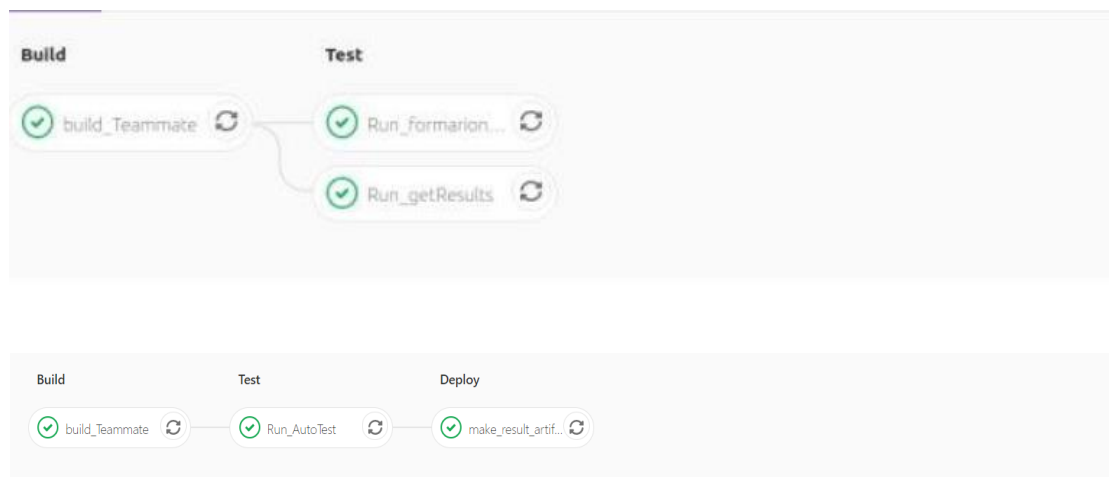
تصویر ۱۳- داکر/ایمج تهیه شده به صورت متن باز در اختیار است

۴-۳ مراحل اجرای خط لوله و راه اندازی آن و دریافت نتایج

در بخش قبلی به موضوع خط لوله پرداختیم و کلیات آن را توضیح دادیم ، در این بخش به تفصیل در مورد پیاده سازی و همچنین هماهنگی بین کامپوننت های ^۱ متفاوت خط لوله می پردازیم. همانطور که اشاره شد برای این پروژه خط لوله از سه بخش اصلی تشکیل شده است :

^۱ Component

- بیلد شدن کد
- اجرای آزمون‌ها
- دریافت خروجی و نتایج کد



تصویر ۱۴- خط‌لوله‌ی این پروژه شامل سه بخش کلی است

روال اجرای خط‌لوله به این شکل است که در صورتی که کاربر تغییراتی را در کد خود اعمال کند پس از آن، خط لوله‌ی مورد نظر برای کدی که تغییرات روی آن اعمال شده‌است اجرا می‌شود.

در گیت‌لب برای پیکربندی خط‌لوله از یک زبان طبقه‌بندی به نام YAML استفاده می‌شود. این زبان وظیفه‌ی انتقال دستورات و پیکربندی خط‌لوله‌ی مورد نظر را بر عهده دارد. برای اجرا شدن دستورات کافی است فایل پیکر بندی با فرمت .yaml در کنار دایرکتوری^۱ جاری پروژه، به همراه کدمورد نظر، کامیت شود، سپس گیت‌لب فایل مورد نظر پیکربندی را شناسایی کرده و اصطلاحاً خط‌لوله^۲ تریگر^۲ می‌خورد یعنی دستورات نوشته شده در فایل اجرا می‌شود. این فایل پیکربندی قواعد و امکاناتی دارد که آنچه در این فایل برای خط‌لوله‌ی مورد نظر پیاده‌سازی شده است در ادامه توضیح داده خواهد شد.

^۱ Directory

^۲ Trigger

۳-۴-۱ فایل YML و توضیحات مربوط به پیکربندی خطلوله

وظیفه‌ی فایل پیکربندی انتقال دستورات به همراه کد مورد نظر به گیتلب است . پس از کامیت شدن این دستورات ، خطوط فایل پیکربندی ، در اجراکننده‌ی خصوصی و یا عمومی که دسترسی‌های مورد نظر را دارد اجرا می‌شود. این فایل پیکربندی به طور کلی از تعدادی مراحل^۱ تشکیل شده است . به طور کلی سه مرحله‌ی قابل تعریف در فایل پیکربندی وجود دارد .

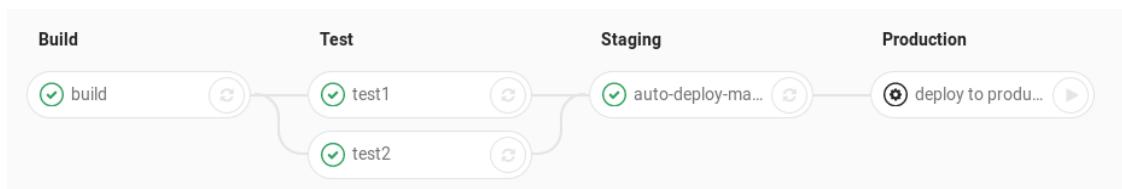
۱. Build stage

۲. Test Stage

۳. Deploy Stage

هر کدام از مراحل بالا شامل مجموعه‌ای از کارهاست^۲، که تمام این کارها به صورت موازی در یک

مرحله انجام می‌شود .



عکس ۱۵ نمونه‌ی یک خطلوله در گیتلب ، هر خطلوله شامل تعدادی مرحله و هر مرحله شامل تعدادی کار است که به صورت پیشفرض به صورت موازی اجرا می‌شود.

سناریوی^۳ مورد نظر برای خطلوله‌ی پیاده‌سازی شده در این پروژه ، دارای سه مرحله و در مرحله‌ی آزمون ، کاربر یکی از آزمون‌های تعریف شده در پروژه را به دلخواه تعیین و اجرا می‌کند . تعیین نوع آزمون برعهده‌ی اعضای تیم و با استفاده از ذکرکردن کلیدواژه‌ی مربوط به آن در متن کامیت است . پس از اجرای آزمون ،

^۱ stage

^۲ Jobs

^۳ senario

نتایج به ابزار مانیتورینگ ارسال می‌شود.

۳-۴-۲ مانیتورینگ نتایج

در این بخش به تفصیل امکانات ابزار مانیتورینگ را شرح می‌دهیم. در بخش‌های قبل گفتیم که طبق سناریوی تعریف‌شده در این پروژه، هریک از اعضا پس از کامیت کردن کد خود، و اجرای خط‌لوله و مراحل آزمون بر روی کد مورد نظر، قادر خواهند بود نتایج آزمون‌ها را مشاهده کنند. برای این کار یک نرم‌افزار تحت وب پیاده‌سازی شده است. در این ابزار مشاهده و بررسی نتایج به صورت مشروح در دسترس قرار گرفته‌است و یکی از نقاط قوت آن، بازخورد مناسب از وضعیت یک تیم شبیه‌سازی دوبعدی فوتبال است. برای پیاده‌سازی این ابزار در سمت کاربر و در سمت سرور از زبان‌ها و فریم‌ورک^۱ های تحت وبی استفاده شده است که در ادامه به چالش‌های پیاده‌سازی این بخش می‌پردازیم:

۳-۴-۲-۱ سمت کاربر

در سمت کاربر برای پیاده‌سازی این ابزار نیاز به طراحی ۵ صفحه بود:

۱. صفحه‌ی ابتدایی و توضیحات ابزار

۲. صفحه‌ی ورود اعضا

۳. صفحه‌ی انتخاب پروژه

۴. صفحه‌ی انتخاب آزمون

۵. صفحه‌ی مشاهده نتایج

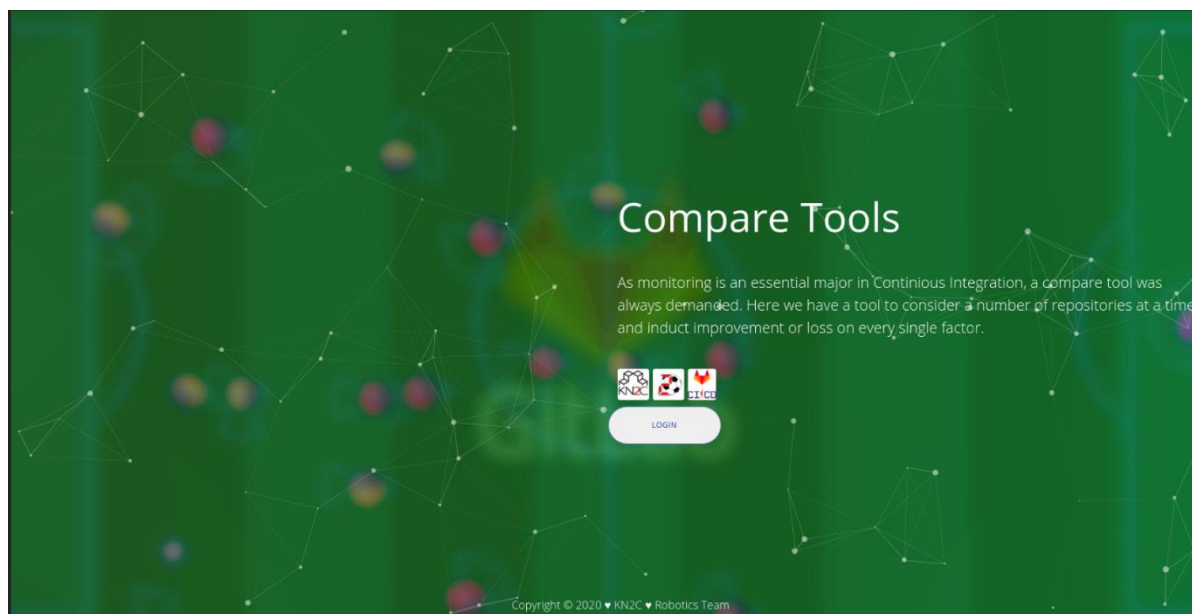
طراحی این صفحات با استفاده از Html و CSS و JavaScript انجام شده است و از کتابخانه‌هایی

نظیر Google charts و JQuery نیز استفاده شده است.

^۱ framework

۱. صفحه اول : توضیحات کلی این ابزار

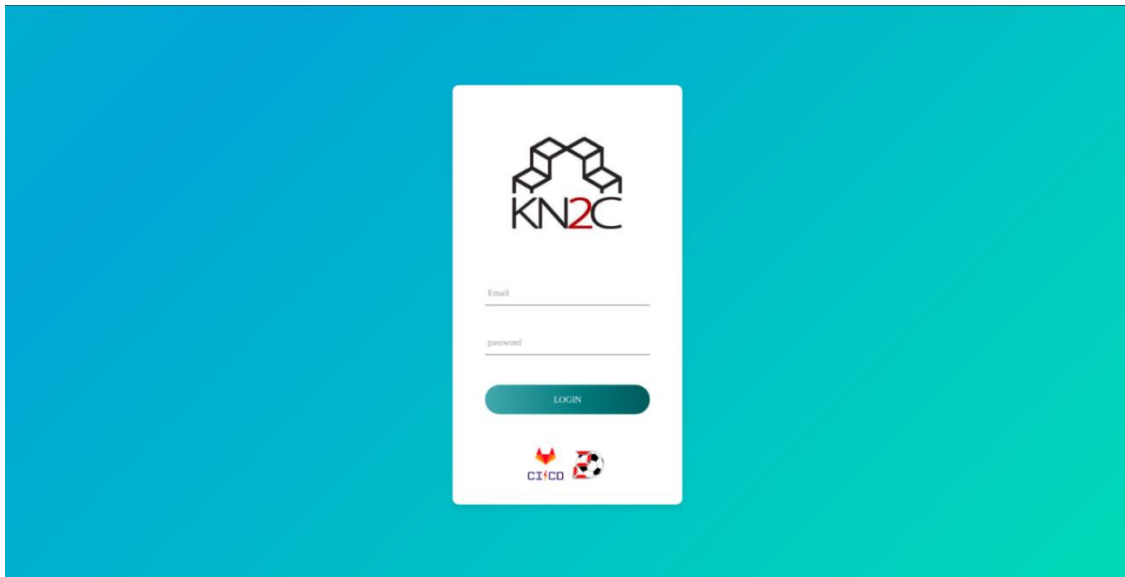
در این صفحه ، توضیحات کلی پروژه شرح داده شده است ، در واقع این صفحه معرفی اجمالی این ابزار است .



تصویر ۱۶- تصویر صفحه‌ی اصلی ابزار مانیتور و مشاهده‌ی نتایج

۲. صفحه دوم : ورود اعضا

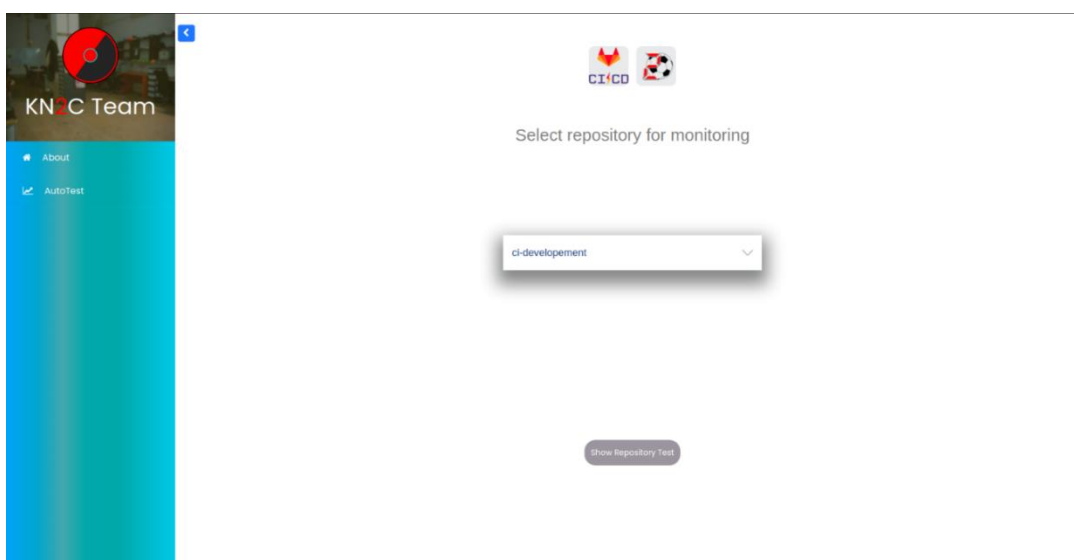
برای آنکه تنها اعضای یک تیم به صورت واحد حق دسترسی به نتایج و مشاهده‌ی آنرا داشته باشند صفحه ورود اعضا طراحی شده است .



تصویر ۱۷- صفحه‌ی ورود اعضای تیم

۳. صفحه سوم : انتخاب پروژه

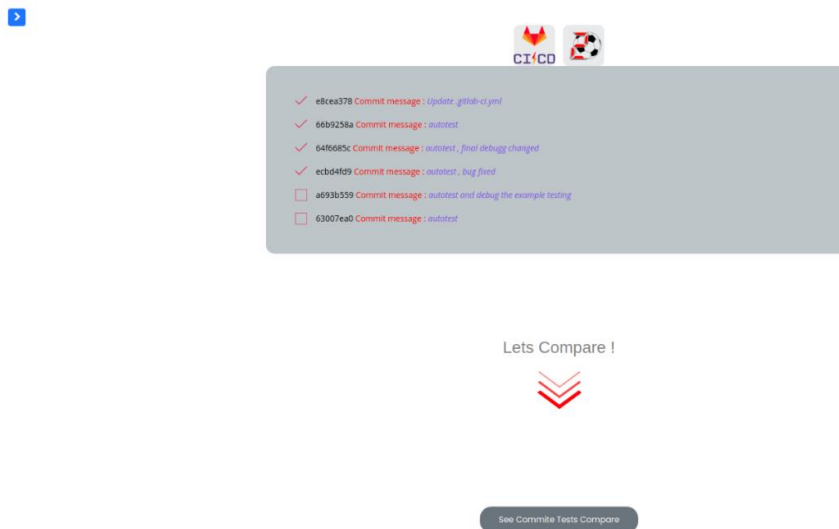
در یک تیم شبیه‌سازی دوبعدی فوتبال ، تعداد زیادی پروژه ممکن است در مخزن گیتلب تیم وجود داشته باشد ، که در هر یک از آن‌ها آزمون‌های مورد نیاز پیاده‌سازی شده است . برای آنکه کاربر تمایز بهتری بین انواع آزمون‌ها در تیم داشته‌باشد طراحی این بخش الزامی بود .



تصویر ۱۸- کاربر با انتخاب هر مخزن به صورت مجزا در مرحله‌ی بعد می‌تواند نتایج را مشاهده کند .

۴. صفحه چهارم : انتخاب آزمون های مورد مقایسه

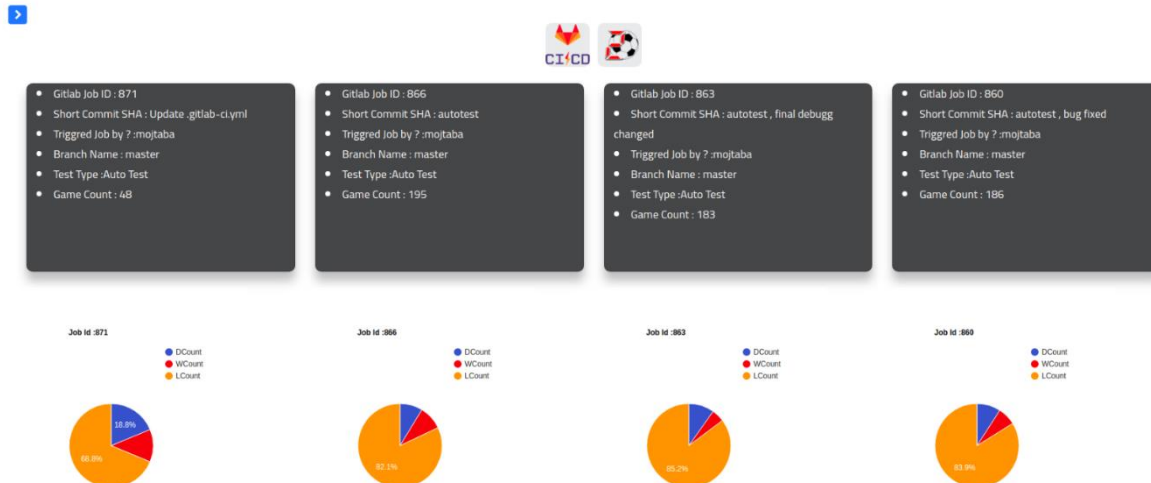
برای آنکه کاربر بتواند پس از انتخاب پروژه ، آزمون های اخیر را مورد مقایسه قرار دهد این بخش طراحی شد . در این قسمت کاربر تمامی آزمون های اجرا شده بر روی پروژه ، به همراه شماره ی کامیت آن ها را مشاهده می کند که می تواند هر تعداد از این آزمون ها را برای مقایسه با یکدیگر انتخاب کند .



تصویر ۱۹- در این قسمت هر کاربر قادر است کلیه ی آزمون های اجرا شده بر روی این پروژه را مشاهده کند . با انتخاب هر کدام از آن ها در مرحله بعد نتایج به صورت تفکیکی نمایش داده می شود.

۵. صفحه پنجم : مشاهده نتایج و مقایسه

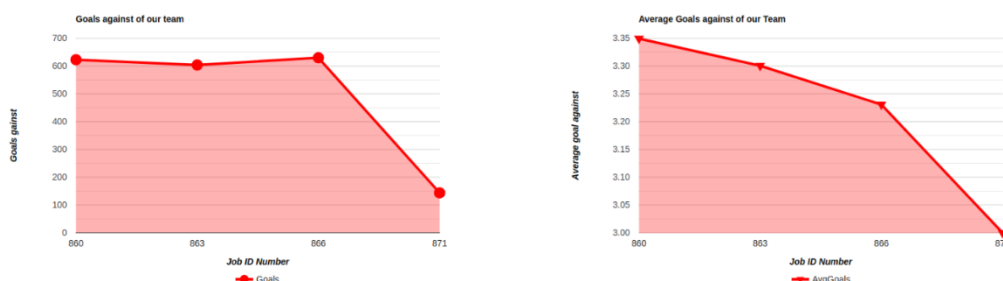
هدف از این بخش ، مشاهده نتایج آزمون های انتخاب شده در صفحه ی قبل است. برای پیاده سازی نمودارهای این بخش از کتابخانه های گوگل استفاده شده است . در این صفحه اطلاعات هر آزمون به صورت جدا ، شامل تعداد بردها و باخت ها و مساوی ها نمایش داده می شود. علاوه بر آن نمودارهای موجود در این صفحه به سه دسته تقسیم می شوند.



تصویر ۲۰ - اطلاعات مجزا برای هر آزمون

۱. بخش اول ، تعداد باخت و مساوی و بردهای تیم در این قسمت مشخص می‌شود. این یک وضعیت کلی از آزمون است که نشان‌دهنده‌ی برآیند وضعیت دفاعی و یا حمله‌ی تیم می‌باشد.
۲. در بخش دوم ، اطلاعات دفاعی آزمون‌ها با یکدیگر مقایسه می‌شود . این اطلاعات شامل تعداد گل‌های خورده و میانگین آن‌هاست.

in this section we can improve with monitoring defense strategy !

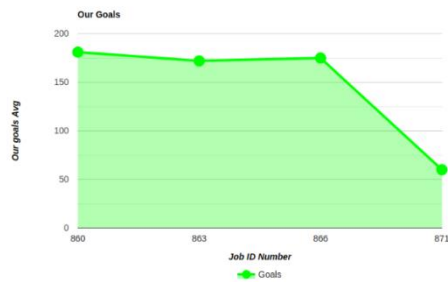
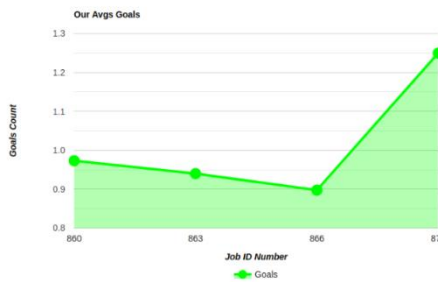


عکس ۲۱ اطلاعات وضعیت دفاعی تیم در آزمون‌های متفاوت

۳. در بخش سوم ، اطلاعات موجود در خط حمله با یکدیگر مقایسه می‌شود. این اطلاعات شامل تعداد گل‌زده به حریف و میانگین آن‌هاست .

offensive strategy 📊

in this section we can improve with monitoring offensive strategy !



تصویر ۲۲- مقایسه‌ی وضعیت حمله در آزمون‌های متفاوت

۳-۲-۴- سمت سرور

وظایف دستورات سمت سرور عبارت است از :

۱. پردازش فرم‌ها شامل فرم‌های ورود اعضا و درخواست اطلاعات مورد نظر برای پروژه‌ها و مخازن
۲. پردازش درخواست‌های کاربر در رابطه با مقایسه‌ی آزمون‌های موردنظر
۳. درخواست و دریافت نتایج از سمت سرور گیتلب و واسط نرم‌افزاری آن

درخواست‌های سمت سرور توسط زبان‌های تحت وب php^۱ و فریم‌ورک محبوب لاراول^۲ پیاده‌سازی

شده است .

۱. زبان سمت سرور php : php یک زبان برنامه‌نویسی است که می‌توان از آن برای پیاده‌سازی صفحات پویا استفاده کرد . این زبان یک زبان سمت سرور است یعنی با استفاده از آن نمی‌توان تاثیر مستقیمی بر روی ظاهر نرم‌افزار گذاشت . با استفاده از آن می‌توان درخواست‌های متفاوت به واسط‌های گوناگون و پایگاه‌داده‌های مورد نظر ارسال کرد .

^۱ PHP Hypertext Preprocessor

^۲ laravel

۲. فریم ورک لاراول : این فریم‌ورک در واقع یکی از فریم‌ورک‌های محبوب زبان php است که برای توسعه نرم‌افزارهای تحت وب استفاده می‌شود. این فریم‌ورک بر پایه‌ی معماری نرم‌افزاری MVC^۱ ساخته شده است.

۳-۲-۴-۳ GraphQL

گراف کیوال یک ساختار جدید است که برای توسعه و درخواست^۲ واسط‌های نرم‌افزاری، مورد استفاده قرار می‌گیرد و جایگزین مناسبی برای برای واسط‌هایی است که با REST توسعه داده شده‌اند.

در این زبان، درخواست‌های کاربر یک نقطه‌ی انتهایی^۳ فرستاده می‌شود. و در این درخواست هر آنچه که مورد نیاز کاربر است بازگردانی می‌شود (برخلاف واسط‌های رست^۴ که در آن چندین نقطه‌ی انتهایی برای درخواست‌ها وجود دارد که پس از درخواست کاربر تمامی اطلاعات بازگردانده می‌شود) همچنین در این زبان، می‌توان خروجی بازگردانی شده از نقطه‌ی انتهایی را به دلخواه تغییر داد و هر آنچه که مورد نیاز است درخواست شود. به این ترتیب گراف کیوال بستری را فراهم کرده‌است که کاربر را از پیچیدگی‌های زیاد دور می‌کند و باعث می‌شود درخواست‌های فرستاده شده به نقطه‌ی انتهایی از نظر بیشتر برخوردار باشند.

```
type Query {
  me: User
}

type User {
  id: ID
  name: String
}
```

تصویر ۲۳- در این زبان درخواست‌های ارسالی، شامل تمام نیازمندی‌های کاربران است.

یکی از ویژگی‌های واسط نرم‌افزاری گیتلب پشتیبانی از این زبان است. استفاده از این ساختار، علاوه

^۱ Model-View-Controller

^۲ Query

^۳ EndPoint

^۴ RestApi

بر ساده سازی درخواست ها ، از انتقال اطلاعات اضافه ، جلوگیری می کند زیرا ، در این ساختار ما هر آنچه که مورد نیاز است را به واسط نرم افزاری درخواست می دهیم و دقیقا مطابق این درخواست را بدون هیچ اطلاعات اضافی دریافت خواهیم کرد . [۱۳]

۴ فصل چهارم : جزئیات پیاده‌سازی خط‌لوله و ابزار مشاهده نتایج

در فصل قبل به بررسی اهداف و کتابخانه ها و ساختارهای استفاده شده در این پروژه پرداختیم و چالش های مربوط به آن را بررسی کردیم . در این فصل قصد داریم جزئیات پیاده سازی را به صورت کامل مورد بررسی قرار دهیم . به طور کلی این پروژه از دو بخش کلی تشکیل شده است :

۱. پیاده سازی خطلوله و پیکربندی آن و همچنین پیاده سازی آزمون های مورد نیاز

۲. پیاده سازی ابزار مانیتورینگ نتایج بازی های شبیه سازی دوبعدی فوتبال

در ادامه این دوبخش به صورت کامل توضیح داده خواهد شد .

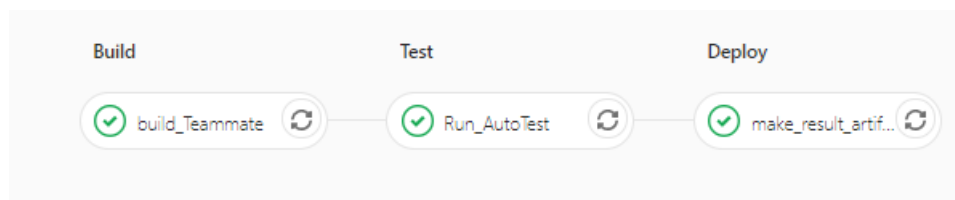
۱-۴ پیاده سازی و پیکربندی خطلوله

خطلوله پیاده سازی شده شامل سه مرحله است :

۱. بیلدشدن کد

۲. اجرای آزمون ها

۳. خروجی نتایج



تصویر ۲۴- خطلوله ی پیاده سازی شده

در ادامه به پیکربندی این خطلوله می پردازیم :

پیکربندی خطلوله در گیتلب در فایل پیکربندی مبتنی بر زبان YAML انجام میشود . در قطعه کد

زیر محتویات این فایل مشخص و توضیح داده شده است :

هر پیکربندی خطلوله ، دارای یک سری تنظیمات و کلمات کلیدی است که در ادامه به بررسی آنها می‌پردازیم .

```
1 image: docker
2
3 stages:
4   - build
5   - test
6   - deploy
7
8 build_Teammate:
9   stage: build
10  variables:
11    GIT_STRATEGY: clone
12    GIT_CHECKOUT: "true"
13  tags:
14    - 2d
15  script:
16    - pwd
17    - ls
18    - cd our
19    - ./configure CXXFLAGS='-std=c++03'
20    - make clean
21    - make -j8
22    - cd ..
23
24 Run_AutoTest:
25   stage: test
26   variables:
27     GIT_STRATEGY: none
28     GIT_CHECKOUT: "false"
29   image: moshtebam/librcsc-image:2.4
30   tags:
31     - 2d
32   script:
33     - ls
34     - pwd
35     - cd /home/kn2c/2DSim/AutoTest2/AutoTest
36     - sudo chmod 777 *
37     - whoami
38     - sudo ./test.sh
39   rules:
40     - if: '$CI_COMMIT_MESSAGE =~ /^autotest.*/'
41       when: always
42
```

تصویر ۲۵- پیکربندی مراحل ساخته‌شدن و آزمون در خطلوله

۴-۱-۱ کلمه‌ی کلیدی Stages

در این قسمت ، مراحل خطلوله تعریف شده است . هر مرحله ^۱ شامل یک یا بیشتر ، کار ^۲ است که به صورت پیش فرض به صورت موازی انجام می شود . سه مرحله ی اصلی در این خطلوله عبارت است از :

- Build
- Test
- Deploy

۴-۱-۲ پیکربندی کار

هر کار زیر مجموعه ی یک مرحله از خطلوله است . برای مثال ، اولین کار زیر مجموعه ی مرحله ی Build شناسایی می شود. هر کار ، از چند قسمت تشکیل شده است ، قسمت اول که با کلمه ی کلیدی Script تعریف شده است دستوراتی است که در این کار انجام می شود . در هر کار تعدادی متغیر ^۳ های محلی وجود دارد که تنظیمات مربوط به آن را در برمی گیرد . شرح این متغیرها که در شکل قبل از آن استفاده شده است در زیر توضیح داده شده است :

۱. Git_Strategy : سه نوع استراتژی برای ^۴ هر کار می توان تعریف کرد :

a. Clone : این استراتژی ، هر بار پس از کامیت شدن کد مورد نظر ، آن را در سرور

محلی به روزرسانی می کند.

b. Fetch : این استراتژی پس از هر بار کامیت شدن کد مورد نظر ، آن را با آخرین

نسخه ی موجود در مخزن ، به روزرسانی می کند .

c. None : این استراتژی ، نتیجه ی محاسبات کار قبلی را حفظ می کند و روی آن ها

عملیات را انجام می دهد .

^۱ Stage

^۲ Job

^۳ Variable

^۴ Strategy

۲. `Git_CheckOut` : این متغیر تعیین می‌کند که آیا تغییرات حاصل شده در مرحله‌ی قبل را

بر شاخه‌ی ^۱حال حاضر اعمال کند یا خیر در صورت درست بودن ^۲این متغیر این تغییرات اعمال می‌شود .

۳. `Image` : در صورتی که این متغیر مقدار بگیرد ، تمام دستورات مربوط به آن کار در یک

کانتینر از ایمجی که آدرس آن داده شده است اجرا می‌شود . لازم به ذکر است کانتینتر مورد نظر تحت سروری که پیکربندی شده است اجرا می‌شود . ابتدا ایمج مورد نظر از آدرسی که به آن داده شده دریافت می‌شود و سپس دستورات در یک کانتینر ، اجرا می‌شود .

۴. `Tag` : هر سرور در یک پروژه یک تگ ^۳دارد . این بخش تعیین می‌کند که این کار با استفاده

از سرورهایی که تگ مورد نظر را دارند اجرا شود یا خیر .

۵. `Script` : در این بخش دستورات مربوط به آن بخش نوشته می‌شود . دستوراتی که در این

بخش نوشته می‌شوند با توجه به راه ارتباطی با سرور مشخص می‌شوند . ما از راه ارتباطی شل ^۴در این پروژه استفاده کرده‌ایم و به همین دلیل دستورات این بخش تحت شل نوشته شده است .

۶. `Rules` : این بخش محدودیت‌های مربوط به پیکربندی را تعیین می‌کند .

a. `If` : بخش اصلی محدودیت‌ها در این قسمت نوشته شده است . ذکر شده است که در

صورتی که پیام کامیت شامل یک الگوی مشخص باشد ، این کار اجرا شود و اگر

شامل این بخش نباشد این کار اجرا نمی‌شود .

b. `When` : تعیین می‌کند شرط ذکر شده در خط قبلی چه زمانی اجرا شود . دو حالت

دارد یا این شرط همیشه اجرا می‌شود یا می‌توان آن را در صورت نقض شدن شروط

^۱ Branch

^۲ True

^۳ Tag

^۴ Shell

اجرا کرد .

۴-۱-۲-۱ تولید خروجی نتایج

در این بخش نتایج حاصل شده در مرحله قبل باید تولید شود . تولید نتایج به این صورت است که پس از اجرای آزمون ها نتایج در خروجی جیسان^۱ ذخیره می شود . این خروجی تحت عنوان آرتیفت^۲ در سرور محلی ذخیره شده و در هر زمانی می توان به تفکیک آزمون به آن ها مراجعه کرد . پس از آن نتایج به تفکیک در دسترسی ابزار مانیتورینگ برای تجزیه و تحلیل آن ها قرار میگیرد . همچنین نتایج به تفکیک در شبکه های اجتماعی تیم شبیه ساز دوبعدی فوتبال نشر داده می شود.

```

43
44 make_result_artifact:
45   stage: deploy
46   variables:
47     GIT_STRATEGY: none
48     GIT_CHECKOUT: "false"
49   image: moshtebam/librcsc-image:2.4
50   tags:
51     - 2d
52   script:
53     - ls
54     - cd /home/kn2c/2DSim/AutoTest2/AutoTest
55     - sudo chmod 777 *
56     - echo $CI_COMMIT_SHORT_SHA
57     - sudo ./result.sh >> /var/lib/gitlab-runner/builds/c_ugVwfZ/0/kn2c-cicd/ci-developement/Result.txt
58   artifacts:
59     paths:
60       - GameData.json
61       - Result.txt
62     expire_in: "1 month"
63
64 deploy_result:
65   stage: deploy
66   variables:
67     GIT_STRATEGY: none
68     GIT_CHECKOUT: "false"
69   image: moshtebam/librcsc-image:2.4
70   tags:
71     - 2d
72   script:
73     - telegram-send --configure-channel < input.txt
74     - cd AutoTest
75     - telegram-send --file Result.txt
76

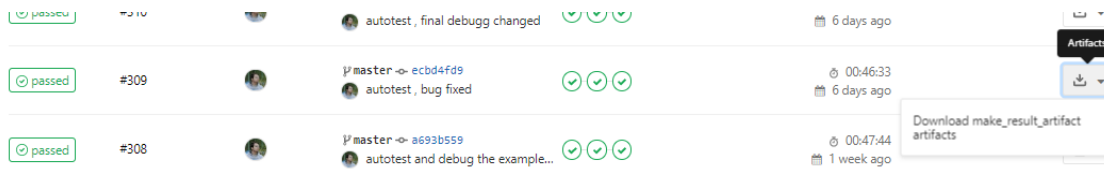
```

تصویر ۲۶- پیکربندی مراحل تولید خروجی

^۱ Json

^۲ artifacts

نتایج تحت عنوان آرتیفت‌ذخیره و نمایش داده می‌شود.



تصویر ۲۷- خروجی نتایج به صورت مجزا برای هر آزمون قابل دانلود است

همچنین خروجی آزمون در یک فایل متنی نیز ذخیره می‌شود که برای نمونه در شکل زیر آورده شده

است :

```
Diff Goals Distribution:
~ -9: 1 [ ] 0.54%
~ -8: 1 [ ] 0.54%
~ -7: 4 [ ] 2.15%
~ -6: 9 [#] 4.84%
~ -5: 11 [#] 5.91%
~ -4: 25 [####] 13.44%
~ -3: 28 [####] 15.05%
~ -2: 44 [#####] 23.66%
~ -1: 33 [#####] 17.74%
~ 0: 17 [###] 9.14%
~ 1: 9 [#] 4.84%
~ 2: 4 [ ] 2.15%
```

```
Game Count: 186 (14 left)
Goals: 181 : 623 (diff: -442)
Points: 56 : 485 (diff: -429)
Avg Goals: 0.97 : 3.35 (diff: -2.38)
Avg Points: 0.30 : 2.61 (diff: -2.31)
Left Team: Win 13, Draw 17, Lost 156
Left Team: WinRate 6.99%, ExpectedWinRate 7.69%
Left Team: 95% Confidence Interval [3.33%, 10.65%]
Left Team: MaxWinRate 13.50%, MinWinRate 6.50%
```

تصویر ۲۸- خروجی نتایج بازی‌ها

۴-۱-۲-۲ ارسال نتایج به شبکه‌های اجتماعی

برای دسترسی بیشتر نتایج آزمون‌های اجرا شده در هر مرحله توسط یک درخواست به یک کانال

^۱تلگرامی که قابل مشاهده برای اعضای تیم است فرستاده می‌شود. برای این امر از رابط‌های نرم‌افزاری ^۲تلگرام

^۱ channel

^۲ Application Programming Interface (API)

استفاده شده است .

```

63
64 deploy_result:
65   stage: deploy
66   variables:
67     GIT_STRATEGY: none
68     GIT_CHECKOUT: "false"
69   image: moshtebam/librsc-image:2.4
70   tags:
71     - 2d
72   script:
73     - telegram-send --configure-channel < input.txt
74     - cd AutoTest
75     - telegram-send --file Result.txt
76

```

تصویر ۲۹ - فرستادن نتایج به کانال تلگرام

پس از ذخیره سازی و اجرای تمامی آزمونهای مورد نظر بر روی کد شبیه سازی دوبعدی فوتبال ، حال به توضیحات بخش دوم پروژه یعنی ابزار مقایسه ی این نتایج می پردازیم .

۲-۴ پیاده سازی آزمون ها

۱-۲-۴ AutoTest

این آزمون به صورت متن باز توسط تیم رایتینگل نوشته شده است . اما برای این پروژه به دلیل خروجی - های متفاوت مورد نیاز ، مورد بازبینی قرار گرفته است .

۲-۲-۴ AutotestFormation

این آزمون ، چینش هایی که در یک دایرکتوری به نام formation قرار گرفته است به عنوان چینش بازیکنان هر بار با استفاده از آزمون Autotest اجرا می کند .


```

27 main() {
28
29     echo "-----!TestStarted!-----"
30
31     cd $FORMATIONS_PATH
32     FORMS=()
33     echo "Formations : "
34     formation_count=0
35
36     for i in $( for i in $(ls -d */); do echo ${i%%/}; done ); do
37         if [ $i != "." ] && [ $i != ".." ] ; then
38             ((formation_count+=1))
39             echo " " $formation_count $i
40             FORMS+="$i "
41         fi
42     done
43
44     cd ../$OPP_PATH
45     TESTTEAMS=()
46     echo "Test Teams : "
47     team_count=0
48
49     for i in $( for i in $(ls -d */); do echo ${i%%/}; done ); do
50         if [ $i != "." ] && [ $i != ".." ] ; then
51             ((team_count+=1))
52             echo " " $team_count $i
53             TESTTEAMS+="$i "
54         fi
55     done
56
57     cd ../$AUTOTEST
58     echo "--Start Test--"
59     i=0
60     j=0
61     for team in $TESTTEAMS;do
62         ((i+=1))
63         for form in $FORMS;do
64             ((j+=1))
65             echo " ----> $form"
66             rm ../$OURCODE/Formations/*
67             cp ../$FORMATIONS_PATH/$form/* ../$OURCODE/Formations/
68             echo " T${i} of ${team_count} | F${j} of ${formation_count}"
69             echo " -----"
70             echo " Form Path - > ${form}"
71             ./test.sh -r $ROUND -p $PROCESS -f "${form}" -q "../TheirCodes/${team}" -w "${team}" -h 2
72             sleep 1
73         done

```

تصویر ۳۰- پیاده سازی آزمون دوم، در این آزمون هر بار چینی مورد نظر جایگزین چینی اصلی می‌شود و آزمون قبلی روی آن اجرا می‌شود.

پس از اجرای بازی‌ها و تمام شدن آن، در نهایت برای خروجی یک تابع در زبان پایتون صدا زده

می‌شود.

```

30 parseall() {
31     local TITLE="N/A"
32     local CACHE_DIR="cache.d"
33     local CACHE_FILE="$CACHE_DIR/cache"
34
35     mkdir $CACHE_DIR 2>/dev/null
36     touch $CACHE_FILE
37     chmod 777 $CACHE_DIR $CACHE_FILE 2>/dev/null
38
39     for i in $RESULT_LIST; do
40         if [ "$TITLE" = "N/A" ]; then
41             TITLE=`cat $i | grep '\<vs\>' | sed -e 's/\t//g'`
42             if [ -z "$TITLE" ]; then
43                 TITLE="N/A"
44             fi
45         fi
46         if [ ! -f $CACHE_DIR/$i ]; then
47             local OUTPUT=`awk -f $PARSE $i`
48             if [ ! -z "$OUTPUT" ]; then
49                 echo "$OUTPUT" >>$CACHE_FILE
50                 touch $CACHE_DIR/$i
51             fi
52         fi
53     done
54
55     echo $TITLE
56     cat $CACHE_FILE
57 }
58
59 parseall | /usr/bin/python2 $PROCESS $* >>$RESULT
60
61 if [ $SPINNER_PID -gt 0 ]; then
62     exec 2>/dev/null
63     kill $SPINNER_PID
64 fi

```

تصویر ۳۱- صدازدن تابع process

تابع process یک تابع برای محاسبه‌ی تمامی خروجی ها در زبان پایتون است که در شکل زیر تبدیل

خروجی به فایل جیسان مشاهده می‌شود .

تصویر ۳۲- تبدیل خروجی به فرمت جیسان

در فصل قبل به این موضوع اشاره کردیم که برای آنکه وابستگی‌ها را در کنار کد منبع داشته باشیم نیاز به وجود تصویر داکری^۱ بود که این وابستگی‌ها در آن موجود باشد. برای این که داکر فایل مورد نظر ساخته شود ابتدا داکر فایل مورد نظر از وابستگی‌ها نوشته شد و پس از آن فایل داکر ایمج از این داکر فایل ساخته شد. در ادامه چگونگی ساختن این فایل ایمج می‌پردازیم:

^ Docker Image

```

1 FROM ubuntu:16.04
2
3 workdir ./agent
4 #install gcc
5
6 #install library
7 COPY librcsc-4.1.0.tar.gz .
8 COPY rcssserver-15.5.0.tar.gz .
9 COPY agent2d-3.1.1.tar.gz .
10 RUN apt-get update \
11     && apt-get update \
12     && apt-get install -y build-essential curl wget g++-4.9 \
13     | tar bison flex libboost-all-dev \
14     && tar xvf librcsc-4.1.0.tar.gz \
15     && tar xvf agent2d-3.1.1.tar.gz \
16     && tar xvf rcssserver-15.5.0.tar.gz \
17     && cd librcsc-4.1.0 \
18     && ./configure \
19     | && make \
20     && make install \
21     && cd .. \
22     && cd rcssserver-15.5.0 \
23     && ./configure \
24     && make \
25     && make install \
26     && cd .. \
27     && cd agent2d-3.1.1 \
28     && ./configure \
29     && make \
30     && make install \
31     && cd ..

```

عکس ۳۳ داکر فایل مورد نظر

۲. پس از تولید این داکر فایل آن را به اسم dockerfile ذخیر می کنیم و پس از آن با استفاده از

دستور زیر تصویر داکری را تولید می کنیم .

Docker build -t rcss-image .

پس از دستور بالا می توان با استفاده از دستور زیر ، اطلاعات تصویر تولید شده را مشاهده کرد :

Docker image ls

۴-۴ پیاده‌سازی ابزار مقایسه نتایج

در بخش قبل نتایج بازی‌ها را به چندین شکل مختلف ذخیره کردیم . مشاهده خروجی آزمون‌ها به صورت مجزا ، مزایا و معایبی دارد . یکی از معایب آن سخت بودن مقایسه‌ی آزمون‌های مختلف است . هدف از پیاده‌سازی این ابزار آسان‌تر کردن این مقایسه‌ها با یکدیگر و مشاهده‌ی روند بهبودی تیم می‌باشد . پیاده‌سازی ابزار مقایسه همانطور که در بخش قبلی نیز اشاره شد شامل دویبخش کلی سمت کاربر و سمت سرور است . که در ادامه آن‌ها را بررسی میکنیم .

۴-۴-۱ سمت کاربر

در این قسمت صفحات مربوط به بخش کاربر و واسط کاربری آن‌ها طراحی شده است . این قسمت شامل چند صفحه است که در فصل قبل به آن اشاره شد .

```

98 @foreach ($All_GameData as $key => $value)
99     @php
100         $commitID=$JobInformation[$key]['ComitTitle'];
101         $Author_Name=$JobInformation[$key]['author_name'];
102         $Branch= $JobInformation[$key]['branch'];
103         $test_Type = 'Auto Test';
104         $GameCount = $value['GameCount'];
105     @endphp
106     <div class="col card card-1">
107         <li class="repointo">Gitlab Job ID : {{ $key }}</li>
108         <li class="repointo">Short Commit SHA : {{ $commitID }}</li>
109         <li class="repointo">Triggred Job by ? :{{ $Author_Name }}</li>
110         <li class="repointo">Branch Name : {{ $Branch }}</li>
111         <li class="repointo">Test Type :{{ $test_Type }}</li>
112         <li class="repointo">Game Count : {{ $GameCount }}</li>
113         {{-- <p class="repointo">{{ $JobID }}</p><br>
114         --}}
115     </div>
116 @endforeach
117 </div>
118 <div class="row">
119     @foreach ($GameWDLInfo as $key => $value)

```

تصویر ۳۴- نمایش اطلاعات دریافتی از آزمون‌های اجراشده

```

137 </div>
138 <div id="defcharts" >
139   <div class="GeneralChart" id="AVGoals" style="width: 900px; height: 500px"></div>
140   <div class="GeneralChart" id="Goals" style="width: 900px; height: 500px"></div>
141 </div>
142
143 <div id="chartstag">
144   <h1 class="headertag">offensive strategy <button id="hideOffensecharts" class="btn btn-primary"> <i class="fa
fa-chevron-circle-down"></i></button></h1>
145   <p id="paraghraphtag">in this section we can improve with monitoring offensive strategy !</p>
146 </div>
147   <div id="offensecharts">
148     <div class="GeneralChart" id="Avggoalzade" style="width: 900px; height: 500px"></div>
149     <div class="GeneralChart" id="Goalzade" style="width: 900px; height: 500px"></div>
150   </div>

```

تصویر ۳۵ - رسم نمودار نتایج

برای رسم نمودارهای مقایسه‌ای نتایج از کتابخانه‌ی مخصوص گوگل استفاده شده که پیاده‌سازی آن را

در تصویر زیر مشاهده می‌کنید .

```

3  function drawOurGoalsChart() {
4      var ChartWinRate = new Array();
5      console.log(goalzade)
6      ChartWinRate[0] = ['JobID', 'Goals'];
7      var i =1 ;
8      $.each(goalzade, function( index, value ) {
9          ChartWinRate[i]=[index,value];
10         i++;
11     });
12     var data = google.visualization.arrayToDataTable(ChartWinRate ,false);
13
14     var options = {
15         title: 'Our  Goals ',
16         colors: [ '#04ff00'],
17         lineWidth :4,
18         pointSize: 16,
19         hAxis: {
20             title: 'Job ID Number' ,
21             titleTextStyle :{
22                 bold :true,
23                 color: 'black'
24             }
25         },
26         vAxis: {
27             title: 'Our goals Avg',
28             titleTextStyle :{
29                 bold :true,
30                 color: 'black'
31             }
32         },
33         curveType: 'function',
34         legend: {
35             position: 'bottom'
36         }
37     };
38
39     var chart = new google.visualization.AreaChart(document.getElementById('Avggoalzade'));
40
41     chart.draw(data, options);
42 }
43

```

تصویر ۳۶ - پیاده‌سازی تابع رسم نمودارهای مقایسه‌ای نتایج

۴-۴-۲ سمت سرور

در سمت سرور پردازش اطلاعات به سه بخش کلی تقسیم می‌شود :

۱. تجزیه و تحلیل فرم‌های ورود و درخواست آزمون‌های هر مخزن
۲. تجزیه و تحلیل درخواست‌های مقایسه‌ی کاربر بر اساس کلید اصلی هر کامیت
۳. ارسال درخواست‌ها به واسط کاربری در گیتلب و دریافت نتایج هر آزمون

```
//
public function login(Request $req){
    $email = $req->input("email");
    $password= $req->input("password");
    $checklogin = DB::table('users')->where(['email' => $email , 'password' => $password])->get();
    if (count($checklogin) >0 ){
        return view('home');
    }
    else{
        echo "fail login";
    }
}
```

تصویر ۳۷- کنترل‌کننده‌ی صفحه ورود/عضا

در بخش تجزیه و تحلیل فرم‌های درخواست کاربر ، نیاز به دریافت نتایج بازی‌ها از واسط نرم‌افزاری گیتلب وجود دارد . برای این بخش از انواع درخواست‌های گراف کیوال و Api های Rest استفاده شده است.

```

91 Route::post('seeResultsofAutotest', function(Request $request){
92     session_start();
93     $JobInformation = [];
94     $i=0;
95     foreach ($SESSION['JobInformation'] as $key => $value) {
96         if($value['status']== 'success' and
97             $value['name']!='make_result_artifact'){
98             $JobInformation[$value['id']]['commitTitle']=$value['commit']['title'];
99             $JobInformation[$value['id']]['author_name']=$value['commit']['author_name'];
100             $JobInformation[$value['id']]['branch']=$value['pipeline']['ref'];
101         }
102     }
103     $projectIdNumber = $_POST['idnumber'];
104     array_pop($_POST);
105     $willCompareCommitsid= [];
106     $index = 0 ;
107     foreach( $_POST as $stuff => $val ) {
108         if ($index !=0) {
109             array_push($willCompareCommitsid , [
110                 $stuff=>$val
111             ]);
112         }
113         $index = $index +1;
114     }
115     $jobId=0;
116     $All_GameData=[];
117     foreach ($willCompareCommitsid as $key => $value) {
118         foreach ($value as $commit => $jobIdnumber) {
119             $jobId=$jobIdnumber;
120         }
121         header('Content-Type: application/json');
122         $endpoint='http://10.10.10.100/api/v4/projects/jobs/artifacts';
123         $endpoint = substr_replace($endpoint, $projectIdNumber ,36 , 0);
124         $endpoint = substr_replace($endpoint, $jobId ,44 , 0);
125         $newstr = $endpoint;
126         $token = '7FaqSolZAZtFzuxFj7Fs';
127         $sch = curl_init($newstr);
128         $zipFile ="Artifacts/".$jobId.".zip";
129         $fp = fopen ("Artifacts/".$jobId.".zip", 'w');
130         $authorization = "Authorization: Bearer " . $token;
131         curl_setopt($sch, CURLOPT_HTTPHEADER, array('Content-Type: application/json' , $authorization ));
132         curl_setopt($sch, CURLOPT_RETURNTRANSFER, true);
133         curl_setopt($sch, CURLOPT_FILE, $fp);
134         curl_setopt($sch, CURLOPT_FOLLOWLOCATION, true);
135         curl_setopt($sch, CURLOPT_HTTPGET, 1);
136         $result = curl_exec($sch);
137         curl_close($sch);
138         $zip = new ZipArchive;
139         $extractPath = "Artifacts/Raw";
140         if($zip->open($zipFile) != "true"){
141             echo "Error :- Unable to open the Zip File";
142         }
143         /* Extract Zip File */
144         $zip->extractTo($extractPath);
145         $zip->close();
146         // json to data array
147         $Specific_GameDatadata = json_decode(file_get_contents('Artifacts/Raw/GameData.json'), true);
148         $All_GameData[$jobId]=$Specific_GameDatadata;
149     }
150     return view('showResults' , compact('All_GameData' , 'JobInformation'));
151 }

```

تصویر ۳۸- دریافت نتایج بازی ها و پردازش آنها برای نمایش به کاربر

پس از دریافت نتایج ، به سمت کاربر فرستاده می شوند تا برای نمایش به کاربر مورد استفاده قرار بگیرند.

همچنین برخی از درخواست ها از طریق GraphQL به شکل زیر فرستاده شده است .


```

29 Route::get('/AutotestResult',function(){
30     $client = \Softonic\GraphQL\ClientBuilder::build('http://10.10.10.100/api/graphql');
31
32     $query = '
33     query {
34         group(fullPath: "kn2c-cicd") {
35             id
36             name
37             projects {
38                 nodes {
39                     name
40                     id
41                 }
42             }
43         }
44     }';
45
46     $variables = [
47         'Authorization: Bearer' => '7FaqSoLzAZtFzuxFj7Fs',
48         'Content-Type' => 'application/json',
49     ];
50     $response = $client->query($query, $variables);
51     $data = json_encode($response->getData());
52     return view('Autotest' , compact('data'));
53 }
54 );

```

تصویر ۳۹- دریافت اطلاعات با استفاده از GraphQL

همانطور که در تصویر بالا مشاهده می‌شود ، درخواست‌ها در این ساختار به شکل دلخواه فرستاده می‌شود و در همان قالب درخواستی ، به درخواست مورد نظر پاسخ داده می‌شود که این از مزایای سرورهای مبتنی بر GraphQL است .

بسته‌های نرم افزار استفاده شده در این قسمت به طور کلی به شرح زیر است :

۱. برای رابط کاربری از بسته‌های JQuery استفاده شده است .
۲. در سمت سرور از بسته‌های نرم‌افزاری Laravel استفاده شده است
۳. برای فرستادن و دریافت اطلاعات از طریف GraphQL از بسته‌ی نرم‌افزاری GraphQL-Relay php استفاده شده است .

۴-۵ سورس کنترل

نسخه ی کاملی از کدبرنامه و راهنمای راه اندازی و استفاده از آن به صورت بسته های سورس کنترل

^۱گیت ^۲در سایت گیتلب ^۳و در مسیر <https://gitlab.com/mr.moazen/rcss۲d-cicd-comparetools>

موجود است .

^۱ Source controll

^۲Git

^۳ Gitlab

۵ - نتیجه گیری و پیشنهادات

با توجه به اهمیت آزمون در بستر شبیه سازی دوبعدی فوتبال ، تیم های فعال در این زمینه با استفاده از این ابزار ، می توانند روند رو به رشد تیم خود را در طی چندین ماه به صورت دقیق مورد تجزیه و تحلیل قرار دهند . همچنین با استفاده از این ابزار اعضای تیم وقت کمتری برای انجام آزمون ها هزینه خواهند کرد و با خودکارسازی این مراحل ، ارتقای نقاط قوت یک تیم شبیه سازی دوبعدی فوتبال ، آسان تر خواهد بود .

لازم به ذکر است برای پیشرفت این پروژه چندین پیشنهاد مطرح می گردد :

۱-۵ پیشنهادات

۱-۱-۵ پیاده سازی آزمون های متنوع دیگر

یکی از پیشنهادات مهم در این زمینه پیاده سازی آزمون های متفاوت دیگری برای بررسی روند تیم ها است . یکی از این آزمون ها آزمونی است که می توان از آن به طور مخصوص در بخش دفاعی استفاده کرد .

- همواره بازیکنان دفاعی یک تیم فوتبال باید نقاطی را برای ایستادن انتخاب کنند که هم به بازیکنان حریف نزدیک باشد و هم در صورت ضدحمله ، بتوانند سریعاً به عقب بازگردند . پیاده سازی این شیوهی جایگیری در زمین باعث افزایش قدرت دفاعی تیم می شود . ولی در صورتی که این پیاده سازی به خوبی انجام نشود می تواند تاثیرات مخربی بر روی نتایج آزمون - های معرفی شده در این پروژه داشته باشد به خصوص می تواند تعداد گل های خورده و میانگین آنرا به شدت افزایش دهد . برای آزمودن این روش جایگیری می توان ساختاری تهیه کرد که به طور مخصوص این قسمت را مورد ارزیابی قرار دهد و با قرار دادن تیم در موقعیت های ضدحمله بتوان آنرا به طور کامل سنجید .

- یکی دیگر از آزمون های مورد نیاز ، پیاده سازی آزمون گرفتن توپ از بازیکنی است که توپ را در اختیار دارد . به این عمل ، بلاک^۱ می گویند . برای این کار می توان به تعداد دفعات متعدد

^۱ Block

توپ را به بازیکن حریف داد و پس از آن با شمارش تعداد موفقیت‌های بازیکن خودی، میزان بهبودی این عمل را سنجید. لازم به ذکر است این ایده نیز به طور خاص بخش دفاعی تیم را مورد سنجش قرار می‌دهد.

- آزمون دیگری برای بخش حمله‌ی تیم می‌توان پیاده‌سازی کرد. یکی از چالش‌های قسمت حمله در یک تیم شبیه‌سازی دوبعدی فوتبال، چالش فرار از بازیکنان حریف و نفوذ به محوطه‌ی جریمه‌ی آنها است. برای سنجش میزان بهبودی این پیاده‌سازی، می‌توان موقعیت‌های مشابه را ساخت و با سنجش تعداد موقعیت‌های خطرناک که منجر به گل می‌شوند این پیاده‌سازی را مورد سنجش قرار داد.

۲-۵ پیاده‌سازی مشاهده‌ی نتایج در ابزار مانیتورینگ نتایج

برای هر یک از آزمون‌های اجرا شده در مرحله قبل می‌توان معیارهای متفاوتی را برای سنجش آن در نظر گرفت که همه‌ی آنها در برنامه‌ی مانیتورینگ به صورت مجزا از هم باید با یکدیگر مقایسه کرد. لازم به ذکر است سنجش مجزای هر آزمون، علاوه بر زمان بر بودن مقایسه‌ی آنها با یکدیگر نمی‌تواند مقایسه‌ی خوبی از ایده‌های پیاده‌سازی آن، به اعضای تیم ارائه کند.

