دانشکده مهندسی کامپیوتر

درستی‌یابی نرم‌افزار با استفاده از تبدیل فرآیند درستی‌یابی به بازی

پایان‌نامه برای دریافت درجه کارشناسی

در رشته مهندسی کامپیوتر گرایش نرم‌افزار

عماد آقاجانی

استاد راهنما:

بهروز مینایی بیدگلی

شهریورماه ۱۳۹۳

دانشکده مهندسی کامپیوتر

بررسی فنون درستی‌یابی نرم‌افزار عمومی با استفاده از تبدیل فرآیند درستی‌یابی به بازی

پایان‌نامه برای دریافت درجه کارشناسی

در رشته مهندسی کامپیوتر گرایش نرم‌افزار

عماد آقاجانی

استاد راهنما:

بهروز مینایی بیدگلی

شهریورماه ۱۳۹۳


بسم الله الرحمن الرحیم

تأییدیه‌ی هیأت داوران جلسه‌ی دفاع از پایان‌نامه/رساله

نام دانشکده:

نام دانشجو:

عنوان پایان‌نامه یا رساله:

تاریخ دفاع:

رشته:

گرایش:

ردیف	سمت	نام و نام خانوادگی	مرتبه دانشگاهی	دانشگاه یا مؤسسه	امضا
۱	استاد راهنما				
۲	استاد راهنما				
۳	استاد مشاور				
٤	استاد مشاور				
٥	استاد مدعو خارجی				
٦	استاد مدعو خارجی				
۷	استاد مدعو داخلی				
۸	استاد مدعو داخلی				

تأییدیه‌ی صحت و اصالت نتایج

باسمه‌تعالی

اینجانب عماد آقاجانی به شماره دانشجویی ۸۸۵۲۱۳٤٤ دانشجوی رشته مهندسی کامپیوتر مقطع تحصیلی کارشناسی تأیید می‌نمایم که کلیه‌ی نتایج این پایان‌نامه/رساله حاصل کار اینجانب و بدون هرگونه دخل و تصرف است و موارد نسخه‌برداری‌شده از آثار دیگران را با ذکر کامل مشخصات منبع ذکر کرده‌ام. در صورت اثبات خلاف مندرجات فوق، به تشخیص دانشگاه مطابق با ضوابط و مقررات حاکم (قانون حمایت از حقوق مؤلفان و مصنفان و قانون ترجمه و تکثیر کتب و نشریات و آثار صوتی، ضوابط و مقررات آموزشی، پژوهشی و انضباطی ...) با اینجانب رفتار خواهد شد و حق هرگونه اعتراض در خصوص احقاق حقوق مکتسب و تشخیص و تعیین تخلف و مجازات را از خویش سلب می‌نمایم. در ضمن، مسؤولیت هرگونه پاسخگویی به اشخاص اعم از حقیقی و حقوقی و مراجع ذی‌صلاح (اعم از اداری و قضایی) به عهده‌ی اینجانب خواهد بود و دانشگاه هیچ‌گونه مسؤولیتی در این خصوص نخواهد داشت.

نام و نام خانوادگی:

امضا و تاریخ:

مجوز بهره‌برداری از پایان‌نامه

بهره‌برداری از این پایان‌نامه در چهارچوب مقررات کتابخانه و با توجه به محدودیتی که توسط استاد راهنما به شرح زیر تعیین می‌شود، بلامانع است:

☐ بهره‌برداری از این پایان‌نامه/ رساله برای همگان بلامانع است.

☐ بهره‌برداری از این پایان‌نامه/ رساله با اخذ مجوز از استاد راهنما، بلامانع است.

☐ بهره‌برداری از این پایان‌نامه/ رساله تا تاریخ ..... ممنوع است.

نام استاد یا اساتید راهنما:

تاریخ:

امضا:

تقدیم به:

پدر و مادر همیشه مهربانم ...

تشکر و قدردانی:

نخستین سپاس و ستایش از آن خداوندی است که بنده کوچکش را در دریای بیکران اندیشه، قطره‌ای ساخت تا وسعت آن را از دریچه اندیشه‌های ناب آموزگارانی بزرگ به تماشا نشیند. لذا اکنون‌که در سایه‌سار بنده‌نوازی‌هایش پایان‌نامه حاضر به انجام رسیده است، بر خود لازم میدانم تا مراتب سپاس را از بزرگوارانی به‌جا آورم که اگر دست یاریگرشان نبود، هرگز این پایان‌نامه به انجام نمی‌رسید.

ابتدا از استاد گرانقدرم جناب آقای دکتر مینایی که زحمت راهنمایی این پایان‌نامه را بر عهده داشتند، کمال سپاس رادارم.

سپس از مهندس آشتیانی که دلسوزانه بنده را در تمامی مراحل پژوهش و نگارش این پایان‌نامه یاری نمودند.

و سپاس ویژه به مهربان‌ترین همراهان زندگی‌ام، به پدر، مادر و برادر عزیزم تقدیم می‌کنم که حضورشان در فضای زندگی‌ام مصداق بی‌ریای سخاوت بوده است.

چکیده

فرایند دیجیتالی شـدن ابزارهای پیرامون و نیاز به سـیسـتم‌های نرم‌افزاری مختلف در آن‌ها، سـرعت توسعه نرم‌افزارها در دهه اخیر به شـکل چشم‌گیری افزایش داده است. به دنبال این افزایش، نرم‌افزارهای گوناگونی در بطن زندگی ما وارد گشـته اسـت که انتظارات از عملکرد این نرم‌افزارها را افزایش داده اسـت. همچنین ورود نرم‌افزارها به حوزه سـیسـتم‌های حیاتی و حسـاس مانند سـیستم‌های کنترل پرواز نیاز به قابل‌اطمینان بودن این نرم‌افزارها را بسیار افزایش داده است.

درستی‌یابی نرم‌افزار تنها راه بررسی صحت عملکرد یک قطعه نرم‌افزاری از خطاهای[۱] مختلف است. ازاین‌رو در دهـه‌های اخیر روش‌های گوناگونی بدین منظور ایجاد گشـــته اســـت که در تمام این روش‌ها، علیرغم تلاش‌های انجام‌شـده، نیازمند حضور مهندسین نرم‌افزار است که در پروژه‌های بزرگ و حساس، این نیروی کار هزینه بالایی دربر دارد.

ما در این پروژه قصـد داریم تا هزینه درستی‌یابی نرم‌افزار را با کاهش میزان دانش لازم برای انجام این کار و درنتیجه آن، افزایش گسـتره نیروی کار ممکن برای آن، کاهش دهیم. این پژوهش شـیوه‌ای برای نگاشـت فضـای کد برنامه‌نویسی و فرایندهای اشکال‌زدایی[۲] در آن به یک جورچین بصری (بازی)، به شکل خودکار، ارائه خواهد داد. ازآنجاکه انجام یک بازی نیاز به دانش خاصی نداشته، می‌توان فرایند درستی‌یابی را با کمک میلیون‌ها کاربر با هزینه پایین انجام داد.

واژه‌های کلیدی: جمع‌سپاری[۳]، درستی‌یابی[٤]، بازی، سیستم نوع[٥]

---

۱ Error

2 Debugging

۳ Crowdsourcing

٤ Verification

٥ Type system

فهرست مطالب

فهرست اشکال

فهرست جداول

فصل ۱:

مقدمه

۱-۱- مقدمه

هدف از آزمون و درستی‌یابی یک قطعه کد بررسی درستی و رفع خطاهای احتمالی در آن است. درگذشته آزمون‌های نرم‌افزاری تنها در پروژه‌های بزرگ و حساس[۱] و گران‌قیمت، مانند نرم‌افزارهای بخش کنترل پرواز یا سیستم‌های نرم‌افزاری موجود در یک فضاپیما، موردتوجه قرار می‌گرفتند. ولی اکنون آزمون به یکی از مراحل ثابت در چرخه حیاط تولید نرم‌افزارها مبدل گشته و در این حوزه فعالیت‌های گسترده‌ای صورت گرفته است.

در یک دسته‌بندی کلی، آزمون‌های نرم‌افزاری به دو گونه اصلی ایستا[۲] و پویا[۳] قابل تقسیم‌بندی است[1]. در شیوه ایستا درستی کد به کمک تحلیل[۴] قطعه کد، بدون نیاز به اجرای آن، صورت می‌گیرد. ولی شیوه پویا بر پایه اجرای مکرر برنامه به ازای ورودی‌های مختلف انجام می‌گیرد. در دنیای امروز استفاده از هردو شیوه مرسوم است.

این پژوهش قصد دارد تا تکنیکی برای انجام آزمون‌های ایستا به شیوه‌ای نوین ارائه دهد که در آن فرآیند بررسی کد در یک دامنه[۵] متفاوت از فضای مهندسی نرم‌افزار انجام بپذیرد. این ایده باعث می‌شود تا نیاز به متخصصین حوزه آزمون نرم‌افزار کم شود و هزینه‌های حاصل ازاین‌دست فرآیندها کاهش پیدا کند. این کاهش هزینه به رواج فرآیند آزمون در حوزه مهندسی نرم‌افزار نیز کمک خواهد کرد.

تمرکز اصلی این پژوهش بر درستی‌یابی ایستا و در محدوده سیستم‌های نوع موجود در زبان‌های برنامه‌نویسی خواهد بود.

۱-۲- انگیزه‌های پژوهش

ازآنجاکه آزمون‌های نوع پویا به اجرای قطعه کد مورد آزمون نیاز دارند، نیازمند آگاهی از مسیرهای اجرایی موجود در کد خواهند بود. نوشتن ورودی‌های متفاوت برای آزمودن تمام این مسیرها مشروط به‌صرف هزینه زمانی و فنی بسیار بالایی است. همچنین عدم قطعیت در اعلام درستی کد به کمک آزمون‌های پویا، این دسته از آزمون‌ها را به مسئله‌ای جذاب در حوزه مهندسی نرم‌افزار مبدل ساخته است. هرچند پژوهشگران

---

۱ Critical

۲ Static

۳ Dynamic

۴ Analyse

۵ Domain

زیادی در حال کار، بر روی این موضوع هستند و در چند سال اخیر نیز کارهای ارزشمندی در این حوزه صورت گرفته است، ولی همچنان نتایج ارائه‌شده نشان می‌دهد که آزمون‌های پویا همچنان توانایی ارائه اطمینان لازم از درستی یک قطعه کد را ندارند.

این در حالی است که قدرت اثبات درستی در روش‌های ایستا، این شیوه را به یکی از حائز اهمیت‌ترین مراحل توسعه نرم‌افزارهای حساس و نظامی مبدل ساخته است. مشکل اصلی در شیوه ایستا هزینه بالای انجام آن به دلیل نیاز به نیروی کار متخصص در این حوزه است. هرچند سعی در به‌کارگیری از هوش مصنوعی برای درستی‌یابی به شیوه ایستا این مشکل را تا حدودی در سالیان اخیر حل کرده است، ولی خطای بالای کامپیوترها همچنان این حوزه را برای ارائه ایده‌های بدیع و کارا باز نگه‌داشته است.

همچنین سرعت رشد تولید نرم‌افزار در سالیان اخیر افزایش چشم‌گیری داشته است و آموزش نیروهای متخصص، متناسب با این سرعت رشد، کاری بسیار دشوار و تا حدودی ناممکن خواهد بود. در ضمن شیوه‌های آزمون ایستای یک نرم‌افزار نیازمند دانش و تخصص بسیار بالاتری نسبت به شیوه‌های پویا است که درنتیجه نیازمند هزینه‌های بالاتر و دوره‌های زمانی طولانی‌تری برای آموزش است.

۱-۳- دستاوردها

انسان در بسیاری از حوزه‌ها از قبیل تشخیص الگوها[1]، مسائل بصری[2] از ماشین‌ها برتری دارند. در شیوه ارائه‌شده در این پژوهش جمع‌سپاری و ارائه یک بازی باقابلیت یادگیری سریع برای اقشار معمولی جامعه، به‌عنوان کلید استفاده‌شده است. در شیوه ارائه‌شده شیوه ایستا سنتی به فضای یک بازی نگاشت می‌گردد. نکته حائز اهمیت در استفاده از فضای بازی، جذب راحت کاربران است و شاید معرفی یک ابزار نمی‌توانست به میزان یک بازی مورد استقبال عموم مردم (که در این پژوهش ارزشمند است) قرار بگیرد و نیازمند صرف یک هزینه جدا برای ایجاد انگیزه در کاربران برای استفاده از آن ابزار می‌گشت.

همچنین وجود مؤلفه «رقابت» که در بازی‌ها وجود دارد می‌تواند به استفاده و رواج هرچه بیشتر آن در راستای جمع‌سپاری کمک کند. علت تأکید بر تعداد بالای کاربران استفاده‌کننده در نحوه و نتایج حاصل از این پژوهش تأثیرگذار بوده و مؤلفه‌ای ضروری است.

همچنین این پژوهش قصد دارد تا بتواند شیوه‌ای ارائه دهد که در آن‌ها نه‌تنها خطای موجود در روش‌های به‌کارگیری هوش مصنوعی نباشد، بلکه نیاز به نیروی متخصص را به حداقل برساند و در همه‌گیر شدن

[1] Pattern Recognition

[2] Visual Problems

شیوه‌های ایستا، حتی در پروژه‌های سطح متوسط، کمک کند.

همچنین یکی از نکـات جانبی رعایت شـــده در این پژوهش در بخش پیاده‌ســـازی آن اســـت. در بحث پیاده‌ســازی، پژوهش جاری به‌گونه‌ای انجام‌شـده اسـت که امکان جمع‌سپاری در آن به‌واسطه امکان خروجی گرفتن سیستم نرم‌افزاری ارائه‌شده در بسیاری از پلتفرم‌ها موجود (چه کامپیوترهای خانگی و چه دستگاه‌های قابل‌حمل) به‌راحتی امکان‌پذیر است.

فصل ۲:

مبانی و مفاهیم ابتدایی

۲-۱- مقدمه

در این فصل تلاش خواهد شد تا اصطلاحات و مفاهیم بکار گرفته‌شده در ادامه‌ی این گزارش به زبان ساده توضیح داده شود. در این بخش در ابتدا سیستم‌های نوع[1] تعریف خواهند شد. سیستم‌های نوع در قسمت‌های مختلف این پژوهش بکار گرفته‌شده است و در ادامه به آن نیاز خواهد شد. سپس مقدمه‌ای درستی‌یابی و آزمون‌های نرم‌افزار بیان خواهد شد تا تفاوت‌های این دو مفهوم تا حد ممکن آشکار گردد. در پایان نیز تعریفی از جمع‌سپاری ارائه داده خواهد شد.

۲-۲- سیستم‌های نوع

سیستم‌های نوع در تعریف رسمی به مجموعه‌ای قوانین اطلاق می‌گردد که به کمک آن‌ها می‌توان یک ویژگی[2] را که نوع می‌نامیم را به انواع ساختارهای موجود در یک‌زبان برنامه‌نویسی از قبیل متغیر، تابع، عبارت و غیره انتساب داد[2].

هدف اصلی از طراحی سیستم‌های نوع کاهش خطاها[3] در برنامه‌های کامپیوتری است. نحوه کار سیستم‌های نوع به این شکل است که قسمت‌های مختلف برنامه باواسطه‌ای به یکدیگر مرتبط می‌گردد و سپس در صورت لزوم ارتباط درست ارتباط‌های موجود در برنامه بررسی می‌گردد. این بررسی می‌تواند به‌صورت ایستا و در زمان ترجمه[4] صورت بگیرد و یا به‌صورت پویا، یعنی در زمان اجرا و یا ترکیبی از هردو نوع ایستا و پویا. از دیگر اهداف سیستم‌های نوع ایجاد امکان بهینه‌سازی کد بر اساس ارتباط‌های تعریف‌شده در برنامه است.

به‌عنوان یک مثال ساده از سیستم‌های نوع می‌توان از زبان C استفاده نمود. در یک برنامه قطعاتی به‌عنوان تابع ارائه می‌شود. یک تابع می‌تواند توابع دیگر را فراخوانی کند. هر تابع نام و پارامترهای دریافتی خود را معرفی می‌کند. برنامه از این نام می‌تواند در فراخوانی آن استفاده کند. این نام بعدتر می‌تواند به‌عنوان وسیله‌ای برای بررسی صحت کد استفاده شود و فراخوانی بانام‌های غیر معتبر را از فراخوانی‌های معتبر متمایز کند.

---

[1] Type Systems

[2] property

[3] Bugs

[4] Compile

یک مثال از سیستم‌های نوع، نوع داده‌ای[1] است. این مثال از رایج‌ترین سیستم‌های نوع بین زبان‌های برنامه‌نویسی است. در این سیستم نوع، برنامه‌نویس به هر یک از متغیرهای برنامه یک نوع داده‌ای اختصاص می‌دهد. در بعضی زبان‌های پیشرفته می‌توان انواع داده‌ای جدید تعریف نمود.

از این نوع‌های داده‌ای در زمان ترجمه برای بررسی صحت انتسابات و یا صحت پارامترهای ارسالی در توابع استفاده می‌گردد. همچنین در زمان اجرا نیز می‌توان از نوع‌های داده‌ای برای بررسی جنس محتوای یک متغیر استفاده نمود.

همچنین در بخش در ۴-۴-۳ توضیحات تکمیلی در رابطه با سیستم‌های نوع و ارتباطشان با پژوهش جاری ارائه گشته است.

۲-۳- آزمون نرم‌افزار

آزمون نرم‌افزار به‌صورت پویا (جعبه سیاه) و ایستا (جعبه سفید) انجام می‌شود .در آزمون پویا، آزمون بر روی کد اجرائی برنامه انجام می‌شود و در آزمون ایستا، کد مبدأ[2] در اختیار است و آزمون بر روی آن انجام می‌شود[3]. به زبان دیگر، در شیوه ایستا درستی کد به کمک تحلیل[3] کد، بدون نیاز به اجرای آن، صورت می‌گیرد ولی شیوه پویا بر پایه اجرای مکرر برنامه به ازای ورودی‌های مختلف انجام می‌گیرد. البته می‌توان تقسیم‌بندی‌های دیگری نیز برای آزمون ارائه داد.

در یک تقسیم‌بندی متفاوت، آزمون نرم‌افزار در چهار سطح مختلف صورت می‌گیرد که این چهار مرحله به‌صورت ترتیبی انجام می‌پذیرند و عبارت‌اند از:

۱. آزمون واحد[4]
۲. آزمون مجتمع سازی[5]
۳. آزمون سیستم[6]

---

[1] Data Type

[2] Source Code

[3] Analyse

[4] Unit Testing

[5] Integration Testing

[6] System Testing

۴. آزمون پذیرش[1]

۲-۳-۱- آزمون واحد

یک واحد کوچک‌ترین قسمت قابل آزمون یک نرم‌افزار است. این واحد در برنامه‌نویسی شیءگرا می‌تواند یک متد باشد و در برنامه‌نویسی رویه‌ای می‌تواند کل برنامه (در زبانی مانند کوبول[2]) یا یک تابع و غیره باشد. هدف در این سطح از آزمون این است که آیا واحد موردنظر به‌تنهایی کاری را که باید انجام بدهد می‌دهد یا خیر.

۲-۳-۲- آزمون مجتمع سازی

آزمون واحد را برای هرکدام از واحدها به‌صورت جداگانه انجام دادید و از صحت عملکرد آن‌ها مطمئن شدید. همه واحدها به‌صورت منفرد به‌طور صحیح وظایف خود را انجام می‌دهند، آیا نیازی به آزمون اینکه وقتی واحدها کنار هم قرار گرفتند و ارتباط برقرار کردند وظایفشان را به شکل صحیح انجام می‌دهند هست یا نیست. فرضی کنید دو نفر مشغول کاری هستند هنگامی‌که موارد موردنیاز برای انجام کار به‌طور کامل مهیا باشد هرکدام از آن دو فرد می‌توانند کارشان را به شکل کامل انجام بدهند؛ اما اگر موارد موردنیاز برای یکی از آن‌ها توسط دیگری تأمین شود ممکن است موارد تهیه‌شده دقیقاً چیزی نباشد که فرد دوم نیاز دارد، یا زمانی زیاد برای تحویل آن موارد موردنیاز باشد که عملکرد فرد دوم را با مشکل روبرو کند. پس ما نیاز داریم تا مطمئن شویم که آیا واحدها در کنار هم کار می‌کنند، به‌درستی فراخوانی می‌شوند و داده‌های درستی را در زمان مناسبی از طریق واسط‌های آن‌ها عبور می‌دهند. آزمون مجتمع سازی یکی از مهم‌ترین و شاید مهم‌ترین سطح از آزمون باشد، بخصوص زمانی که سیستم تغییرات زیادی دارد بعد از انجام تغییرات هرگز این مرحله را نباید فراموش کرد.

۲-۳-۳- آزمون سیستم

فرض کنید آزمون مجتمع سازی را برای یک نرم‌افزار موردنظر انجام دادید و از این مطمئن شدید که تمام قطعات در کنار هم می‌توانند قرار بگیرند و بدون هیچ مشکلی وظایفشان را انجام دهند. قطعات در کنار هم مجتمع شده‌اند و پیکره اصلی نرم‌افزار تشکیل شد، ولی نرم‌افزار خود جزئی از یک سیستم است و نیاز است که با عناصر دیگر این سیستم مانند سخت‌افزارها ارتباط برقرار کند و با آن‌ها مجتمع شود. درنتیجه نیاز

[1] Acceptance Testing

[2] Cobol

داریم تا مطمئن شـویم که سیستم به‌عنوان یک واحد به‌طور کامل عمل خواهد کرد و نیازمندی‌های سیستم را برآورده می‌کند. این سطح از آزمون آخرین سطحی است که توسط توسعه‌دهندگان صورت می‌گیرد تا قبل از تحویل نرم‌افزار به کاربر نهایی برای آزمون از عملکرد آن مطمئن شـویم. برای نمونه دو مورد زیر در آزمون سیستم موردبررسی قرار می‌گیرند:

- **آزمون امنیت[1]:**

فرضـی کنید سیستم باید اطلاعات حساس و حیاتی را پردازش و مدیریت کند و افرادی هستند که به دنبال دسـترسـی غیرمجاز به این اطلاعات و سـوءاسـتفاده از آن هستند. برای اطمینان از عملکرد سیستم در برابر نفوذ گران ما باید تکنیک امنیتی ایجادشـده در سیستم را بررسی کنیم تا مطمئن شویم که سیستم می‌تواند نفوذهای غیرقانونی را تشخیص دهد و در برابر آن‌ها عکس‌العمل نشان دهد.

- **آزمون بازیابی[2]:**

در این نوع آزمایش باعث ایجاد مشکل و ازکارافتادن سیستم به روش‌های مختلف می‌شویم و بررسی می‌کنیم که آیا سیستم می‌تواند خود را به‌طور خودکار بازیابی کند و به فعالیت خود ادامه دهد.

۲-۳-۴- آزمون پذیرش

نرم‌افزار به‌طور کامل توسط توسعه‌دهندگان در تمام سطوح آزمون، با موفقیت آزمون شد، اما آیا نرم‌افزار واقعاً بـه‌طور کـامـل (آن‌گونه که کاربر نهایی می‌خواهد) کار می‌کند. آیا تمام نیازهای فعلی کاربر نهایی را برآورده می‌کند. پس ما به آزمایشـی نیاز داریم که توسـط کاربران نهایی، مشـتریان و نه توسـعه‌دهندگان صـورت می‌گیرد و هدف آن اسـت که کاربر مشـخص کند عملیاتی که برنامه انجام می‌دهد نیازمندی‌های آن‌ها را برآورده می‌کند یا نه. آزمون پذیرش دارای انواع مختلفی است که می‌توان به موارد زیر اشاره کرد:

- **آزمون آلفا:**

آزمون آلفا در سـایت توسـعه‌دهنده نرم‌افزار و در اغلب موارد توسـط کارمندان داخلی و در بعضـی از موارد توسط مشتری انجام می‌گیرد.

- **آزمون بتا:**

[1] Security Testing

[2] Recovery Testing

آزمون بتا در سایت مشتریان و توسط مشتریان که از سیستم استفاده خواهند کرد صورت می‌گیرد و مشکلات مشاهده‌شده را به توسعه‌دهندگان گزارش می‌کنند.

۲-۴- راهبردهای آزمون

از مطرح‌ترین راهبردهای آزمون نرم‌افزار می‌توان به جعبه سفید، جعبه سیاه و جعبه طوسی اشاره داشت. این آزمون‌ها عموماً در قالب آزمون نفوذپذیری[۱] مطرح می‌گردند، همان‌طور که از نامش پیداست تمام توان در این آزمون‌ها برای پیدا کردن حفره‌ها و عیوب سیستم بکار گرفته می‌شود. در ادامه به‌طور مفصل درزمینه سه روش اشاره‌شده بحث می‌کنیم؛ اما پیش‌ازاین لازم است بدانی که تفاوت روش‌های مطرح‌شده در میزان اطلاعاتی از نرم‌افزاری است که در اختیار آزمون کننده قرار می‌گیرد. با این پیش مقدمه در ادامه این بخش به معرفی یکی از این راهبردها می‌رویم و از سایر روش‌های صرف‌نظر می‌کنیم.

۲-۴-۱- راهبرد آزمون جعبه سفید[۲]

در این قسمت برای معرفی راهبردهای آزمون، یکی از راهبردهای مهم معرفی خواهد شد. در ابتدا این سؤال را مطرح کنیم که آزمون جعبه سفید چیست؟ نام جعبه سفید این راهبرد به‌نوعی نمادین است، فقط در جهت هماهنگی با جعبه سیاه و حس تضاد به چنین اسمی معروف است، در غیر این صورت باید آن را جعبه شفاف یا شیشه‌ای نامید[۳]. این راهبرد یکی از روش‌های برجسته طراحی موارد آزمون یا آزمون نرم‌افزار است که هدف اصلی آن بررسی منطق درونی نرم‌افزار است.

درواقع طی این آزمون روال منطقی برنامه دنبال خواهد شد. در راهبرد جعبه سفید درواقع گویی جزئیات نرم‌افزار مانند کدهای منبع، مستندات طراحی و غیره را درون یک جعبه شیشه‌ای گذاشته‌اند و می‌توان محتویات داخل آن را مشاهده و از نحوه عملکرد آن آگاه شد.

افرادی که این راهبرد را پیاده‌سازی می‌کنند معمولاً اعضای تیم توسعه و تیم مستقل آزمون هستند[4]. این افراد توسط این راهبرد به منطق درونی و ساختار طراحی و کد نویسی نرم‌افزار احاطه خواهند داشت. به زبان ساده عرض کنم که وقتی از مستندات و جزئیات دقیق یک محصول نرم‌افزاری آگاه باشیم، قاعدتاً از نحوه پیاده‌سازی آن‌هم آگاه خواهیم بود؛ بنابراین فرد یا تیم موردنظر با آگاهی کامل از ریز جزئیات نرم‌افزار به بررسی آن می‌پردازد. ازجمله اعمالی که در این راهبرد انجام می‌شود:

---

۱ Penetration Test

۲ White-Box Testing Strategy

۳ Transparent-Box Testing Strategy

۱. بررسی خطوط کد منبع، به‌صورت خط به خط و جزءبه‌جزء، به صورتی که خطوط کد و مسیرهای مستقل داخل یک پیمانه حداقل یک‌بار اجرا و آزمون شوند.
۲. چک کردن تصمیمات منطقی برنامه، برای مثال تمامی شرط‌ها را، حتی else هایی که شاید هیچ‌گاه اجرا نشوند را آزمون کنیم. (چیزی شبیه کنترل نوع ایستا در طراحی و پیاده‌سازی زبان‌های برنامه‌سازی)
۳. همه حلقه‌ها باید آزمون شوند، زیرا میدانیم که حلقه‌ها نقش تأثیرگذاری در برنامه، خصوصاً در بخش حافظه ایفا می‌کنند؛ بنابراین بررسی روند کارکرد و فضای مصرفی آن‌ها لازم به نظر می‌رسد.
۴. از تمامی ساختارهای اطلاعاتی داخلی در جهت تضمین اعتبار نرم‌افزار استفاده کنیم.

۲-۵- روش‌های ایستا

ازجمله کاربردهای مهم روش‌های تحلیل کد، بازنگری کد مبدأ و تصحیح خطاها است. در اوایل دوران توسعه نرم‌افزارها، الزامی برای بررسی و بازبینی وجود نداشت تا این‌که در سال ۱۹۷۰ بازنگری رسمی برای تولید و توسعه محصولات نرم‌افزاری صورت گرفت. برای تحلیل، از دو رویکرد ایستا و پویا استفاده می‌شود. در ادامه به بررسی روش ایستا پرداخته خواهد شد.

برنامه‌نویسان اولیه تمایل به تحلیل کد به روش ایستا داشتند، اما این به معنی چشم‌پوشی از روش دیگر نیست، بلکه بهترین روش، استفاده ترکیبی از هر دو رویکرد است. تعاریف معین‌شده :

- درروش تحلیل کد ایستا، کد اصلی، قواعد مهم و نحوه استفاده از آرگومان‌ها بررسی می‌شود.
- درروش تحلیل کد پویا، کد اجرا می‌شود و نتایج وابسته به برنامه موردتحقیق قرار می‌گیرد.

به‌طور دقیق‌تر، در رویکرد تحلیل ایستا، بررسی کد بدون اجراشدن برنامه صورت می‌گیرد. در شرایطی خاص خود کدها بررسی می‌شوند.

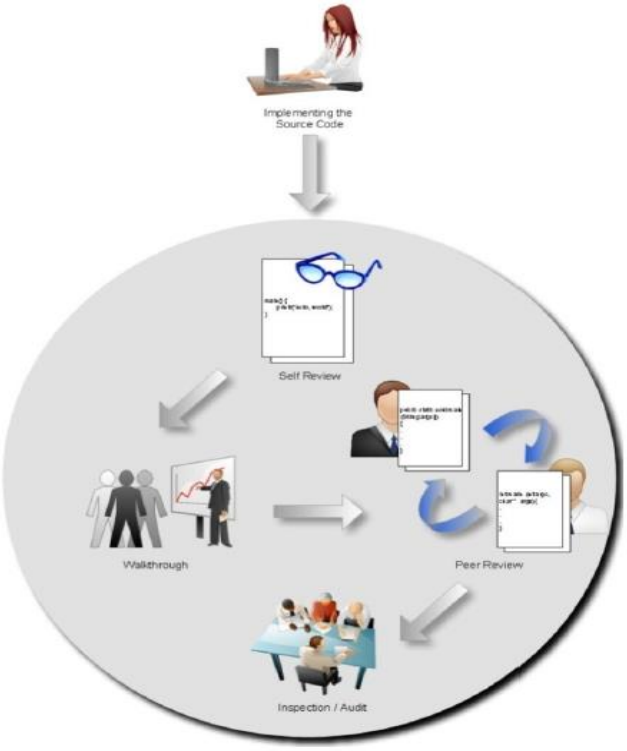
یکی از روش‌های این رویکرد مرور دستی[1] است. این‌گونه بررسی، در هر مرحله‌ای از توسعه نرم‌افزار انجام می‌شود اما بهترین زمان‌برای انجام آن، مراحل آغازین کد نویسی است؛ زیرا با پیشرفت پروژه، حجم کد نویسی بسیار زیاد می‌شود و تحلیل نقاط آسیب‌پذیر برنامه به‌سختی انجام خواهد شد. این بررسی فقط منحصر به مشاهده کدهای برنامه نیست، بلکه شامل بازنگری در مستندات، نیازمندی‌ها و طراحی هم می‌شود و می‌تواند توسط دو دسته از افراد صورت گیرد:

[1] Manual Review

۱. افرادی که به‌تنهایی کار بررسی و تحلیل را انجام می‌دهند.

۲. تیم تحلیل.

برای دسته اول، دانستن مراحلی که در توسعه نرم‌افزار طی می‌شود (طراحی، نیازسنجی و غیره) اهمیت بالایی دارد؛ اما برای دسته دوم، کار کمی پیچیده‌تر است. آن‌ها می‌توانند با تحلیل اولیه و ارائه آن به مخاطبانشان به نتیجه‌ای مطلوب برسند. شکل ۲-۱ نشان‌دهنده فرآیند تحلیل طبق دسته‌بندی بالا است.



شکل (۲-۱) فرآیند تحلیل ایستا

۲-۵-۲- کاربرد ابزارها در تحلیل ایستا

ابزارهایی که برای این کار طراحی‌شده است، غالباً برای تسریع در تحلیل کد به کار می‌روند. بهترین ابزارهای تحلیل، ویژگی‌های بارزی همچون استفاده آسان و ایمن رادارند؛ اما باید این نکته را یادآوری کرد که این ابزارها تمامی مشکلات و مسائل امنیتی برنامه را حل نمی‌کنند. مطابق زبان‌های برنامه‌نویسی مختلف، ابزارهای مناسبی تولیدشده است که در زیر فهرستی از آن را مشاهده می‌کنید:

۱. زبان‌های تحت .NET

FxCop
StyleCop

۲. زبان Java

FindBugs

PMD
Jlint

۳. زبان C++/C

Lint
CodeSonar
Mygcc

۲-۶- درستی‌یابی

در فرایند درستی‌یابی یک قطعه کد، هدف رسیدن به یکی از دو وضعیت نهایی «درستی کد» و یا «خطای کد» است[5]. لازم به ذکر است که درستی‌یابی در این پژوهش به‌تنهایی معنایی نمی‌پذیرد و درستی‌یابی یک قطعه کد به ازای یک سیستم نوع خاص مدنظر است. درنتیجه «درستی» یک قطعه کد در این پژوهش به معنی سازگاری تمام اجزای کد در رابطه با سیستم نوع موردنظر است و در طرف مقابل، «خطای کد» به معنی وجود یک یا بیش از یک ناسازگاری در کد خواهد بود.

۲-۷- جمع‌بندی

هدف از این فصل، همانطور که مشاهده شد، آشنایی مختصری با مفاهیم لازم و استفاده شده در فصل‌های بعد بود و تلاش شد تا تنها به بیان آن‌دسته از مطالب که در ادامه بیان مجدد نگشته است، پرداخته شود. در این بخش تعریفی از سیستم های نوع ارایه شد. این مفهوم در بخش ۴-۴-۳ به دقت بررسی خواهد شد و ارتباط آن با پروژه شفاف خواهد شد. همچنین تعریفی ساده از درستی‌یابی ارایه شد.

فصل ۳:

کارهای مرتبط

۳-۱- مقدمه

در این بخش سعی می‌شود تا نگاهی به کارهای انجام‌شده در حوزه پژوهش جاری پرداخته شود. ازآنجاکه پژوهش انجام‌شده ترکیبی از چند حوزه درستی‌یابی سیستم‌های نرم‌افزاری، جمع‌سپاری یک فعالیت فردی و همچنین استفاده از بازی‌های ویدئویی در اجرای عمل است، حوزه‌ای جدید و خاص محسوب می‌گردد. ازاین‌جهت تنها کار جدی مرتبط با این پژوهش، در دنیا تنها توسط یک تیم از دانشگاه واشنگتن[1] انجام‌گرفته است[6].

اگر بتوان پژوهش انجام‌شده را به دو بخش کلی:

۱. تلاش برای درستی‌یابی یک نرم‌افزار به‌صورت خودکار
۲. تلاش برای نگاشت یک فعالیت به بازی به کمک جمع‌سپاری

تقسیم نمود، می‌توان به یکی از پژوهش‌های دانشگاه برکلی[2] اشاره نمود[2]. در پژوهش انجام‌شده توسط این تیم به‌منظور کشف الگوها از انسان‌ها که در این عمل از ماشین می‌توانند بهتر عمل کنند بهره گرفته‌شده است و برای رسیدن سریع‌تر به این هدف، شیوه جمع‌سپاری در قالب یک بازی کامپیوتری صورت گرفته است.

ادامه این فصل بدین‌صورت سازمان‌دهی شده است: به ترتیب در بخش‌های ۳-۲ و ۳-۳ مقاله‌های دانشگاه برکلی و واشنگتن بررسی‌شده است. در بخش ۳-۳ علاوه بر بررسی گزارش و سیستم ارائه‌شده، به مقایسه و بررسی تناظر آن با سیستم و بخصوص بازی ارائه‌شده در پژوهش جاری پرداخته خواهد شد. در بخش پایانی نیز به جمع‌بندی مقاله‌های اشاره‌شده پرداخته خواهد شد.

۳-۲- توضیح پژوهش دانشگاه برکلی

در این بخش مقاله‌ای با عنوان[3] «کرادماین: به‌سوی جمع‌سپاری کردن ابزارهای کمکی در درستی‌یابی توسط انسان» موردبررسی می‌گردد. این پژوهش یکی از معدود پژوهش‌های انجام‌شده در حوزه جمع‌سپاری فرآیند درستی‌یابی به‌واسطه بازی‌های ویدئویی است.

تیم پژوهشگر در این مقاله باهدف انتقال فرآیند درستی‌یابی به دست مردم نامتخصص که به‌راحتی

[1] washington.edu

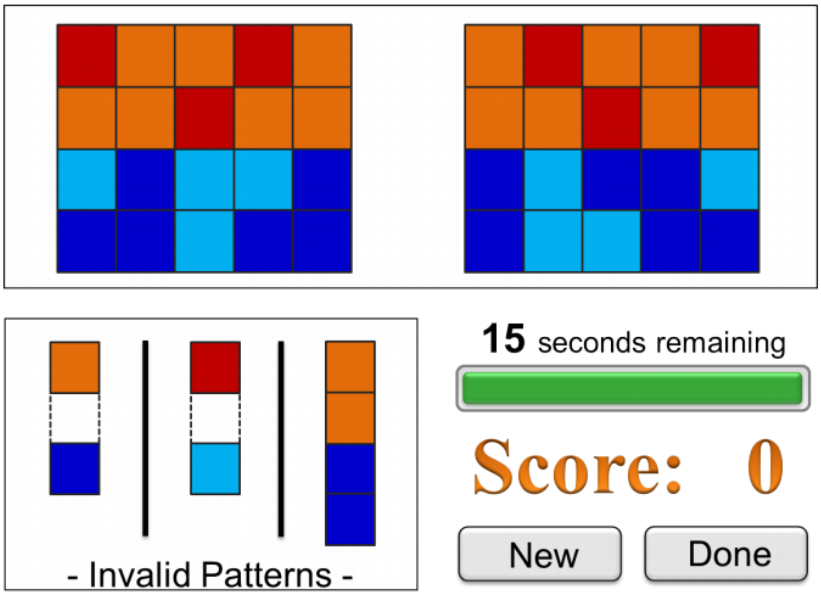
[2] berkeley.edu

[3] CrowdMind: Towards Crowdsourced Human-Assisted Verification

می‌توانند کارهای ساده و فعالیت‌های تکراری را انجام دهند، از یک بازی کمک گرفته است. در این بازی فعالیت اصلی تشخیص الگوها است. فعالیتی که انسان به‌راحتی توانایی انجام آن را دارا است.

در این پژوهش فرآیند درستی یابی به‌صورت یکجا به بازی نگاشت نشده است. بلکه هدف این تیم، نگاشتی تنها قسمت‌هایی از فرآیند درستی یابی، عموماً آن دسته از فعالیت‌هایی که ساده ولی وقت‌گیر هستند، به بازی فوق است.

ایده اصلی این پژوهش در تشخیص الگوهای موجود در یک تصویر دوبعدی به‌منظور کشف و استخراج خصوصیات کد است. در شکل ۳-۱ نمایی از بازی اشاره‌شده آورده شده است.



شکل (۳-۱) بازی CrowdMind

۳-۳- توضیح پژوهش دانشگاه واشینگتن

همان‌طور که پیش‌تر در مقدمه اشاره شد، کار انجام‌شده توسط این دانشگاه منحصربه‌فردترین پژوهشی است که ارتباط مستقیمی با پژوهش جاری دارد. این پژوهش در قالب مقاله‌ای با عنوان «بازی‌های درستی یابی: تبدیل درستی‌یابی به تفریح[1]» آورده شده است.

یکی از دلایل قوت این پژوهش ناشی از همکاری دوجانبه دو تیم تخصصی این دانشگاه، گروه مهندسی زبان‌های برنامه‌سازی و نرم‌افزار[2] و مرکز علوم بازی[3]، است. لازم به ذکر است که به خاطر اهمیت بالای حوزه

[1] Verification Games: Making Verification Fun

[2] Programming Languages & Software Engineering Group

[3] Center for Game Science

جمع‌سپاری درستی‌یابی نرم‌افزار و ارزش نتایج حاصل از این پژوهش، پژوهش این تیم در سال ۲۰۱۲ زیر نظر و حمایت سازمان پروژه‌های تحقیقاتی پیشرفته دفاعی[1] قرارگرفته و از حمایت‌های این سازمان‌برای پیشبرد پروژه استفاده نموده‌اند. همچنین سازمان فوق، برنامه‌ای[2] مختص تحقیقات بر روی جمع‌سپاری درستی بابی نرم‌افزارها با عنوان CSFV نیز ایجاد نموده است. کارهای انجام شده تحت پوشش این برنامه در منابع [8-13] آورده شده است که در این پژوهش‌ها نتیجه عملی مطلوبی بدست نیامده است.

در این بخش مبنای صحبت پژوهش این دانشگاه بوده و به بررسی ابعاد مختلف و کارهای انجام‌شده در آن پرداخته خواهد شد. هدف اصلی پژوهش این تیم ایجاد روشی جایگزین برای شیوه معمول درستی‌یابی در حوزه ایستا است. آن‌ها اهداف این کار را بدین شکل می‌شمارند:

۱. کاهش هزینه‌های درستی‌یابی به شیوه ایستا
۲. افزایش نیروی کار ممکن در این حوزه
۳. گسترش و شیوع استفاده از روش‌های درستی‌یابی ایستا برای تمام سطوح نرم‌افزاری (نه‌فقط نرم‌افزارهای حساس و بزرگ)
۴. کاهش سطح و میزان دانش لازم برای فعالیت‌های درستی‌یابی

این تیم انگیزه اولیه خود از انتخاب درستی‌یابی به شیوه‌های ایستا را ضعف در شیوه‌های پویا و هزینه‌های بالا در شیوه‌های ایستا ذکر می‌کند و هیچ‌کدام از این دو روش را، به سبک فعلی و با توجه به حجم نرم‌افزارهای موجود و درحال‌توسعه، کافی و مناسب نمی‌داند. آن‌ها نتیجه پژوهش خود را در افزایش سطح اطمینان نرم‌افزارهای در سطح وسیع می‌دانند و آینده این حوزه را در صورت وجود شیوه‌های ارزان‌قیمت جایگزین، بسیار روشن‌تر می‌دانند.

این تیم ایده خود را در قالب نگاشت فعالیت درستی‌یابی به سبک فعلی به یک یازی ویدئویی به نام PipeJam باقابلیت جمع‌سپاری بیان می‌کند. همان‌طور که مشاهده می‌شود این دقیقاً مشابه پژوهش جاری نگارنده بوده و این دو در ایده اصلی خود به یکدیگر بسیار نزدیک هستند.

نرم‌افزار ارائه‌شده توسط تیم دانشگاه واشنگتن دو ورودی خواهد داشت:

۱. یک قطعه کد به زبان جاوا و
۲. یک مؤلفه مورد آزمون یا همان یک سیستم نوع.

[1] Defense Advanced Research Projects Agency (DARPA)

[2] www.DARPA.mil/Our_Work/I2O/Programs/Crowd_Sourced_Formal_Verification_%28CSFV%29.aspx

درواقع قطعه کد ورودی ازنقطه‌نظر سیستم نوع موردنظر تحلیل و حاصل این تحلیل همان بازی PipeJam خواهد بود. حاصل نهایی نرم‌افزار فوق یکی از دو مورد زیر می‌شود:

۱. اثبات درستی قطعه کد ورودی که نشان می‌دهد برنامه ورودی ازنقطه‌نظر سیستم نوع (همان مؤلفه) انتخابی مشکلی ندارد، یا

۲. مکان‌هایی از قطعه کد ورودی که کد در آن مکان‌ها نیازمند تجدیدنظر توسط برنامه‌نویس است و ازنقطه‌نظر سیستم نوع موردنظر دارای ناسازگاری است.

درواقع، سیستم ارائه‌شده به‌صورت خودکار قطعه کد ورودی را بر اساس سیستم نوع موردنظر به یک چیدمان از یک مرحله در بازی PipeJam تبدیل خواهد نمود که می‌تواند توسط مردم با هیچ‌دانشی در خصوص درستی‌یابی بازی شود. وقتی بازیکن معمای یک مرحله از بازی را تمام کند، سیستم بر اساس امتیاز کسب‌شده، چیدمان جاری عنصرهای بازی را به اثباتی برای درستی قطعه کد اولیه ترجمه خواهد نمود. به‌صورت دقیق‌تر، چیدمان بازی معادل یک انتخاب از سیستم نوع در کد اصلی خواهد بود.

در سیستم فوق اگر حاصل نهایی مرحله پس از اتمام نشان از وجود ناسازگاری در کد اصلی بدهد و نتواند اثباتی از درستی کد را ارائه دهد، می‌تواند به علت یکی از این دو حالت زیر باشد:

۱. قطعه کد ورودی امن نبوده و ازنظر سیستم نوع موردنظر دارای سازگاری است.

۲. قطعه کد ورودی مشکلی نداشته و اثبات درستی کد فراتر از توانایی‌های روش‌های درستی‌یابی فعلی است. این درواقع مشابه هشدارهای اشتباه[1] در مترجم‌های امروزی است.

البته مقاله در ادامه توضیحات بیشتری در ارتباط باحالت دوم ارائه نمی‌دهد و مثالی از آن را نشان نمی‌دهد. درواقع آن‌ها با اشاره به اینکه تشخیص علت ناامنی از بین دو حالت فوق ناممکن بوده، از این مورد در ادامه صرف‌نظر می‌نماید. سیستم در این موارد با گزارش خطایی از مکان این ناامنی، تشخیص را به برنامه‌نویس واگذار می‌نماید. در این حالت برنامه‌نویس خود می‌تواند تشخیص دهد که آیا خطای گزارش‌شده یک ناامنی بی‌مورد بوده و یا کد نیازمند اصلاح است.

بنا بر مقاله این تیم درصورتی‌که یک بازیکن (حتی بازیکنانی که در طول زمان به مهارت بالا در بازی دست‌یافته‌اند) نتواند به‌واسطه اشکالات کد آن را درستی‌یابی کنند، یک درستی‌یابی نسبی[2] نیز خود ارزشمند بوده. این مقوله در پژوهش نگارنده نیز لحاظ گردیده است.

[1] False True

[2] Partial Verification

در ادامه مقاله، ویژگی‌های اصلی و بارز یک ابزار بررسی سیستم‌های نوع بررسی گشته است؛ اما ازآنجایی‌که این ابزارها درروند کار پژوهش جاری نگارنده بی‌تأثیر بوده و همچنین آگاهی از این ویژگی‌ها دانشی در راستای پژوهش این حوزه بر خواننده نمی‌افزاید. از بررسی آن صرف‌نظر می‌گردد؛ اما این نکته حائز اهمیت است که سیستم تعبیه‌شده این تیم از یک شیوه اتصال‌پذیر[1] بهره می‌جوید. این به معنی آن است که سیستم توانایی پشتیبانی از سیستم‌های نوع متفاوت در طول زمان را خواهد داشت و این تکنیک در پژوهش جاری، به‌گونه‌ای متفاوت، وجود دارد.

خطاهای پشتیبانی شده در پژوهش با توجه به تحلیل و توجه به لیستی از خطرناک‌ترین خطاهای نرم‌افزاری[2] که توسط موسسه CWE[3] ارائه‌شده است و نگاشت آن‌ها به سیستم‌های نوع ممکن به‌دست‌آمده است. از مهم‌ترین سیستم‌های نوع موردتوجه قرارگرفته در این سیستم می‌توان به: اشاره‌گرهای تهی[4]، مقداردهی اولیه[5]، کلید در ساختار داده نگاشت[6]، بررسی تساوی‌ها[7]، اثرات جانبی[8] به‌واسطه تغییرهای ناخواسته[9]، همزمانی و قفل‌گذاری[10]، جریان اطلاعات (سطح امنیت)[11] و بسیاری دیگر اشاره داشت.

در این سیستم اطلاعات یک سیستم نوع (موردنظر) از کد مبدأ به شبکه‌ای از لوله‌ها مبدل می‌گردد. عرض دهانه لوله‌ها که می‌تواند توسط بازیکن کنترل و تنها در دو حالت بزرگ یا کوچک قرار گیرد، به‌صورت مستقیم از وضعیت نوع موجود در برنامه نگاشت می‌گردد. همچنین سایر ارتباطات و محدودیت‌های موجود بر این لوله‌ها و سایر عنصرهای موجود در بازی، حاصل محدودیت‌ها و ارتباط‌های موجود بر نوع‌های موجود در کد است.

---

1 Pluggable

2 http://cwe.mitre.org/top25/?2011

3 cwe.mitre.org

4 Null Pointer

5 Initilization

6 Map Key

7 Equality Tests

8 Side Effects

9 Incorrect Mutation

10 Concurrency and Locking

11 Information Flow (Trust and Security)

برای روشن‌تر شدن این مسئله مثال زیر را در نظر بگیرید. ابتدا فرض می‌کنیم سیستم نوع موردنظر، تهی یا مقدار دار بودن اشاره‌گرها باشد (سیستم نوع اشاره‌گرهای تهی). در این حالت طبیعتاً برای هر متغیر برنامه‌نویس یک نوع مشخص کرده است:

۱. حتماً مقدار دارد، یا

۲. می‌تواند تهی نیز باشد.

در این حالت منظور از نوع‌های موجود در کد، نوع مشخص‌شده برای هر متغیر است و منظور از یک محدودیت حاکم بر کد، فرض‌های برنامه‌نویس از نوع حاکم بر یک متغیر است (که غالباً به کمک دستورات ادعایی[1] است).

درروند بازی، بازیکن باید سعی کند به شکلی بهینه برای هر متغیر نوعی مشخص نماید. این کار از دیدگاه این تیم بسیار ارزشمند است. چراکه مسئله استنتاج نوع به‌صورت کلی[2] همچنان حل‌نشده است که می‌توان از جمع‌سپاری در حل آن بهره جست.

همچنین در رابطه با علت این دیدگاه، این تیم در گزارش خود بیان می‌کند که آن‌ها معتقدند که انسان‌ها در دسته‌ای از شرایط مانند زمان‌هایی که قطعه کد مبدأ قابلیت اثبات درستی نداشته باشد، از هوش مصنوعی ماشینی بسیار بهتر عمل خواهند کرد.

کد زمانی امکان تأیید درستی ندارد که در آن کد یک یا چندین ناسازگاری نوعی وجود داشته باشد؛ و یا همان‌طور که پیش‌تر بیان شد، علت درستی آن فراتر از دانش علوم درستی‌یابی فعلی باشد. در هرکدام از این دو حالت، دسته‌ای از بازیکنان از یک ماشین بهتر عمل کرده و یک گزارش خطا برای برنامه‌نویس ارسال خواهد شد.

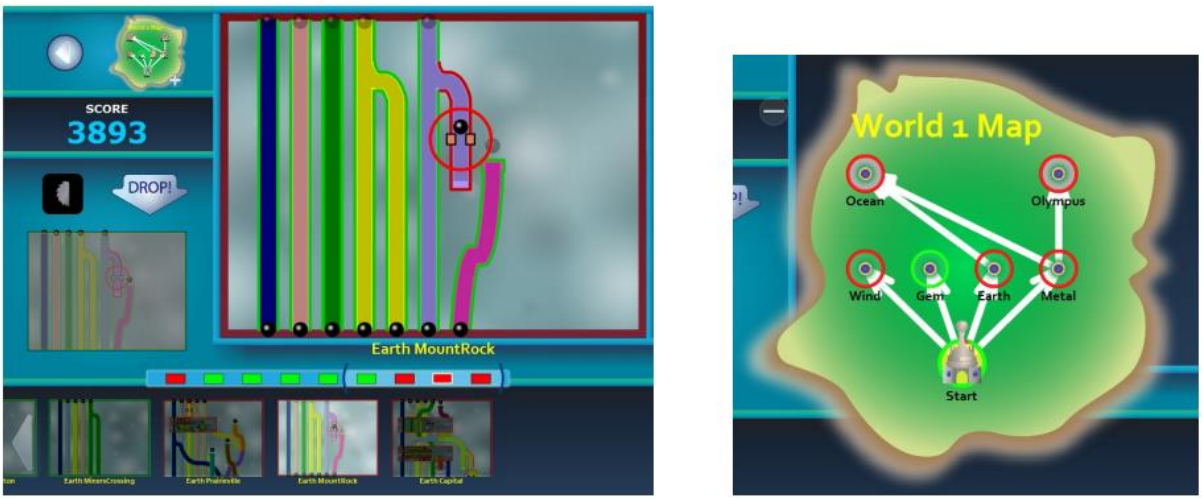
حال در این بخش به جزییات بازی ارائه‌شده توسط این تیم پرداخته می‌شود و سعی می‌شود تا به‌صورت کامل مفاهیم موجود در اجزای بازی، ویژگی مهم سیستم و نقطه‌ضعف‌های احتمالی آن بررسی و با سیستم ارائه‌شده در پژوهش جاری مقایسه گردد. در بعضی موارد اطلاعات ارائه‌شده در این بخش حاصل بررسی شخصی نگارنده بوده و برای اطلاعات بیشتر می‌توان از وب‌سایت[3] اختصاص داده‌شده به این پژوهش دریافت نمود. در شکل ۳-۲ نمایی از بازی که در گزارش این تیم آمده را می‌توان مشاهده نمود:

---

[1] Assertion

[2] Genral Type Inference

[3] http://cs.washington.edu/verigames



شکل (۳-۲) نمایی از بازی PipeJam

در بازی PipeJam بازیکن با یکسری لوله و توپ روبرو است. هر لوله یا عریض است و یا باریک. همین‌طور توپ‌ها یا بزرگ خواهند بود و یا کوچک. توپ‌های بزرگ از لوله‌های باریک عبور نخواهند کرد و یک توپ کوچک از هر لوله‌ای عبور خواهد کرد. بازیکن در بازی می‌تواند نوع بعضی از لوله‌ها را که در بازی به‌عنوان مجاز مشخص‌شده‌اند، عوض کند. در ابتدا یکسری توپ در بالای لوله‌های هر مرحله قرارگرفته است. هدف بازی گذر تمامی توپ به انتهای آن‌ها است.

یک بازیکن شاید بخواهد تمام لوله‌ها را به حالت عریض ببرد تا بازی را تمام کند. ولی این کار به دلیل محدودیت در بعضی لوله‌ها امکان‌پذیر نخواهد بود. این عدم امکان به دو حالت در بازی مشخص‌شده است:

۱. لوله‌های بی‌رنگ
۲. وجود گلوگاه[1] در بعضی مکان‌ها که امکان گذر توپ بزرگ را نخواهند داد.

همچنین در بعضی مراحل بعضی از توپ‌ها امکان تغییر اندازه را نخواهند داشت. این توپ‌ها بارنگ خاکستری مشخص‌شده‌اند.

درواقع محدودیت‌های حاکم بر بازی شامل اندازه‌های ثابت و نحوه انتقال یک لوله به لوله‌های دیگر است. بازی وقتی تمام خواهد شد که بازیکن طوری اندازه لوله‌ها و همچنین توپ‌ها را تعیین کند که تمام محدودیت‌های حاکم بر بازی (اندازه‌های ثابت و نحوه انتقال یک لوله به لوله‌های دیگر) را ارضا و توپ‌ها بتوانند از لوله‌ها با پایین آن‌ها گذر کنند.

ایده ساده بازی PipeJam به جمع‌سپاری آن کمک می‌کند و به هرکسی اجازه خواهد داد که این بازی را در زمان بسیار کوتاهی فرابگیرد. این نکته در بازی ارائه‌شده در این پژوهش موردتوجه قرارگرفته.

[1] Pinch Point

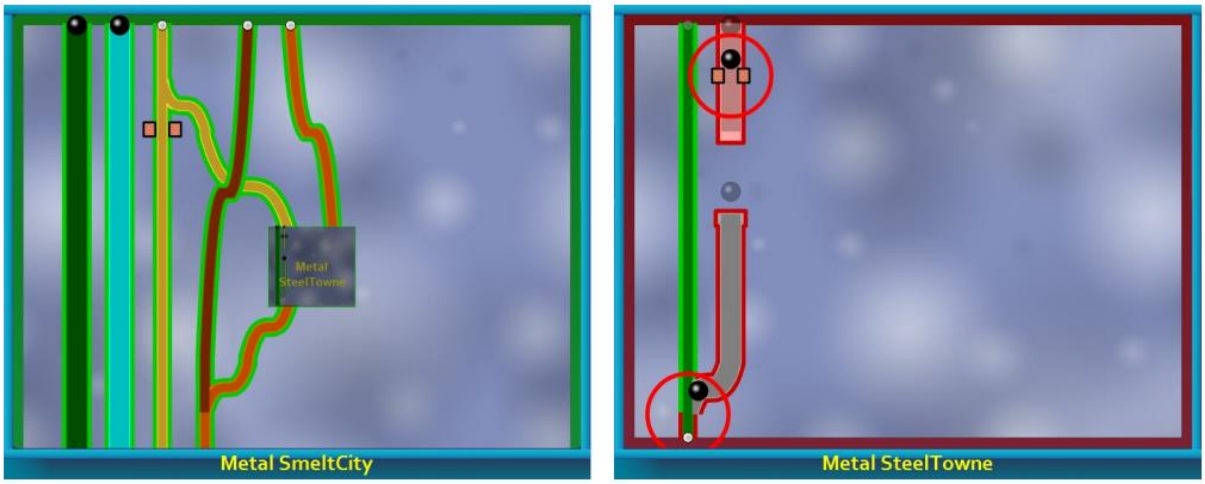
۳-۳-۲- بازی PipeJam

در ادامه به بررسی اجزای بازی PipeJam و تناظر آن با بازی ارائه‌شده در پژوهش جاری پرداخته می‌شود.

برد بازی[1]: بازی PipeJam به مجموعه‌ای بردها، مرحله‌ها و جهان‌ها تقسیم گشته است. در این بازی، یک برد به یک شبکه از لوله‌ها اطلاق می‌گردد. تصاویر بازی هرکدام بیانگر یک برد در این بازی هستند.

مرحله بازی[2]: یک مرحله از مجموعه‌ای چند برد تشکیل می‌گردد. غالباً مجموعه‌ای بردهای مرتبط در کنار هم یک مرحله را تولید می‌کنند. برد در یک مرحله به معنی حل شدن تمام بردها به‌صورت همزمان خواهد بود.

جهان بازی[3]: جهان در بازی PipeJam درواقع مفهومی انتزاعی بوده تا وجود مراحل مختلف در کنار یکدیگر را سازمان‌دهی کند و علت کاربردی دیگری برای درستی‌یابی کد نخواهد داشت. در بازی ارائه‌شده در پژوهش نگارنده گرچه تنها بردها مستقل ارائه گشته ولی ازلحاظ کار تحقیقاتی هر دو جز مرحله و جهان به‌راحتی قابل پیاده‌سازی است و لزوم آن در کارهای آینده ذکر گشته است. نمایی از این مفهوم انتزاعی در شکل ۳-۳ آورده شده است.



شکل (۳-۳) نمایی از شبکه‌های توکار بازی PipeJam

لوله‌ها[4]: در این بازی همانند بازی پژوهش جاری، هر لوله با یکرنگ مشخص گردیده. کاربرد اصلی این رنگ در مشخص نمودن لوله‌هایی با ماهیت یکسان ولی دربردهای مختلف یک مرحله است. درنتیجه اگر یک

[1] Game Board

[2] Game Level

[3] Game World

[4] Pipes

لوله آبی‌رنگ در یک برد توسط بازیکن عوض گردد، تمام خطوط لوله آبی در سایر بردها نیز عوض خواهند شد. درواقع همین موضوع اصلی‌ترین چالش موجود در بازی را پدید می‌آورد.

شبکه‌های توکار[1]: یکی از مسائلی که به‌واسطه زبان‌های شیءگرا و تابعی در بازی خودنمایی می‌کند، وجود شبکه‌های تودرتو در یک مرحله است. درواقع این جز از بازی بازتاب توابع موجود در کد خواهد بود. هرچند ازآنجاکه پژوهش این تیم محدود به زبان جاوا بوده، ولی در پژوهش جاری این تکنیک به‌صورت کلی و برای تمام زبان‌هایی که امکان فراخوانی تابعی دارند، گسترش‌یافته است.

اره‌برقی: ازآنجایی‌که تمام مراحل قابل‌حل نخواهند بود، باید تکنیکی برای گزارش موضوع عدم امکان اثبات درستی در بازی وجود داشته باشد. در بازی PipeJam ابزار اره‌برقی باعث می‌شود تا توپ‌های عبوری از خود را کوچک کند. در بازی ارائه‌شده در این پژوهش ابزار تبدیل‌کننده جادویی نقشی متناظر با اره‌برقی را ایفا خواهد نمود که می‌توان هردو را ازنقطه‌نظر کارایی در یک طبقه قرارداد.

سیستم امتیازدهی: ازآنجایی‌که هر مرحله به چندید روش مختلف قابل‌حل است و بعضی راه‌حل‌های مدنظر هدف سیستم نیست، نیازمند سیستمی برای ارزش‌گذاری راه‌حل‌های مختلف هستیم. ازاین‌رو این سیستم از اهمیت بسیار بالایی در این بازی برخوردار است. این بخش به تفضیل در پژوهش جاری اشاره خواهد شد. در بازی Pipejam امتیازدهی بر اساس تعداد مراحل باقی‌مانده برای حل، تعداد بردهای حل‌نشده در هر مرحله، ناسازگاری‌های موجود در هر برد و تعداد اره‌برقی‌های استفاده‌شده محاسبه می‌گردد.

۳-۳-۳- نحوه نگاشت سیستم‌های نوع به بازی

به‌صورت کلی هر سیستم نوع از تعدادی توصیف‌کنند[2] تشکیل‌شده است. هر متغیر می‌تواند یکی از توصیف‌کننده‌های یک سیستم نوع را اختیار کند. به‌عنوان‌مثال سیستم نوع اشاره‌گرهای تهی دارای دو توصیف‌کننده قابل تهی شدن و قابل تهی نشدن است. به‌عنوان‌مثال دیگر، سیستم نوع، نوع داده‌ای از تعداد نامحدودی توصیف‌کننده تشکیل‌شده است. چراکه به ازای هر متغیر تعداد نامحدودی نوع داده‌ای قابل انتساب وجود دارد.

با توجه به مقدمه بالا، در پژوهش این تیم فرآیند نگاشت از دو جنبه قابل‌بررسی است. در زیر به توضیح هرکدام پرداخته خواهد شد

[1] Embeded Networks

[2] Qulifier

- **کار انجام‌شده است**

در سیستم ارائه‌شده تنها آن دسته از خطاهایی از میان لیست خطاهای خطرناک مدنظر قرارگرفته که قابل نگاشت به یک سیستم نوع با دو توصیف‌گر باشد. علت این محدودیت می‌تواند از طراحی بازی[1] PipeJam نشأت‌گرفته باشد. چراکه در این بازی لوله‌ها تنها می‌توانند یکی از دو اندازه باریک و یا عریض را انتخاب کنند.

- **دیدگاه کلی به سیستم نگاشت**

باوجود کارهای توضیح داده‌شده، از دیدگاه تیم دانشگاه واشنگتن، می‌توان سیستم را برای پشتیبانی از سیستم‌های نوع با بیش از دو توصیف‌گر ارتقا داد که این کار یکی از ویژگی‌های بارز و انجام‌شده در پژوهش جاری است.

همچنین یکی دیگر از محدودیت‌های سیستم فعلی عدم توانایی در به‌کارگیری از سیستم‌های نوع با توصیفگرهای غیرقابل مقایسه است. به‌عنوان یک مثال از سیستم نوع نوع داده‌ای، دو توصیفگر نوع داده‌ای A و B می‌توانند کاملاً مجزا و بی‌ربط با یکدیگر باشد. در اینجا داده‌ای از جنس A بر B برتری و یا همپوشانی ندارد و نمی‌توان هیچ‌کدام را بالاتر از دیگری دانست. این در حالی است که در طراحی بازی PipeJam تنها ملاک مقایسه بوده. این محدودیت نیز در پژوهش جاری به‌واسطه طراحی بهتر بازی حل گشته است که در فصل چهارم توضیح داده‌شده است.

۳-۴- وب‌سایت VeriGames[2]

هدف از این بخش معرفی وب‌سایتی است که به‌منظور معرفی بازی‌هایی باهدف درستی‌یابی کد ارائه‌شده است. شکل ۳-۴ نمایی از این وب‌سایت را نمایش می‌دهد.

[1] Game Design

[2] http://verigames.com

شکل (۳-۴) سایت VeriGames

این وب‌سایت باهدف آزمون پروژه‌های درستی‌یابی به کمک فرایند جمع‌سپاری توسط سازمان پروژه‌های تحقیقاتی پیشرفته دفاعی و با همکاری یکی از بزرگ‌ترین اجتماعات[1] برنامه‌نویسان جهان به نام TopCoder[2] ایجاد گشته است.

در حال حاضر پنج بازی زیر در این سایت به‌صورت آزمایشی قرارگرفته است:

۱. بازی CircuitBot[3]
۲. بازی FlowJam[4]
۳. بازی GhostMap[5]
٤. بازی StormBound[6]

[1] Community
[2] TopCoder.com
[3] http://circuitbot.verigames.com
[4] http://flowjam.verigames.com/
[5] http://ghostmap.verigames.com/
[6] http://stormbound.verigames.com/

۵. بازی Xylem[1]

اطلاعات جمع‌آوری‌شده در این سایت بر اساس مقررات سایت[2] در اختیار برنامه CSFV که زیر نظر سازمان پروژه‌های تحقیقاتی پیشرفته دفاعی است، گذاشته می‌شود.

۳-۵- نتیجه‌گیری

همان‌طور که مشاهده شد در حوزه درستی‌یابی با روش‌های نوین اقدامات محدودی صورت گرفته و این حوزه از حوزه‌های بسیار با پتانسیل برای انجام کارهای جدید است. این در حالی است که ارزش نتایج حاصل از تحقیقات در چنین حوزه‌هایی به‌صورت مستقیم درآمدزا خواهد بود. شاید یکی از دلایل ایجاد برنامه‌ای اختصاصی با همین عنوان توسط سازمان پروژه‌های تحقیقاتی پیشرفته دفاعی برای ایجاد انگیزه در بین پژوهشگران ایالات‌متحده آمریکا، چشم داشتن به نتایج بسیار کاربردی حاصل از آن و همچنین وجود خلأ در این حوزه باشد.

[1] http://xylem.verigames.com/

[2] http://verigames.com/privacy

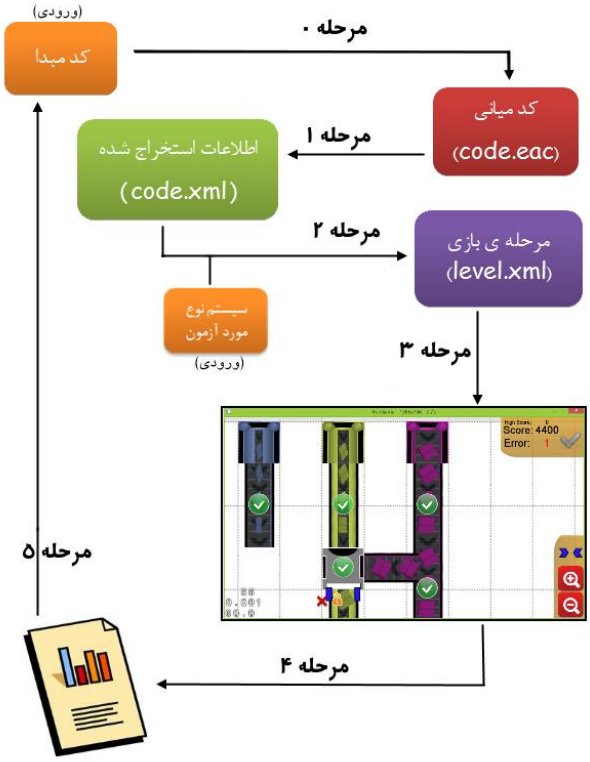
فصل ٤:

ایده پیشنهادی

۴-۱- مقدمه

حاصـــل این پژوهش در قالب یک ســـیســـتم قابل‌ارائه اســـت که هر بخش آن‌یک وظیفه خاص در چرخه درســتی‌یابی قطعه کد مبدأ خواهد داشت. منظور از سیستم در ادامه این گزارش، مجموعه‌ای از چند نرم‌افزار مختلف است که هر یک به‌تنهایی خروجی کاربردی‌ای نخواهند داشت.

شـــکـــل شـــماره ۴-۱ نمایی کلی از فرآیندها را نمایش می‌دهد که در ادامه هر بخش و وظیفه آن را معرفی می‌شود.



شکل (۴-۱) مراحل درستی‌یابی در سیستم ارائه‌شده

۱. مرحله صفرم:

این بخش درواقع وظیفه تبدیل کد اصـــلی را به کد میانی دارد. کد اصـــلی به کدی به زبان نامشـــخص اطلاق می‌گردد. این زبان محدود به یک یا دو زبان نبوده. ما به ازای هر زبان یک مبدل خواهیم داشت تا آن زبان را به زبان مبدأ ســیســتم (زبان میانی با پسوند eac که مخفف دو کلمه ea و code بوده) تبدیل نماید. به‌صورت دقیق‌تر، این مرحله از وظایف ســیستم نبوده و از آن در ادامه این گزارش صرف‌نظر می‌گردد. البته توضیحات کاملی از زبان میانی و گرامر آن در این فصل وجود دارد تا مبنایی برای نوشتن این مبدل‌ها قرار بگیرد.

۲. **مرحله اول:**

سیستم پس از دریافت کد میانی، با انجام مجموعه‌ای از پردازش‌ها و تشکیل جداول نماد[1] به استخراج اطلاعات کد میانی می‌پردازد. این اطلاعات در قالب یک فایل در اختیار بخش بعدی سیستم قرار می‌گیرد.

۳. **مرحله دوم:**

در این بخش با استفاده از اطلاعات استخراج‌شده در مرحله اول و بر اساس یک مؤلفه ورودی که در عمل یک سیستم نوع از بین سیستم‌های نوع مورد پشتیبانی سیستم است، چیدمانی از عنصرهای موجود در بازی را در قالب یک فایل خروجی تولید می‌گردد. به ازای یک کد ورودی و به سیستم‌های نوع مختلف، مراحل تولیدی می‌توانند متفاوت باشند.

۴. **مرحله سوم:**

در این مرحله فایل مرحله بازی، به خود بازی تبدیل گشته و در اختیار بازیکنان قرار می‌گیرد. مرحله جمع‌سپاری از انتشار این بازی حاصل می‌گردد.

۵. **مرحله چهارم:**

پس از اتمام بازی، اطلاعات حاصل که درواقع همان چیدمان نهایی اجزا بازی است، بر اساس معیارهایی مانند امتیاز کسب‌شده، به سیستم به‌منظور تصمیم‌گیری برای بازتاب یا عدم بازتاب به برنامه‌نویس، گزارش می‌گردد.

۶. **مرحله پنجم:**

در این مرحله در صورت لزوم برنامه‌نویس به اصلاح کد خود می‌پردازد. این مرحله نیز درواقع بخشی از فرآیند درستی‌یابی بوده و جزئی از سیستم نیست.

این بازی باید بین جمعیت بازیکنان در سطح وسیع توزیع گردد. درنتیجه بازی باید طوری طراحی شود که از طریق اینترنت قابلیت ارتباط و انتقال یکسری اطلاعات از بازیکنان را داشته باشد. بر اساس امتیازهای کسب‌شده در بازی توسط بازیکنان و ارتباط آن با درستی کد که توصیف آن و نحوه محاسبه آن در همین فصل آمده است، اطلاعات مفید برای درستی‌یابی قطعه کد آغازین به دست می‌آید.

سیستم در مرحله چهارم بر اساس اطلاعات حاصله از بازیکنان، یکی از دو حالت زیر را به برنامه‌نویس کد، باز خواهد گرداند:

۱. کد ازنقطه‌نظر مؤلفه مورد آزمون صحیح بوده و خطایی در کد وجود ندارد.

---

[1] Symbol Tables

۲. کد در بعضی مکان‌ها دارای اشکال بوده و راه‌حل احتمالی نیز ارائه داده خواهد شد.

ازآنجاکه نتیجه بازگردانده شده به برنامه‌نویس ماحصل فکر هزاران انسان است، می‌توان این اطمینان را داشت که احتمال خطای انسانی بسیار پایین بوده. همچنین ازآنجاکه بازیکن در طول زمان بازی را بارها و بارها انجام داده است، می‌توان ناتوانی بازیکن را به‌عنوان عامل خطا حذف نمود.

در جدول ۴-۱ مراحل توضیح شده در بالا از نحوه تبدیل قطعه کد آغازین تا ارائه خطا، یا اثبات درستی کد آمده است:

جدول (٤-١) هدف و فایل حاصل از هر مرحله

شماره مرحله	هدف مرحله	فایل خروجی مرحله
۱	استخراج اطلاعات کد	Code.xml
۲	نگاشت کد به مرحله با توجه به سیستم نوع مورد آزمون	Level.xml
۳	تولید فضای بازی	Game.exe
۴	ارائه گزارش با توجه به عملکرد بازیکن در بازی	Result.log

در ادامه این فصل به تشریح دقیق تمامی مراحل و فرآیندهای هرکدام به‌صورت دقیق پرداخته خواهد شد. در بخش ۴-۲ زبان میانی بکار گرفته‌شده در پژوهش توصیف خواهد شد. در بخش ۴-۳ فرآیند مرحله اول تشریح خواهد شد. در بخش ۴-۴ بازی و مکانیک‌های تشریح می‌شود. سپس در بخش ۴-۵ به تشریح مرحله دوم فرآیند به‌منظور تولید چیدمان عنصرهای بازی پرداخته خواهد شد و توضیحی بر نحوه اعمال سیستم‌های نوع مختلف در فرآیند تبدیل داده خواهد شد.

۴-۲- زبان میانی

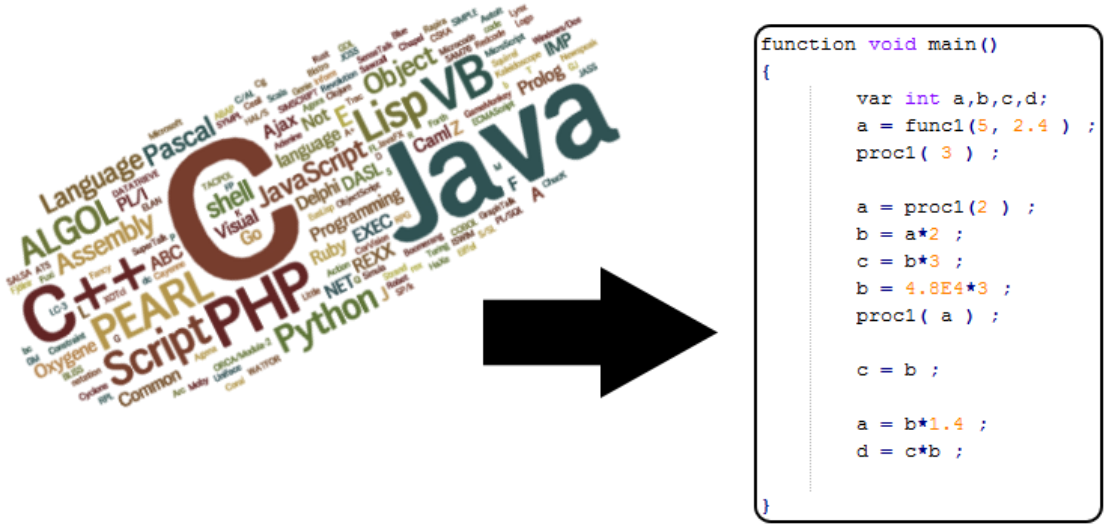
سیستم فعلی بر اساس یک‌زبان میانی کار می‌کند و همان‌طور که در مقدمه اشاره شد، این کد به‌عنوان ورودی سیستم دریافت می‌گردد. یکی از کارهای جدید ارائه‌شده در این پژوهش از بین بردن وجود محدودیت در زبان کد ورودی است. بدین معنا که قطعه کدی که قصد آزمودن آن وجود دارد، مستقل از زبان آن می‌تواند به‌عنوان ورودی سیستم داده شود.

هرچند که لازم به ذکراست که تابعی[1] و یا شی‌گرایی[2] بودن زبان تنها محدودیت فعلی خواهد بود که این محدودیت در قسمت کارهای آینده جا گرفته است. در ادامه به دلایل استفاده از یک کد میانی در فرآیند

[1] Procedural

[2] Object Oriented (OO)

درســـتی‌یابی ارائه‌شـــده این پژوهش و اهمیت آن خواهیم پرداخت. همچنین گرامر ارائه‌شـــده برای این زبان تشریح خواهد شد.



شکل (۴-۲) در مرحله صفر، کد باید به زبان میانی تبدیل گردد

۴-۲-۲- هدف از زبان میانی

سـؤال اولی که می‌تواند پرسیده شود، دلیل و چرایی لزوم به‌کارگیری یک‌زبان میانی در این پژوهش است. ما در این قسمت سعی در توضیح این مسئله خواهیم داشت.

همان‌طور که در فصـل کارهای مرتبط اشـاره گشـت، کار انجام‌شـده توسـط مرجع [6] که قوی‌ترین کار انجام‌شـده در این حوزه اسـت، بر مبنای زبان برنامه‌نویسی جاوا ارائه گشته است که یک محدودیت محسوب می‌گردد. یکی از اهداف این پژوهش از بین بردن این محدودیت و گسترش دامنه کاربرد این شیوه برای سایر زبان‌ها است.

در کنـار مورداشـــاره شـــد، دلایـل دیگری نیز وجود دارد. ازجمله این دلایل عدم نیاز به گســـتره‌ای از قسـمت‌های کد اصـلی اسـت. قسـمت‌هایی که در حوزه درسـتی‌یابی این پژوهش کاربردی نخواهند داشـت. همان‌طور که در ابتدای گزارش اشـاره شـد، در این پژوهش تمرکز اصـلی بر درسـتی‌یابی ایستا و در محدوده سـیستم‌های نوع موجود در زبان‌های برنامه‌سازی است. به همین دلیل مسیرهای اجرایی در برنامه که توسط دسـتورات شـرطی[1]، حلقه‌ها[2] و غیره ایجاد گشـته اسـت، درروند درسـتی‌یابی این پژوهش بی‌تأثیر بوده و

[1] Conditional Statements

[2] Loops

درنتیجه قابل‌چشم‌پوشی است. درنتیجه، یکی از ویژگی‌های زبان میانی حذف این دسته از دستورات که در ادامه روند فعلی بی‌تأثیر هستند، است.

در این قسمت ویژگی‌های زبان میانی ارائه‌شده در این پژوهش اشاره می‌گردند:

۱. **یکپارچه‌سازی سیستم برای تمام زبان‌ها**:

ارائه یک راه‌حل باقابلیت پشتیبانی از گستره‌ی وسیعی از زبان‌های فعلی بسیار ارزشمند بوده و کمک می‌کند تا تمام تکنیک‌های پیاده‌سازی شده در مراحل مختلف سیستم، تنها یک‌بار پیاده‌سازی شوند و در طول زمان و با آمدن زبان‌های جدید، نیازمند به‌روزرسانی و نگهداری نباشند[14].

البته این کار مستلزم آن است که در تمامی مراحل سیستم، کار به‌صورت به‌گونه‌ای طراحی شود که این ویژگی را در خود داشته باشد و در این پژوهش این مهم برآورده شده است. در سیستم ارائه‌شده، طراحی و پیاده‌سازی تمامی مراحل بر مبنای استفاده در تمام زبان‌ها و با شناخت ویژگی‌های قابل‌استخراج از سایر زبان‌ها انجام‌شده است.

نتیجه این کار ازاین‌رو بسیار ارزشمند خواهد بود که این پژوهش باهدف کاربردی آغاز گشته است و این ویژگی از پروژه امکان بهره بری از پروژه رو چندین برابر افزایش می‌دهد و راه را برای استفاده در صنایع مختلف که تحت زبان‌های خاص خود انجام می‌دهند، باز می‌گذارد.

به‌عنوان‌مثال در فعالیت‌های حوزه صنایع پیشرفته زبان‌های سریع و کارا مانند زبان c++ کاربرد خواهند داشت و در حوزه وب زبان‌هایی مانند پایتون[1] کاربرد بیشتری خواهند داشت. درنتیجه در این پژوهش سعی شده است که با ارائه یک‌زبان میانی این امکان برای توسعه‌دهندگان در زبان‌های برنامه‌نویسی مختلف وجود داشته باشد که با تبدیل زبان مورداستفاده خود به این زبان میانی از امکانات این پژوهش بهره کامل را ببرند.

۲. **حذف دستورات کنترلی**:

مسیرهای اجرایی در برنامه از مواردی هستند که در درستی‌یابی های ایستا معمولاً بی‌تأثیر است و سیستم موظف است مستقل از مسیرها، تمامی کد را بررسی و تحلیل نماید. ازاین‌رو در کد میانی به حذف این دسته از دستورات پرداخته‌شده است. این نکته لازم به ذکر است که در بخش کارهای آینده به اهمیت این دستورات و چگونگی استفاده از آن‌ها در ادامه مسیر این پژوهش اشاره‌شده است.

۳. **حذف دستورات زائد**:

یکی دیگر از انگیزه‌های وجود زبان میانی، ساده‌سازی پیچیدگی‌های غیر مربوط زبان‌های برنامه‌سازی

[1] python

موجود است. این کار نه‌تنها به فرآیند توسعه سیستم جاری کمک کرده است، بلکه پیچیدگی‌های غیرضروری را نیز از سیستم حذف نموده است.

۴. **ساده‌سازی توسعه سیستم برای سایر سیستم‌های نوع**:

سیستم فعلی یک سیستم زنده است که در آینده توانایی ارتقا برای پشتیبانی از سیستم‌های نوع گوناگون را خواهد داشت. درنتیجه باید بتوان سیستم را برای توسعه و گسترش به تیم های اجرایی جدید آموزش داد. حال اگر سیستمی با پیچیدگی بالا در دست باشد، این عمل بسیار هزینه‌بر خواهد بود و در طول زمان باعث کنار گذاشتن آن خواهد شد. ازاین‌رو تلاش شد تا معماری نرم‌افزاری به‌دوراز هرگونه پیچیدگی باشد و بدین منظور نیاز بود تا پیچیدگی‌های ورودی‌های سیستم نیز تا جایی که به هدف و کارایی آن ضربه‌ای وارد نکند، حذف بگردد.

۵. **پشتیبانی از زبان‌ها به‌صورت توزیع‌شده**:

جداسازی منطق ورودی سیستم از زبان باعث می‌شود تا بتوان فرآیند استخراج اطلاعات از سیستم خارج گشته و در انحصار قالب سیستم نباشد. این به این معنی خواهد بود که پشتیبانی از یک‌زبان جدید در انحصار توسعه‌دهندگان سیستم نباشد و بتوان پشتیبانی از زبان‌های متفاوت را به‌صورت توزیع‌شده بین تیم های مختلف تقسیم نمود.

۶. **امکان استفاده برای زبان‌های برنامه‌نویسی در آینده**:

همان‌طور که در سال جاری شاهد تولد یک‌زبان جدید[1] بودیم، نباید پشتیبانی از زبان‌های آینده رو فراموش نمود. ازاین‌رو سیستمی با ورودی واحد امکان پشتیبانی از زبان‌های آینده را مهیا خواهد نمود.

۴-۲-۳- گرامر زبان میانی

در این قسمت در مورد گرامر طراحی‌شده و ساختار آن توضیح داده خواهد شد. از این بخش می‌توان به‌عنوان راهنمایی برای سایر توسعه‌دهندگان که قصد نوشتن ابزارهای مبدل زبان موردعلاقه خود به زبان میانی این سیستم رادارند، استفاده نمود.

در ادامه ابتدا ثابت‌های موجود در گرامر به‌صورت یک جدول ارائه گشته‌اند. سپس قوانین[2] گرامر زبان میانی به‌صورت بخش‌های کوچک و مرتبط آورده شده است و هر بخش توضیح داده خواهد شد. در بخش پیوست‌ها، نیز این گرامر به‌صورت یکجا آورده شده است.

[1] زبان Swift که توسط شرکت Apple معرفی گشت

[2] Rule

◘ جدول ثابت‌های گرامر

در جدول ۴-۲ لیستی از ثابت‌های بکار رفته در گرامر قرار داده‌شده است.

جدول (٤-٢) ثابت‌های گرامر

ثابت استفاده‌شده در گرامر	معادل
S_VOID	void
S_BOOL	bool
S_INT	Int
S_FLOAT	float
S_STRING	string
S_TRUE	true
S_FALSE	false
S_TEXT	“some text”
S_RETURN	return
S_ASSERT	assert
S_TYPE	type
S_VAR	var
S_FUNCTION	function
S_MAIN	main
ID	یک نام (تشکیل‌شده از حروف بزرگ و کوچک لاتین و ’_‘)
NUM	عدد صحیح یا اعشاری

◘ بدنه اصلی گرامر (آغاز)

Program → DeclarationList
DeclarationList → Declaration DeclarationList | ε
Declaration → FunctionDeclaration | TypeDeclaration | VariableDeclaration

برنامه (Program) در این گرامر از چندین تعریف (Declaration) تشکیل‌شده است. این تعاریف می‌توانند مربوط بـه تعریف یک تابع (.../Function/Procedure)، نوع داده‌ای (.../Class/Struct) یا متغیر (Variable) باشند.

◘ تعریف نوع داده‌ای

TypeDeclaration → TYPE ID OptionalSuperclass { MemberDeclarationList };
OptionalSuperclass → : ID |
MemberDeclarationList → MemberDeclaration MemberDeclarationList | ε
MemberDeclaration → VariableDeclaration | FunctionDeclaration

قـالـب تعریف نوع‌هـای داده‌ای مشـــابـه تعریف کلاس‌هـا در زبـان C++ بوده. در ابتدا با کلمه‌ی کلیدی

"Type" شروع می‌شود و در ادامه نام نوع داده‌ای (ID) قرار می‌گیرد.

همچنین در تعریف نوع داده‌ای در زبان میانی قابلیت ارث‌بری[1] که در اکثر زبان‌های پیشرفته وجود دارد، پشتیبانی می‌گردد. این کار در قانون OptionalSuperclass گرامر فوق نمایش داده‌شده است.

در بدنه تعریف یک نوع داده‌ای ما با مجموعه‌ای از تعریف متغیرها و توابع روبرو خواهیم بود که در قالب MemberDeclaration ارائه‌شده است.

◘ تعریف توابع

FunctionDeclaration → FUNC FunctionType ID (OptionalParameterList) Block
FunctionType → void | TypeName
OptionalParameterList → Parameter MoreParameters | ε
Parameter → TypeName Variable
MoreParameters → , Parameter MoreParameters | ε
Block → { DeclarationOrStatementList }
DeclarationOrStatementList → DeclarationOrStatement DeclarationOrStatementList | ε

برای تعریف یک تابع ابتدا از کلمه کلیدی "Function" استفاده می‌کنیم. در ادامه ابتدا نوع خروجی و سپس نامی برای تابع می‌آید. پس‌ازآن لیستی از پارامترهای ورودی ظاهر می‌گردد و در انتها بدنه تابع توسط Block مشخص می‌گردد.

تابع می‌تواند هیچ خروجی‌ای نداشته و نوع خروجی آن void باشد. همچنین تابع می‌تواند هیچ پارامتر ورودی نداشته باشد.

◘ تعریف متغیر

TypeName → PrimitiveType | ID
PrimitiveType → bool | int | float | string
VariableDeclaration → VAR TypeName RestOfVariableDeclaration
RestOfVariableDeclaration → Variable MoreVariables;
OptionalPointer → * | ε
Dimensions → [NUM] Dimensions | ε
InitialValue → ID() | NUM | S_TRUE | S_FALSE | S_TEXT
OptionalInitial → = InitialValue | ε
Variable → OptionalPointer ID Dimensions OptionalInitial
MoreVariables → ,Variable MoreVariables | ε

برای تعریف یک متغیر جدید، پس از کلمه کلیدی "Var" نوع داده‌ای موردنظر در قالب TypeName

[1] Inheritance

می‌آید و سپس لیستی از نام متغیرها (RestOfVariableDeclaration) ظاهر می‌گردد.

هر متغیر (Variable) می‌تواند اشاره‌گر بودن خود را با قانون OptionalPointer مشخص کند. همچنین قانون Dimensions برای ایجاد ارائه‌ها بکار می‌رود. همچنین در پایان در صورت لزوم می‌توان مقداردهی اولیه برای متغیرها در نظر گرفت.

◘ عبارات[1]

MoreExpressions → , Expression MoreExpressions | ε
OptionalExpression → Expression | ε
Expression → Primary MoreOperand
MoreOperand → * Primary | ε

Primary → ID MorePrimary | NUM | TRUE | FALSE | TEXT
MorePrimary → ArrayAccess | FieldAccess | FunctionCall | ε

ArrayAccess → [Expression] MorePrimary
FieldAccess → .ID MorePrimary
FunctionCall → (OptionalArgumentList) MorePrimary
OptionalArgumentList → Expression MoreExpressions | ε

در این بخش گرامر عبارات یا Expression معرفی می‌گردد. یک عبارت حداقل از یک بخش اصلی (Primary) تشکیل‌شده است. در صورت لزوم برای نمایش تقابل چند عبارت می‌توان از قانون MoreOperand کمک گرفت.

هر عبارت اصلی ممکن است یک لفظ[2] باشد و یا ممکن است حاصل یک نام و در ادامه یکی از قوانین MorePrimary باشد. قوانین MorePrimary برای مشخص شدن موارد زیر استفاده می‌گردند:

۱. دسترسی به خانه‌ای در آرایه (ArrayAccess)
۲. دسترسی به عضو داخلی یک نوع داده‌ای تعریف‌شده (FieldAccess)
۳. فراخوانی یک تابع (FunctionCall)

◘ دستورات اصلی[3]

DeclarationOrStatement → VariableDeclaration | KeywordStatement | OtherStatement

[1] Expressions

[2] Literal

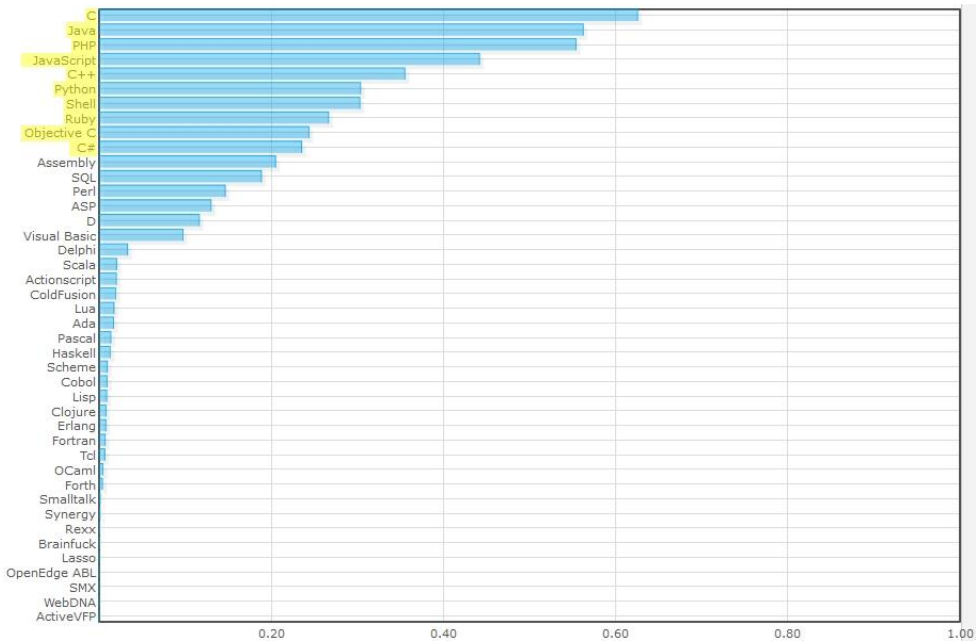
[3] Statemnets

KeywordStatement → Jump | Asserts
Jump → S_RETURN OptionalExpression;
Asserts → DataTypeAssert
DataTypeAssert → S_ASSERT (ID, TypeName);
OtherStatement → ID RestOfExpressionStatement
RestOfExpressionStatement → OptionalAssignment ;
OptionalAssignment → = Expression | ε

بدنه تمام توابع از مجموعه‌ای دستورات اصلی تشکیل می‌گردد. این دستورات به سه دسته اصلی تقسیم می‌گردند. دسته اول به‌منظور تعریف متغیر محلی استفاده می‌گردد. دسته دوم برای دستوارتی که با کلمه کلیدی آغاز می‌گردند (مانند پرش، ادعایی و بازگرداندن مقدار) استفاده می‌گردد. دسته سوم سایر کاربردها مانند انتساب یک مقدار به یک متغیر یا فراخوانی یک تابع استفاده می‌گردد.

◘ جمع‌بندی گرامر

همان‌طور که مشاهده شد گرامر فوق پشتیبانی نسبتاً کاملی از ویژگی‌های موجود در زبان‌های تابعی و شی‌گرای پرکاربرد روز دنیا را خواهد داشت. طبق آمارهای موجود در سایت محبوبیت زبان‌ها [15]، پرکاربردترین زبان‌های برنامه‌نویسی، در شکل ۴-۳ آورده شده است:

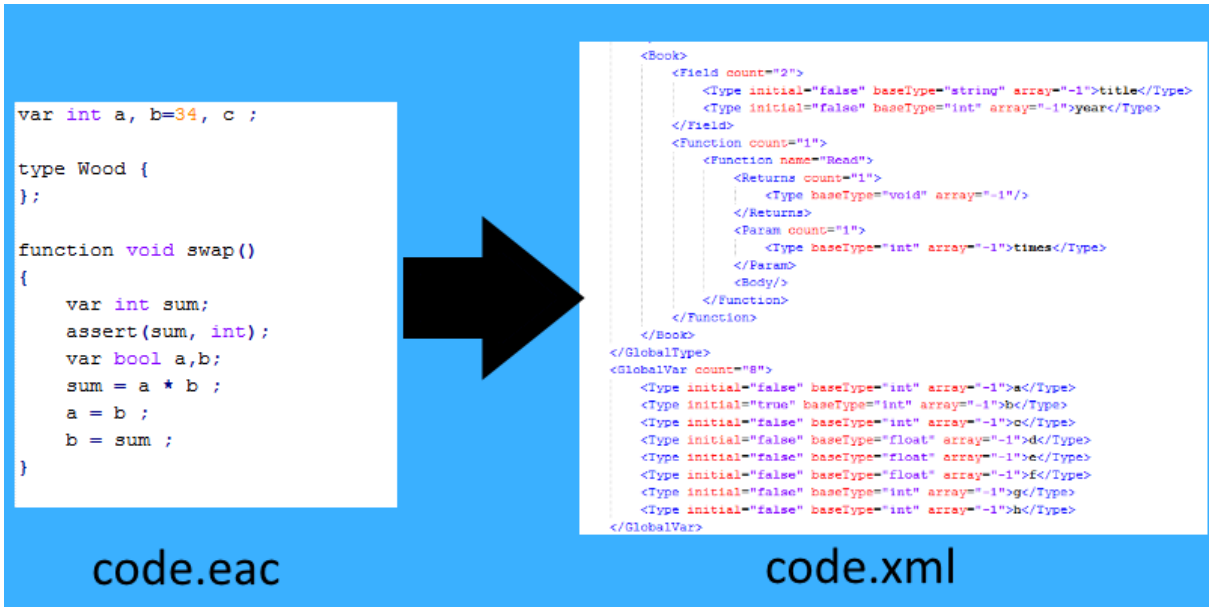


شکل (۴-۳) میزان محبوبیت زبان‌های مختلف برنامه‌سازی

همان‌طور که مشاهده می‌شود تمامی ۱۰ زبان برتر (مشخص‌شده بارنگ زرد) از یکی از دو نوع تابعی و یا شیءگرا تبعیت می‌کنند که آمار بسیار خوبی برای پژوهش فعلی است.

۴-۳- مرحله اول: تحلیل کد میانی

در این بخش به بیان نحوه استخراج اطلاعات این کد میانی پرداخته می‌شود. در اولین گام در چرخه فرآیند درستی‌یابی، کد میانی دریافت شده (به‌عنوان یکی از دو ورودی نرم‌افزار) تحلیل می‌گردد. در ادامه این بخش به علت و نحوه استخراج اطلاعات از تحلیل کد زبان میانی پرداخته می‌شود و به ساختار اطلاعات خارج‌شده که در قالب یک فایل XML است، پرداخته می‌شود.



شکل (۴-۴) هدف از مرحله اول، استخراج اطلاعات از کد ورودی است

در این مرحله کد به کمک ابزارهای پارس مانند lex و yacc پردازش‌شده و اطلاعات مختلفی از قبیل انواع نوع‌های داده‌ای استفاده‌شده در کد، تعداد توابع و نوع پارامترهای ورودی و خروجی و غیره از کد خارج می‌شود. این کار باوجود گرامر ارائه‌شده در بخش ۱-۲-۲ صورت می‌پذیرد.

۴-۳-۲- لزوم این مرحله

سؤالی که در این بخش ممکن است مطرح باشد، چرایی نیاز به این مرحله است. به زبان ساده‌تر چرا نیاز داریم کد میانی را ابتدا تحلیل کرده و سپس در مرحله‌ای جدا مراحل بازی رو تولید کنیم.

برای توضیح علت یادآوری این مورد لازم است که فرآیند درستی‌یابی یک قطعه کد نمی‌تواند یکجا انجام بپذیرد. این بدان معنا است که به ازای هر مؤلفه یا سیستم نوع ورودی، قطعه کد ورودی بر اساس طراحی‌های انجام‌شده در مرحله تولید مرحله بازی (مرحله دوم)، به اشکال مختلف به بازی تبدیل می‌گردد و چیدمان‌های متفاوتی از عنصرهای موجود در بازی تولید خواهد شد. با این مقدمه علت وجود مرحله تحلیل در چرخه درستی‌یابی بدین شرح است:

۱. فرآیند تحلیل معمولاً نیازمند پارس چندمرحله‌ای[1] کد است که این فرآیند هزینه زمانی و پردازشی خواهد داشت. از سوی دیگر درستی‌یابی یک قطعه کد به ازای سیستم‌های نوع مختلف صورت می‌پذیرد. حال اگر مرحله تحلیل را مستقل از سیستم نوع موردنظر در درستی‌یابی، به شکل جداگانه و در یک مرحله مستقل ارائه داده شود، می‌توان این مرحله از درستی‌یابی (تحلیل کد) را به ازای تمام سیستم‌های نوع فاکتور گرفته و میزان پردازش‌ها را در فرآیند درستی‌یابی کد به ازای سیستم‌های نوع مختلف تا حد بسیار زیادی از بین ببریم. درنتیجه می‌توان زمان لازم برای فرآیند درستی‌یابی را درمجموع به‌مراتب کاهش داد.

۲. در تمام سیستم‌های نوع موجود، نیاز به یک مرحله مشترک برای استخراج مجموعه‌ای از اطلاعات (مانند آنچه پیش‌تر بیان شد) در قالب جداول نماد وجود دارد. درنتیجه وجود یک مرحله مستقل و مشترک منطقی خواهد بود.

۳. توسعه سیستم‌های نوع جدید برای این سیستم امری محتمل در آینده خواهد بود. از طرفی تحلیل یک کد با گرامر مختص خود، امری تخصصی و زمان‌بر خواهد بود که نیازمند استفاده از ابزارهای تخصصی نوشته‌شده در این حوزه از قبیل lex و Yacc و غیره خواهد بود. پیچیدگی این فرآیند باعث شد تا در این پژوهش ابزاری برای ذخیره‌سازی اطلاعات قطعه کد میانی ورودی با ساختاری استاندارد مانند XML ارائه داده شود. وجود ساختاری مانند XML برای توسعه سیستم‌های نوع مختلف یک کمک به‌حساب می‌آید و فرآیند نوشتن ابزارهای مبدل برای سیستم‌های نوع جدیدتر را سرعت می‌بخشد.

۴-۳-۳- ساختار ذخیره‌سازی اطلاعت (در قالب XML)

در این بخش به‌صورت مختصر و موردی به دلایل انتخاب XML به‌عنوان سیستم و ساختار ذخیره اطلاعات استخراج‌شده در مرحله تحلیل کد میانی اشاره می‌گردد. این موارد از منبع [16] استنتاج گشته است.

۱. استاندارد بودن فرمت ذخیره‌سازی
۲. توصیفی بودن و قالب قابل‌خواندن برای ماشین و انسان به‌صورت همزمان (برای مسائل اشکال‌زدایی)
۳. ساختار نحوی دقیق آن الگوریتم‌های تجزیه لازم را سریع و کارآمد می‌کند
۴. ساختار وراثتی مناسب برای اغلب (نه همه‌ی) انواع اسناد.

[1] Pass

۵. مستقل از سیستم‌عامل و ایمن برای تغییر در تکنولوژی.

۴-۳-۴- ساختار اطلاعات فایل code.xml

اطلاعات خارج‌شده در مرحله تحلیل در قالب چند دسته اصلی در فایل XML ذخیره می‌گردد. هر بخش وظیفه نگهداری قسمتی از اطلاعات کد میانی اصلی را دارد. این قسمت‌ها طوری طراحی‌شده‌اند که برای توسعه سیستم‌های نوع جدید در آینده به‌راحتی قابل‌استفاده باشند و تا جای ممکن هیچ‌یک از اطلاعات به‌صورت تکراری در دو جا قرار نگیرد.

بخش‌های اصلی موجود در فایل XML خروجی پس از مرحله تحلیل بدین شرح است:

۱. TypeInheritance
۲. GlobalType
۳. GlobalVar
۴. GlobalFunction

در ادامه به توضیح و ارائه یک مثال برای هر بخش پرداخته می‌شود.

◘ بخش اول: TypeInheritance

در این بخش یک گروه‌بندی از نوع‌های داده‌ای موجود در کل ارائه می‌گردد. تعریف یک «گروه از نوع‌های داده‌ای» اشاره به مجموعه‌ای از نوع‌های داده‌ای دارد که قابلیت تبدیل به یکدیگر معنادار باشد.

```
<TypeInheritance>
    <Group id="1">
        <bool size="1"/>
        <int size="2"/>
        <float size="3"/>
    </Group>
    <Group id="2">
        <string size="1"/>
    </Group>
    <Group id="3">
        <Wood size="1"/>
        <Paper size="2"/>
        <Book size="3"/>
    </Group>
</TypeInheritance>
```

شکل (۴-۵) نمونه‌ای گروه‌بندی نوع‌های داده‌ای

به‌عنوان‌مثال نوع‌های داده‌ای int و bool و float و double در زبان‌های برنامه‌نویسی برای نمایش اعداد در مقیاس‌ها و دقت‌های مختلف استفاده می‌شود. هرچند این نوع‌ها داده‌ای یکسان نبوده ولی تبدیلات معنی‌دار به یکدیگر را پشتیبانی می‌کند.

به‌عنوان‌مثال دو عدد ۰ و ۳,۴- در این نوع‌های داده‌ای به‌صورت جدول ۴-۳ معنا می‌پذیرند:

جدول (٤-٣) مثالی از نحوه تبدیلات در نوع‌های داده‌ای پایه

نوع داده‌ای	مقدار "0"	مقدار "-3.4"
bool	False	True
int	0	-3
float, double	0	-3.4

همچنین یک گروه از نوع‌های داده‌ای را می‌توان به مجموعه‌ای از نوع‌ها داده‌ای اطلاق نمود که در یک زنجیره خطی از ارث‌بری قرارگرفته باشند.

به‌عنوان‌مثال نوع‌ها داده‌ای چوب (Wood)، ورق کاغذ (Paper)، کتاب (Book) می‌توانند ۳ نوع داده‌ای باشند که تبدیلات معنی‌داری دارند و یک گروه را تشکیل می‌دهند.

این سه دسته در کد میانی مانند شکل ۴-۶ ظاهر گشته‌اند:

```
type Wood {
    var string woodName;
};

type Paper: Wood {
    var int number;
    var string text;
};

type Book:Paper{
    var string title;
    var int publishDate;
    function void Read(int times){}
};
```

شکل (۴-۶) نمونه‌ای از نوع‌های داده‌ای با ارث‌بری خطی

در جدول ۲-۲ با ذکر یک مثال منظور از تبدیلات معنادار در نوع‌های داده‌ای نشان داده شده است.

جدول (٤-٤) مثالی از نحوه تبدیلات در نوع‌های داده‌ای هم‌گروه

نوع داده‌ای	شیء "کتاب فلان"	شیء "ورق کاغذ"
Wood	هست	هست
Paper	هست	هست
Book	هست	نیست

این گروه‌ها درواقع معیاری برای تبدیلات ممکن خواهند بود و نحوه استفاده آن‌ها را در همین فصل، در بخش ۴-۴-۳ مشاهده خواهد شد.

تحلیل‌گر با بررسی نوع‌های داده‌ای در کد میانی و نحوه ارث‌بری از یکدیگر، ساختار فوق را در فایل

XML می‌سازد. در این ساختار به هر گروه یک شماره اختصاص می‌یابد. ویژگی[1] اندازه[2] در این ساختار به تعداد مراحل ارث‌بری اشاره دارد.

◘ بخش دوم: GlobalType

در این بخش اطلاعات مرتبط با تمامی نوع‌های داده‌ای تعریف‌شده در برنامه نگهداری می‌گردد. این اطلاعات به ازای هر نوع داده‌ای شامل موارد زیر است:

۱. نام نوع داده‌ای
۲. متغیرهای عضو (Fields)
۳. توابع عضو (Functions)

در بخش توابع عضو، اطلاعات مرتبط با بدنه (دستورات تشکیل‌دهنده) نیز نگهداری می‌گردد. در شکل ۴-۷ اطلاعات حاصل از قطعه کد میانی مرتبط با سه کلاس Wood، Paper و Book که در بخش قبل معرفی گشت، آورده شده است.

```
<GlobalType count="3">
    <Wood>
        <Field count="1">
            <Type initial="false" baseType="string" array="-1">woodName</Type>
        </Field>
        <Function count="0"/>
    </Wood>
    <Paper>
        <Field count="2">
            <Type initial="false" baseType="int" array="-1">number</Type>
            <Type initial="false" baseType="string" array="-1">text</Type>
        </Field>
        <Function count="0"/>
    </Paper>
    <Book>
        <Field count="2">
            <Type initial="false" baseType="string" array="-1">title</Type>
            <Type initial="false" baseType="int" array="-1">publishDate</Type>
        </Field>
        <Function count="1">
            <Function name="Read">
                <Returns count="1">
                    <Type baseType="void" array="-1"/>
                </Returns>
                <Param count="1">
                    <Type baseType="int" array="-1">times</Type>
                </Param>
                <Body/>
            </Function>
        </Function>
    </Book>
</GlobalType>
```

شکل (۴-۷) اطلاعات استخراج‌شده از سه کلاس Wood، Paper و Book

علت نام‌گذاری این بخش نیز با توجه به اطلاعات نگهداری شده در آن مشخص است.

[1] Attribute

[2] size

◘ بخش سوم: GlobalVar

در این بخش تمامی متغیرهای غیر محلی که در بدنه فایل[1] تعریف‌شده باشند آورده می‌شود. این اطلاعات در شکل ۴-۸ نمایش داده شده است.

```
<GlobalVar count="8">
    <Type initial="false" baseType="int" array="-1">a</Type>
    <Type initial="true" baseType="int" array="-1">b</Type>
    <Type initial="false" baseType="int" array="-1">c</Type>
    <Type initial="false" baseType="float" array="-1">d</Type>
    <Type initial="false" baseType="float" array="-1">e</Type>
    <Type initial="false" baseType="float" array="-1">f</Type>
    <Type initial="false" baseType="int" array="-1">g</Type>
    <Type initial="false" baseType="int" array="-1">h</Type>
</GlobalVar>
```

شکل (۴-۸) اطلاعات متغیرهای جهانی

◘ بخش چهارم: GlobalFunction

این بخش بزرگ‌ترین و اصلی‌ترین بخش فایل خروجی را تشکیل می‌دهد. چراکه در این بخش تمامی توابع بدنه اصلی کد با تمامی دستورات آن‌ها به‌صورت یکپارچه نگهداری می‌گردد.

به ازای هر تابع یک عنصر Function وجود دارد. هر تابع از سه بخش اصلی تشکیل‌شده است:

۱. Returns: در این بخش لیستی از خروجی‌ها و اطلاعات مختص به آن‌ها نگهداری می‌گردد
۲. Param: در این بخش لیستی از ورودی‌ها و اطلاعات مختص به آن‌ها نگهداری می‌گردد.
۳. Body: این بخش به‌عنوان اصلی‌ترین بخش، وظیفه نگهداری اطلاعات تمامی دستوراتی بدنه را دارد.

در جدول ۴-۵ نحوه نگهداری اطلاعات دستورات مختلف کد نشان داده‌شده است.

جدول (٤-٥) نحوه ذخیره‌سازی اطلاعات دستورات کد میانی

نام دستور اصلی	نحوه ذخیره‌سازی اطلاعات
var int a = 4, b, c;	<Type ref_lineNumber="7" initial="true" baseType="int" array="-1">a</Type> <Type ref_lineNumber="7" initial="false" baseType="int" array="-1">b</Type> <Type ref_lineNumber="7" initial="false" baseType="int" array="-1">c</Type>
var Book d = Book();	<Type ref_lineNumber="8" initial="true" baseType="Book" array="-1">d</Type>
assert(d, Book);	<Assert baseType="Book" array="-1">d</Assert>

[1] Global Scpope

a = b ;	<Assignment ref_lineNumber="10"> <Left>a</Left> <Right count="1"> <Primary source="var">b</Primary> </Right> </Assignment>
a = b * c;	<Assignment ref_lineNumber="11"> <Left>a</Left> <Right count="2"> <Primary source="var">b</Primary> <Primary source="var">c</Primary> </Right> </Assignment>
a = foo();	<Assignment ref_lineNumber="12"> <Left>a</Left> <Right count="1"> <Primary source="call"> <Call name="foo" argCount="0"/> </Primary> </Right> </Assignment>
a = foo(b, true);	<Assignment ref_lineNumber="13"> <Left>a</Left> <Right count="1"> <Primary source="call"> <Call name="foo" argCount="2"> <Primary source="var">b</Primary> <Primary source="data"> <Type baseType="bool" array="-1"/> </Primary> </Call> </Primary> </Right> </Assignment>
foo("p1");	<Assignment ref_lineNumber="14"> <Left count="0"/> <Right count="1"> <Primary source="call"> <Call name="foo" argCount="1"> <Primary source="data"> <Type baseType="string" array="-1"/> </Primary> </Call> </Primary> </Right> </Assignment>

۴-۴- بازی استفاده‌شده در پژوهش: کارخانه بزرگ

در این قسمت به بررسی بازی استفاده‌شده در فرآیند درستی‌یابی خواهیم پرداخت. در طراحی این بازی سعی شده است تا جای ممکن سادگی رعایت شود تا دامنه بازیکنان تا جای ممکن بیشتر گردد. دامنه بازیکنان زیاد برای دریافت نتایج سریع‌تر و دقیق‌تر در فرآیند درستی‌یابی بسیار ارزشمند است.

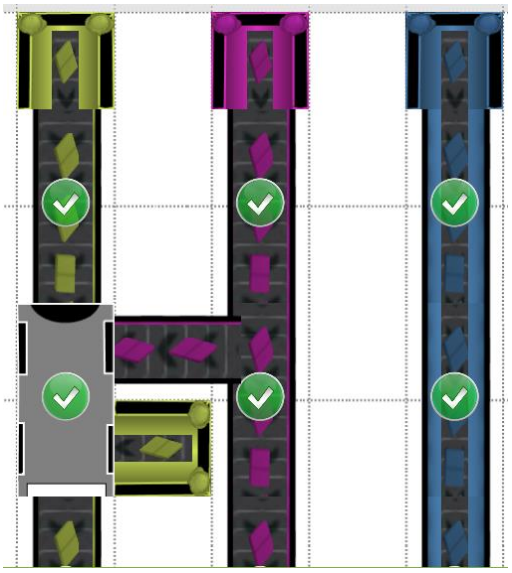
شکل ۴-۹ نمایی از بازی فوق را نمایش می‌دهد. بازی کارخانه بزرگ در این پژوهش برای درستی یابی در سیستم نوع، نوع داده‌ای استفاده شد.

شکل (۴-۹) نمایی از محیط بازی کارخانه بزرگ

در ادامه این بخش به توضیح بازی و مکانیک‌های موجود در بازی پرداخته خواهد شد.

۴-۴-۲- سناریوی بازی

در این بازی، بازیکن با نمایی از سالن یک کارخانه که شامل مجموعه‌ای از خطوط انتقال به کمک نقاله است، روبرو می‌شود. در این کارخانه هر نقاله که عموماً به‌صورت عمودی و از بالا به پایین قرارگرفته‌اند، در حال انتقال نوعی کالا با اندازه مشخص است. هر نقاله با یکرنگ متمایز در محیط ظاهرشده است که در طول مسیر عمودی، این رنگ ثابت خواهد بود.



شکل (۴-۱۰) خطوط نقاله بازی

هر نقـالـه از یک تســـمه‌نقاله برای انتقال کالاها اســـتفاده می‌کند. کالاهای قرار داده‌شـــده بر روی این تســـمه‌نقاله‌ها دارای اندازه‌های مختلف اســـت (عرض‌های مختلف). همچنین خود تســـمه‌نقاله‌ها می‌توانند عرض‌های متفاوت داشـــته باشـــند. در این بازی، بازیکن قادر به کنترل عرض تســـمه بعضـــی نقاله‌ها و بعضـــی کالاهای تولیدی است.

کـالاهـای قرارگرفته بر روی یک نقاله تنها زمانی قابل‌حمل خواهند بود که عرض تســـمه‌نقاله‌ها از اندازه آن‌ها بزرگ‌تر و یا مســـاوی باشـــد. هدف بازیکن در بازی اطمینان از عملکرد درســـت در تمام خطوط انتقال اســـت؛ یعنی تعیین اندازه تســـمه‌نقاله‌ها به‌طوری‌که تمامی نقاله‌ها امکان جابجایی کالاهای خود را داشـــته باشـــند. در اصـطلاح به این حالت، حالت «امن» در نقاله گفته می‌شود و به حالتی که اندازه کالاها بزرگ‌تر از تسمه‌نقاله‌ها باشند، حالت «ناامن» گفته می‌شود.

شکل (۴-۱۱) دو خط نقاله امن و ناامن

بر روی بدنه خطوط نقاله و در فواصـل مشـخص علائمی وجود دارد که وضـعیت فعلی آن خط نقاله را به

بازیکن نمایش می‌دهد:

۱. نماد خط نقاله «امن»:

این بدان معناست که کالای عبوری از این قسمت از خط نقاله کوچکتر-مساوی اندازه تسمه‌نقاله آن است.

شکل (۴-۱۲) نماد امن

۲. نماد خط نقاله «ناامن»:

این نماد بدان معناست که کالای عبوری از این قسمت از خط نقاله بزرگ‌تر از تسمه‌نقاله آن است

شکل (۴-۱۳) نماد ناامن

یک بازیکن ممکن است تلاش کند تا عرض تمام تسمه‌نقاله‌ها را در بزرگ‌ترین حالت قرار دهد، اما این کار امکان‌پذیر نخواهد بود چراکه بر روی بعضی خطوط نقاله محدودیت وجود دارد. این محدودیت با یک نماد قفل () بر روی آن خط نقاله نمایش داده‌شده است (شکل شماره۴-۱۴).

شکل (۴-۱۴) عدم امکان بزرگ شدن تسمه‌نقاله و حالت ناامنی

با توجه به امکان وجود محدودیت فوق بر روی خطوط نقاله، بازیکن باید تلاش کند تا با استفاده از مکانیک‌های بازی (تعویض عرض تسمه و استفاده از «تبدیل‌کننده جادویی» که در ادامه شرح داده خواهد

شد) با بیشترین امتیاز ممکن تمام خطوط انتقال را به‌طور همزمان در حالتی امن قرار دهد.

ایده بازی ارائه‌شده در این بخش متفاوت از کارهای مشابه بوده و سعی شده است در عین حفظ سادگی عنصرهای بازی، امکان گنجاندن مفاهیم مختلف از یک‌زبان برنامه‌نویسی را دارا باشد.

در اینجا می‌توان تعابیر گوناگون از عنصرهای توضیح داده‌شده برداشت نمود. هر برداشت می‌تواند بسته به سیستم نوع مورداستفاده در درستی‌یابی دستخوش تفاوت‌هایی باشد، ولی نکته ارزشمند در یکسان بودن حاصل تبدیل مفاهیم موجود در تمام سیستم‌های نوع به مجموعه‌ای محدود و ثابت از عنصرهای موجود در بازی است. در بخش ۴-۵ چند مثال برای روشن شدن این ارتباط بیان‌شده است.

همچنین یکی دیگر از مواردی که پیش‌تر اشاره گشت، سادگی بازی است. این سادگی که به‌واسطه‌ی محدود بودن تعداد و نوع مکانیک‌های بازی حاصل‌شده است، از دیگر نکات حائز اهمیت در طراحی بازی است. هیچ‌یک از مکانیک‌های موجود در بازی تابع زمان نبوده و روند بازی نیازمند فعالیت فکری و استنتاجی بازیکن است تا اینکه به‌سرعت عمل بازیکن نیاز داشته باشد که باعث می‌شود دامنه بازیکنان محدود به بازکنان حرفه‌ای نباشد. پس در اینجا مؤلفه سرعت نقشی درروند بازی ندارد.

در ادامه به سایر مکانیک‌های موجود در بازی پرداخته می‌شود. این مکانیک‌ها در کنار نقش فنی خود، به هرچه جذاب‌تر شدن بازی نیز کمک خواهند کرد:

◘ مرحله (جورچین)

در بازی طراحی‌شده، منظور از یک مرحله یا جورچین، یک سالن شامل مجموعه‌ای از خطوط نقاله خواهد بود. ازنقطه‌نظر کد میانی، یک مرحله معادل بدنه یک تابع سراسری خواهد بود.

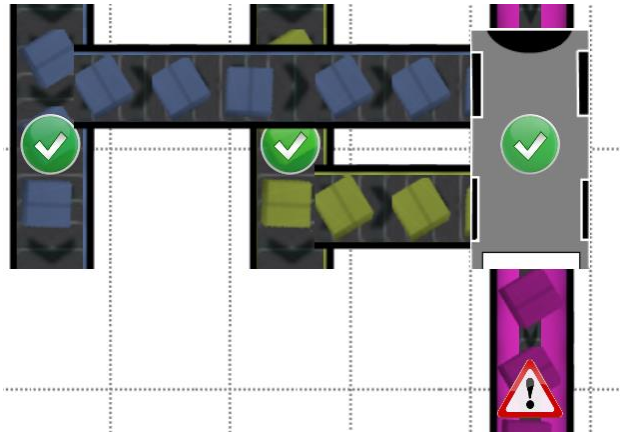
◘ اتصالات

در قسمت‌هایی از محیط کارخانه، خطوط نقاله افقی مشاهده می‌شود. این خطوط وظیفه انتقال کالاهای موجود بر روی یک خط به خط دیگر را دارد. در جریان این انتقال، اتصالات پایانی به‌واسطه یک ابزار بنام «ادغام کنند[1]» صورت می‌پذیرد. وظیفه ادغام کننده مسدودسازی خط نقاله مقصد از انتقال کالاهای موجود بر روی آن و انتقال کالاهای جدید موجود در خطوط نقاله افقی واردشده به ادغام کننده به خط نقاله مقصد است.

نکته مهمی که ادغام کننده درگیر آن خواهد بود، اندازه کالای خروجی از آن است که برابر با اندازه

[1] Merger

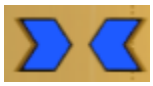
بزرگ‌ترین کالاهای ورودی (از خطوط نقاله افقی) است. شکل ۴-۱۵ یک ادغام کننده را نشان می‌دهد.



شکل (۴-۱۵) سمت راست تصویر: ادغام کننده

◘ تبدیل‌کننده جادویی

تمام مراحل بازی قابل‌حل نیستند. پس ازآنجایی‌که تمام مراحل قابل‌حل نخواهند بود، باید تکنیکی برای گزارش موضوع عدم امکان اثبات درستی در بازی وجود داشته باشد. درواقع به دلیل وجود محدودیت در عرض بعضی از تسمه‌نقاله‌ها، بعضی مراحل به ازای هیچ جایگشتی از انتخاب عرض تسمه‌نقاله‌ها و یا کالاها قابل‌حل نخواهند بود. در این مواقع از بازی، بازیکن باید از وسیله‌ای بنام «تبدیل‌کننده جادویی» استفاده کند.



شکل (۴-۱۶) تبدیل‌کننده جادویی

با قرار دادن تبدیل‌کننده‌های جادویی بر روی یک نقاله می‌توان طول کالاهای عبوری از آن قسمت از خط نقاله را آزادانه کنترل نمود. این کار کمک می‌کند که کالاهای عبوری بزرگ بتوانند پس از تغییر اندازه به‌صورت امن از خطوط نقاله با محدودیت اندازه در عرض تسمه‌نقاله، عبور کنند.

بازیکن می‌تواند بدون هیچ‌گونه محدودیتی در تعداد استفاده، از این تبدیل‌کننده استفاده کند. بازیکن می‌تواند با استفاده مکرر از این تبدیل‌کننده تمام مراحل را با کمترین فکر حل کند. ولی استفاده از تبدیل‌کننده جادویی به ازای از دست دادن بخشی زیادی از امتیاز بازیکن خواهد بود. بدین‌جهت بازیکنان سعی می‌کنند تنها در شرایطی که هیچ راهی وجود ندارد به این وسیله روی بیاورند و استفاده حداقلی را داشته باشند.

دوربین بازی

با توجه به متغیر بودن حجم توابع در کد مبدأ، اندازه نقشـــه بازی (کارخانه) می‌تواند متغیر بوده. هرچند در یک کد «خوب» ازنقطه‌نظر رابرت مارتین[1] نویسنده کتاب «کد تمیز[2]»، توابع نباید بیشتر از ۵ خط باشند، اما عدم رعایت این قانون توسط برنامه‌نویسان امری رایج است. ازاین‌رو برای ایجاد یک کنترل راحت برای بازیکن در نقشه‌های وسیع سیستم جابجایی دست[3] و ابزارهای بزرگنمایی و کوچک نمایی قرار داده‌شده است.

شکل (۴-۱۷) ابزارهای کنترل دوربین بازی

سیستم امتیازدهی

یک مرحله ممکن اسـت چند راه‌حل داشـته باشـد و بعضـی راه‌حل‌های مورد انتظار سیستم نیست. درنتیجه باید سیستمی برای ارزش‌گذاری راه‌حل‌های مختلف وجود داشته باشد.

عواملی که روی میزان امتیاز بازیکن تأثیرگذارند، شامل موارد زیر است:

۱. تعداد خطوط نقاله «ناامن»
۲. تعداد استفاده از تبدیل‌کننده جادویی
۳. عرض خطوط نقاله (عرض کمتر، امتیاز بیشتر)

همـان‌طور که مشـــاهده می‌شـــود معیار زمان در این بازی هیچ امتیازی را در برندارد. علت این کار عدم القای حس عجله در بازیکن اسـت. مطلوب این خواهد بود که بازیکن با دقت و با صـرف زمان کافی، سعی در کسب بالاترین امتیاز داشته باشد.

[1] Robert Cecil Martin

[2] The Clean Coder: A Code of Conduct for Professional Programmers

[3] Panning

شکل (۴-۱۸) سیستم امتیازدهی بازی

ویژگی تابعی و شـــی‌گرایی موجود در کد ورودی کمک می‌کند که بازیکن پس از انجام یک مرحله بتواند بازی را کنار بگذارد و بتواند بعدتر دوباره به ادامه بازی برگرد.

۴-۴-۳- ارتباط عنصرهای بازی با کد میانی بر اساس سیستم های نوع

در این بخش سـعی می‌شـود برای روشـن شدن نحوه ارتباط کد میانی و عنصرهای بازی کارخانه بزرگ، این کار به ازای چندین سیستم نوع مختلف انجام گردد.

بدین منظور ابتدا چهار سـیسـتم نوع منتخب به‌صـورت مختصـر توضیح داده خواهند شد و سپس با ارائه یک مثال به نحوه ارتباط عنصرهای بازی به ازای هر سیستم نوع پرداخته خواهد شد.

◘ سیستم نوع اول: نوع داده‌ای[۱]

این سـیسـتم نوع درواقع رایج‌ترین سـیستم نوع در بین تمام زبان‌های برنامه‌نویسی است. این سیستم نوع به زبان ساده همان نوع‌های داده‌ای انتساب داده‌شده به متغیرها است.

به‌عنوان‌مثال در کد زیر نوع داده‌ای «عدد اعشاری» به متغیر age نسبت داده‌شده است.

```
float age = 23.5 ;
```

در این سـیستم نوع هر متغیر یک نوع داده‌ای مشخص خواهد داشت که اجازه می‌دهد اطلاعاتی از همان جنس را نگهداری کند.

سـطوح توصـیف‌کننده در این سـیسـتم نوع می‌تواند مانند تعداد نوع‌های داده‌ای موجود در یک برنامه، نـامحـدود باشـد. در این پژوهش مفهومی به نام گروه‌های نوع‌های داده‌ای معرفی گردید (به بخش ۴-۳-۴ سـاختار اطلاعات فایل code.xml رجوع گردد). با این مفهوم می‌توان سـطوح سـیسـتم نوع، نوع داده‌ای را نامحدود تلقی نکرده و تعداد این سطوح را متناظر با تعداد اعضای «گروه‌های نوع‌های داده‌ای» دانست.

در بازی کارخانه بزرگ سطوح مختلف نوع داده‌ای ابتدا به مؤلفه اندازه در گروه نوع‌های داده‌ای تبدیل‌شده و درنهایت برابر با اندازه عرض تسمه‌نقاله‌ها و کالاهای عبوری از آن‌ها می‌گردد.

◘ سیستم نوع دوم: اشاره‌گر تهی[۲]

در این سیستم نوع، برنامه‌نویس برای هر متغیر یک نوع مشخص می‌کند:

۱. متغیر حتماً مقدار خواهد داشت، یا

---

۱ Data Type Type System

۲ Nullness Type System

۲. متغیر می‌تواند تهی نیز باشد.

به زبان ساده‌تر این سیستم نوع مشخص می‌کند که یک متغیر خاص امکان پذیرفتن مقدار تهی[۱] را دارد یا خیر. علت وجود این سیستم نوع خطای رایج اشاره‌گرهای تهی[۲] است. به‌عنوان یک مثال کد زیر را در نظر بگیرید:

```
Object o = null;
String s = o.toString(); ===> Throw NullPointerException
```

برای جلوگیری ازاین‌دست خطاها بعضی زبان‌ها (مانند جاوا) سیستم نوع اشاره‌گرهای تهی را معرفی کرده‌اند. برای این منظور می‌توان یک متغیر را از نوع مقداردهی شده[۳] معرفی نمود:

```
Object o = null;
@ NonNull String s = o.toString(); ===> Just Warning!
```

در بازی کارخانه بزرگ می‌توان متغیرهای غیرقابل تهی (متغیرهایی که مشخصاً به این نوع نسبت داده‌شده‌اند و یا دسترسی به توابع/متغیرهای عضو آن‌ها شده) را با اندازه باریک نشان داد (اندازه یک) و سایر را با اندازه‌های عریض (اندازه بزرگ‌تر از یک).

◘ سیستم نوع سوم: رمزنگاری[۴]

این سیستم نوع در نرم‌افزارهای تحت شبکه کاربرد ویژه‌ای دارد. در این سیستم دو نوع داده زیر وجود دارد:

۱) معمولی[۵]

۲) رمزنگاری

در این سیستم متغیرهایی که از نوع رمزنگاری باشند با تسمه‌نقاله‌های عریض (اندازه بزرگ‌تر از یک) و متغیرهایی که از نوع معمولی باشند با تسمه‌نقاله‌های باریک (اندازه یک) نمایش داده می‌شوند.

◘ سیستم نوع چهارم: سطح امنیت[۶]

این سیستم مشابه سیستم نوع پیشین بوده. این سیستم نوع در نرم‌افزارهای نظامی و امنیتی کاربرد ویژه‌ای دارد. در این سیستم تعداد نامحدودی توصیف‌گر وجود دارد. به‌عنوان‌مثال در یک پروژه می‌توان سطح‌های امنیتی یک تا ده را تعریف کرد. در این سیستم اطلاعات یک متغیر با درجه بالاتر قابل‌انتقال به متغیر با

---

۱ Null

۲ NullPointerException

۳ NonNull

۴ Encryption TypeSystem

5 Clear

۶ Security Type System

درجه اهمیت پایین‌تر نیست.

برای استفاده از این سیستم نوه در بازی کارخانه بزرگ کافی است تا به متغیرهای با درجه اهمیت بالاتر، تسمه‌نقاله‌های با عرض بیشتر نسبت داده شود. در این حالت اطلاعات با درجه اهمیت بالاتر (کالاهای بزرگ‌تر) امکان انتقال به تسمه‌نقاله‌های کوچک‌تر که نماینده متغیرهای با درجه اهمیت پایین‌تر هستند را دارا نمی‌باشند.

۴-۵- مرحله دوم: تبدیل کد به مراحل بازی و نحوه نگاشت سیستم‌های نوع

پیاده‌سازی درستی‌یابی بر اساس هر سیستم نوع نیازمند سه مرحله اساسی زیر است:

۱. تعریف نگاشت از کد میانی به عنصرهای بازی (معرفی‌شده در بخش قبل)
۲. نوشتن مبدل از اطلاعات استخراج‌شده به یک چیدمان از عنصرهای بازی (مرحله دوم سیستم)
۳. نوشتن تکنیکی برای گزارش درستی یا خطای احتمالی در کد بر اساس رویدادهای درون بازی

در این پژوهش مراحل فوق برای سیستم نوع، نوع داده‌ای انجام گشته است. علت انتخاب این سیستم نوع دو مورد زیر است:

۱. نگاشت راحت و معنی‌دار این سیستم نوع که در بخش قبل به تفضیل شرح داده شد.
۲. نمایش قابلیت بیش از دو سطح توصیفگر در بازی.

حال که سیستم نوع استفاده‌شده در این پژوهش و علت انتخاب آن شرح داده شد، در ادامه به توضیح نحوه نگاشت این سیستم نوع با عنصرهای بازی و نحوه تشکیل فایل چیدمان مراحل (level.xml) پرداخته خواهد شد.

۴-۵-۱- نحوه نگاشت کد به عنصرهای بازی

در جدول ۴-۶ نحوه نگاشت کد میانی به عنصرهای موجود در بازی ارائه گشته است. در این پژوهش اکثر موارد زیر به‌طور دقیق پیاده‌سازی گشته است و قسمتی کوچک باقیمانده در کارهای آینده آمده است.

جدول (٤-٦) تناظر کد میانی و عنصرهای بازی

کد میانی			عنصر بازی
تعریف تابع سراسری	تعریف تابع		یک سالن (مرحله)
	متغیرهای ورودی		خطوط نقاله واردشده به سالن
	متغیرهای خروجی		خطوط نقاله خارج‌شده از سالن
تعریف متغیر	محلی		یک خط نقاله جدید در سالن
	سراسری		خط نقاله‌ای که در تمامی سالن‌های قرار دارد

انتسابات	T = B * C * D	انتقال کالای خطوط انتقال A و B و غیره به T
	T = Foo(a1, a2, ...)	انتقال کالای خطوط انتقال a1 و a2 و غیره به سالن Foo و انتقال کالای خط نقاله خارج‌شده از سالن به خط نقاله T
	Foo(a1, a2, ...)	انتقال کالای خطوط انتقال a1 و a2 و غیره به سالن Foo
Assert(A, int)		محدود شدن خط نقاله A به عرض خاص

۴-۵-۲- ساختار فایل چیدمان مراحل (level.xml)

در این بخش به بررسی ساختار فایل چیدمان مراحل بازی و نحوه تولید آن در سیستم نوع موردبررسی این پژوهش (نوع داده‌ای) پرداخته می‌شود.

فایل level.xml از دو بخش اصلی زیر تشکیل‌شده که هر بخش توضیح داده خواهد شد:

- **بخش اول: ProductGroups**

این بخش به‌صورت تناظر یک‌به‌یک از بخش TypeInheritance تولید خواهد شد.

مثال: فرض کنید بر اساس تعاریف موجود در کد میانی اصلی، دسته‌بندی شکل ۴-۱۹ تشکیل‌شده باشد.

```
<TypeInheritance>
    <Group id="1">
        <bool size="1"/>
        <int size="2"/>
        <float size="3"/>
    </Group>
    <Group id="2">
        <string size="1"/>
    </Group>
    <Group id="3">
        <Wood size="1"/>
        <Paper size="2"/>
        <Book size="3"/>
    </Group>
</TypeInheritance>
```

شکل (۴-۱۹) گروه‌های نوع داده‌ای قبل از تبدیل

درنتیجه در فایل level.xml با ساختاری مانند شکل ۴-۲۰ روبرو خواهیم شد.

- **بخش دوم: Level**

در این بخش دستوارت موجود در بدنه توابع به چیدمانی از بازی (مطابق جدول ۴-۶) تبدیل خواهند شد. در جدول ۴-۷ نحوه نگاشت اطلاعات استخراج از کد میانی به عنصرهای بازی، بر اساس سیستم نوع، نوع داده‌ای نمایش داده‌شده است.

```
<ProductGroups>
    <ProductGroup id="3" widthCount="3">
        <Product size="1" name="Wood"/>
        <Product size="2" name="Paper"/>
        <Product size="3" name="Book"/>
    </ProductGroup>
    <ProductGroup id="2" widthCount="1">
        <Product size="1" name="string"/>
    </ProductGroup>
    <ProductGroup id="1" widthCount="3">
        <Product size="1" name="bool"/>
        <Product size="2" name="int"/>
        <Product size="3" name="float"/>
    </ProductGroup>
</ProductGroups>
```

شکل (۴-۲۰) گروه‌های نوع داده‌ای پس از تبدیل

جدول (٤-٧) نحوه نگاشت اطلاعات استخراج‌شده به عنصرهای بازی

تعریف متغیر	var int a = 4, b, c;

Code.xml	`<Type ref_lineNumber="7" initial="true" baseType="int" array="-1">a</Type>`
	`<Type ref_lineNumber="7" initial="false" baseType="int" array="-1">b</Type>`
	`<Type ref_lineNumber="7" initial="false" baseType="int" array="-1">c</Type>`

Level.xml	`<Conveyor id="0" ref_lineNumber="7" name="a" productID="1" size="2" initial="1"/>`
	`<Conveyor id="1" ref_lineNumber="7" name="b" productID="1" size="2" initial="0"/>`
	`<Conveyor id="2" ref_lineNumber="7" name="c" productID="1" size="2" initial="0"/>`

تعریف متغیر	var Book d = Book();
Code.xml	`<Type ref_lineNumber="8" initial="true" baseType="Book" array="-1">d</Type>`
Level.xml	`<Conveyor id="3" ref_lineNumber="8" name="d" productID="3" size="3" initial="1"/>`
ادعا	assert(d, Book);
Code.xml	`<Assert baseType="Book" array="-1">d</Assert>`
Level.xml	`<Restriction id="3" restrictedSize="3"/>`
انتساب	a = b ;

Code.xml	`<Assignment ref_lineNumber="10">`
	`  <Left>a</Left>`
	`  <Right count="1">`
	`  <Primary source="var">b</Primary>`
	`  </Right>`
	`</Assignment>`

Level.xml	`<Merger inputCount="1" ref_lineNumber="10">`
	`  <outputConveyorID>0</outputConveyorID>`
	`  <Input from="conveyor">`
	`  <ID>1</ID>`
	`  </Input>`
	`</Merger>`

انتساب	a = b * c;

<Assignment ref_lineNumber="11"> <Left>a</Left> <Right count="2"> <Primary source="var">b</Primary> <Primary source="var">c</Primary> </Right> </Assignment>	Code.xml
<Merger inputCount="2" ref_lineNumber="11"> <outputConveyorID>0</outputConveyorID> <Input from="conveyor"> <ID>1</ID> </Input> <Input from="conveyor"> <ID>2</ID> </Input> </Merger>	Level.xml
a = foo();	فراخوانی و انتساب
<Assignment ref_lineNumber="12"> <Left>a</Left> <Right count="1"> <Primary source="call"> <Call name="foo" argCount="0"/> </Primary> </Right> </Assignment>	Code.xml
<Factory targetLevel="0" inputCount="0"/> <Merger inputCount="1" ref_lineNumber="12"> <outputConveyorID>0</outputConveyorID> <Input from="factory"/> </Merger>	Level.xml
a = foo(b, true);	فراخوانی و انتساب
<Assignment ref_lineNumber="13"> <Left>a</Left> <Right count="1"> <Primary source="call"> <Call name="foo" argCount="2"> <Primary source="var">b</Primary> <Primary source="data"> <Type baseType="bool" array="-1"/> </Primary> </Call> </Primary> </Right> </Assignment>	Code.xml

<Factory targetLevel="0" inputCount="2"> <Input from="conveyor"> <ID>1</ID> </Input> <Input from="producer"> <ProductID>1</ProductID> <Size>1</Size> </Input> </Factory> <Merger inputCount="1" ref_lineNumber="13"> <outputConveyorID>0</outputConveyorID> <Input from="factory"/> </Merger>	Level.xml
foo("p1");	فراخوانی
<Assignment ref_lineNumber="14"> <Left count="0"/> <Right count="1"> <Primary source="call"> <Call name="foo" argCount="1"> <Primary source="data"> <Type baseType="string" array="-1"/> </Primary> </Call> </Primary> </Right> </Assignment>	Code.xml
<Factory targetLevel="0" inputCount="1" ref_lineNumber="14"> <Input from="producer"> <ProductID>2</ProductID> <Size>1</Size> </Input> </Factory>	Level.xml

۴-۶- جمع‌بندی

در این فصل بخش‌های مختلف سیستم ارائه‌شده به‌صورت مجزا توضیح داده‌شده. همچنین در هر بخش نحوه پیاده‌سازی و نحوه ارتباط با بخش بعدی مشخص گشت. باوجود پیچیدگی‌های کار سعی شد تا ایده ارائه‌شده به‌صورت کامل بررسی گردد و به نحوه پیاده‌سازی آن پرداخته شد.

همچنین در این فصل ارتباط بسیاری از مفاهیم ارائه‌شده در فصل دوم {مفاهیم بنیادی} با سیستم ارائه‌شده در این پژوهش مشخص گردید و مقایسه‌ای با سیستم‌های موجود انجام گشت تا توانایی‌های جدید ارائه‌شده در پژوهش جاری متمایز گردد.

برای هرچه بهتر مشخص شدن کارایی سیستم، در فصل بعد نحوه استفاده از اجزای سیستم به‌صورت یکپارچه پرداخته خواهد شد. این کار باعث می‌شود تا در صورت وجود ابهاماتی در نحوه ارتباط اجزای مختلف سیستم، جزییات واضح گردد و در عمل نحوه استفاده از سیستم به‌صورت یکجا مشخص گردد. همچنین در فصل بعد تلاش می‌شود تا یک ارزش سنجی از سیستم صورت بگیرد.

فصل ۵:

مثال‌های موردی

۵-۱- مقدمه

همان‌طور که پیش‌تر اشاره شد هر فرایند درستی‌یابی برای یک قطعه کد به یکی از دو وضعیت نهایی «اثبات درستی کد» و یا «گزارش خطا در کد» منجر خواهد شد. در این فصل سعی می‌گردد نحوه عملکرد سیستم ارائه‌شده در پژوهش با انجام چند سناریو مورد نمایش قرار بگیرد. هدف از این کار ارزیابی پژوهش در قالب چند سناریوی واقعی است.

این کار باعث می‌شود که بتوان بر میزان کارایی این پژوهش و احتمالاً در آینده برای مقایسه با سایر پژوهش‌های مرتبط معیارهای مشترکی وجود داشته باشد. درواقع می‌توان سناریوهای ارائه‌شده در این فصل را به‌عنوان معیار مشترکی برای مقایسه این پژوهش و نتایج حاصل از آن با پژوهش‌های احتمالی در آینده در این حوزه دانست.

همچنین وجود این سناریوها و تحلیل هرکدام به‌صورت تشریحی به فهم بهتر نحوه کار این سیستم کمک خواهد کرد و نقاط پنهان این پژوهش را روشن خواهد کرد. درواقع استفاده از مثال نه‌تنها به‌منظور ارزیابی ارزش پژوهشی کار کمک می‌کند، بلکه باعث می‌شود خواننده به‌صورت کامل بر پژوهش اشراف پیداکرده و در پژوهش‌های بعدی سعی در بهبود آن داشته باشد.

اهمیت این فصل در به‌کارگیری آن در آینده خواهد بود. این فصل باید به‌گونه‌ای جدی در نظر گرفته گردد. چراکه انجام هر پژوهشی که به‌نوعی در ارتباط با موضوعات کلیدی این پژوهش باشد، نیازمند ارزیابی و مدنظر قرار دادن کارها و تحقیقات مشابه است. به‌صورت کلی ارزیابی یک سیستم، تعیین میزان انطباق برنامه با مسیر تعیین‌شده و نمایاندن نقاط ضعف و قوت آن است و نتیجه ارزیابی می‌تواند تصویری شفاف‌تری از چگونگی فعالیت‌ها ارائه نماید تا پژوهشگران این حوزه بتوانند آگاهانه تصمیمات لازم را در جهت تقویت و یا اصلاح و ادامه برنامه بگیرند. در حقیقت دلیل ارزیابی فراهم‌سازی شالوده‌ای برای تصمیم‌گیری درست، درباره وضعیت آغازین تغییرات مورد مشاهده و یا وقوف بر برنامه‌ها است. ایجاد عقلانیت برای تغییرات و جهت‌دهی به برنامه‌ریزی‌ها برای توسعه مسیر پژوهش‌های فعلی و در جریان از دیگر مهم‌ترین اهداف ارزیابی است.

۵-۲- فرضیات

برای ارائه بهتر این فصل ابتدا سعی می‌شود تا یکسری موارد در قالب چارچوبی مشخص تبیین گردد. این کار به هم‌راستا سازی سناریوها و فهم راحت‌تر ارتباط آن‌ها با یکدیگر کمک می‌کند و سناریوها را برای ارزیابی سیستم پیشنهاد داده‌شده همگون می‌سازد.

سیستم نوع انتخاب‌شده در این فصل، نوع‌های داده‌ای خواهد بود. علت این انتخاب دلایل گوناگونی است که در اینجا به یک مورد از آن اشاره خواهیم داشت. در عمل این سیستم نوع در مقایسه با سایر سیستم‌های نوع اشاره‌شده در فصول گذشته، در قسمت نمایش بصری بازی بسیار قابل‌درک‌تر بوده و فرآیند انتقال مفاهیم را ساده می‌کند. این انتقال روان به هرچه بهتر ارزیابی و مقایسه‌شدن سیستم پیشنهادی با سایر کارهای مشابه کمک خواهد کرد.

ازآنجاکه «درستی» یک قطعه کد در این پژوهش به معنی سازگاری تمام اجزای کد در رابطه با سیستم نوع موردنظر در درستی‌یابی و «خطا» در یک قطعه کد به معنی وجود یک یا بیش از یک ناسازگاری است، سناریوهای ارائه‌شده در این فصل از ایجاد ناسازگاری در سیستم نوع نوع‌های داده‌ای استفاده می‌کند.

بدین منظور در تمام سناریوهای خطادار ارائه‌شده در این فصل، درواقع از انتسابات اشتباه ازنقطه‌نظر نوع داده‌ای، برای ایجاد خطا استفاده می‌شود. منظور از انتساب خطا در نوع داده‌ای، انتساب دو نوع داده‌ای قابل‌تبدیل ولی متفاوت است.

به‌عنوان‌مثال دو نوع داده‌ای int و float یا دو نوع داده‌ای Point و Shape مثال‌هایی از انتساب‌های خطادار بین دو نوع داده‌ای قابل‌تبدیل را ارائه می‌دهند. قابل‌تبدیل بودن نوع‌های داده‌ای اصولاً تنها از یک‌سو بدون از دست دادن اطلاعات ممکن است و در این پژوهش نوع‌های داده‌ای با مؤلفه اندازه[1] این ویژگی را در بازی نشان می‌دهند.

بدین منظور در ادامه سه مثال بررسی خواهد شد. هر سناریویک وضعیت متفاوت را پوشش خواهد داد. همچنین سعی شده است که در هر سناریو حالت‌های محتمل قابل انجام توسط بازیکن لحاظ و تشریح گردد تا تمام وضعیت‌ها پوشش داده شود. البته این کار ممکن است به همپوشانی قسمت‌هایی از این چند سناریو منجر شود که انکارناپذیر است.

در سناریو اول، هدف بررسی یک حالت اثبات درستی کد خواهد بود. مثال دوم به بررسی یک قطعه کد حاوی خطا می‌پردازد. برای نمایش مثالی از عملکرد سیستم امتیازدهی، این خطا به دو روش متفاوت رفع خواهد شد. در پایان گزارش ایجادشده برای این دو روش با یکدیگر مقایسه خواهد شد. در مثال سوم نیز همانند مثال دوم، به بررسی یک سناریو حاوی خطا متفاوت پرداخته می‌شود تا نحوه تأثیر دستورات ادعایی نیز مشخص شود.

[1] size

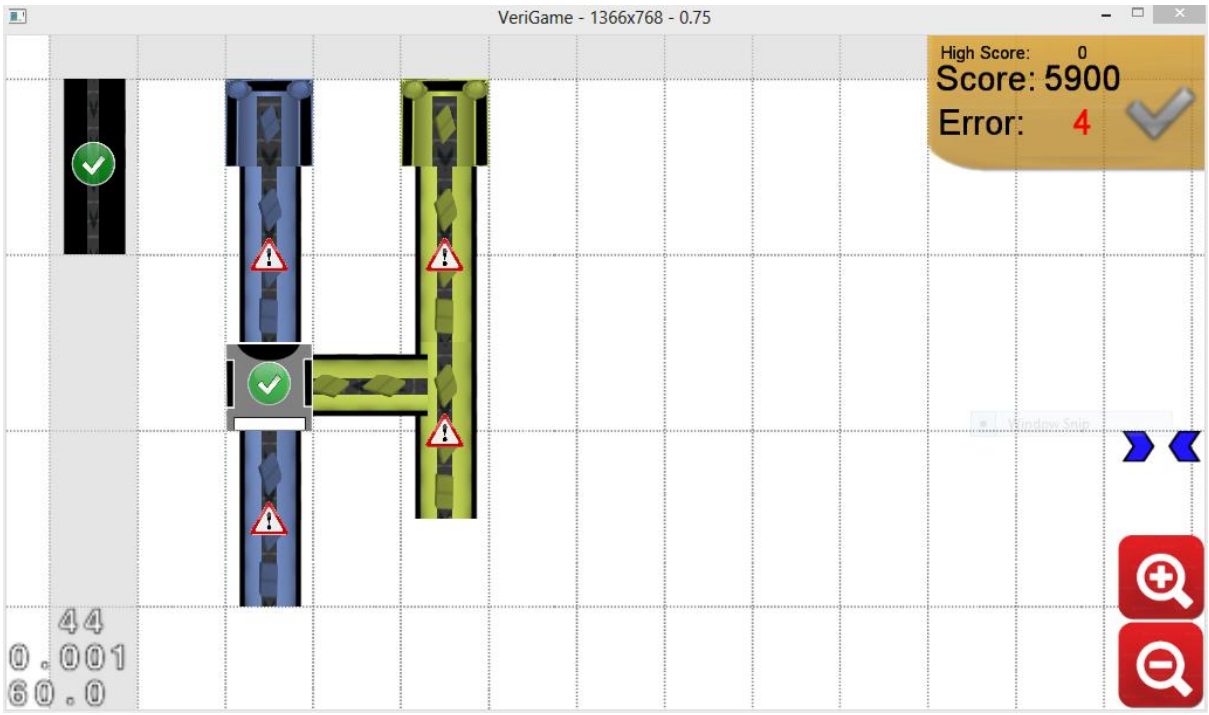
۵-۳- مثال اول: اثبات درستی کد

در این بخش به سناریوی قطعه کد درست پرداخته خواهد شد. بدین منظور از قطعه کد میانی شکل ۵-۱ استفاده‌شده است:

```
function void main()
{
    var Paper p1 = Paper();
    var Paper p2 = Paper();
    p1 = p2 ;
}
```

شکل (۵-۱) کد مثال اول

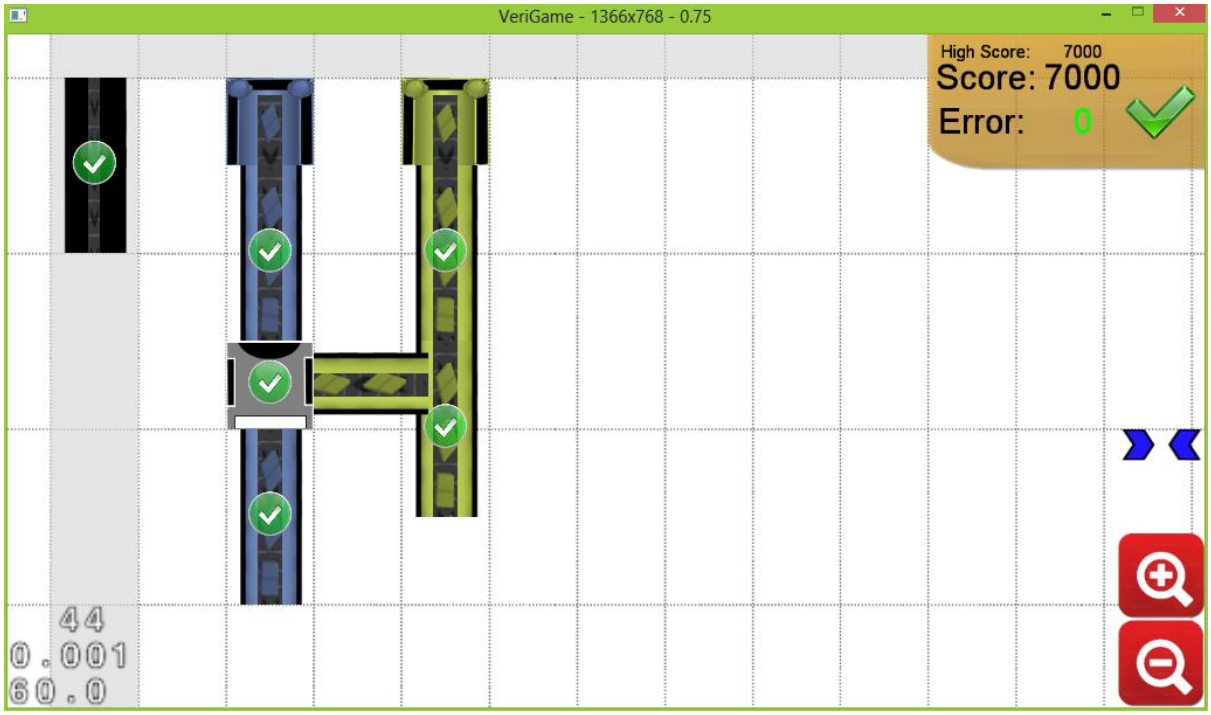
این کد پس از انجام مراحلی تحلیل و تبدیل بر اساس سیستم نوع فرض شده (نوع داده‌ای) به مانند شکل ۵-۲ به بازیکن نمایش داده خواهد شد:



شکل (۵-۲) نمایش مثال اول در بازی

لازم به ذکر است که اجرای مجدد این قطعه کد می‌تواند به چیدمان متفاوتی (ازلحاظ اندازه عرض تسمه‌نقاله‌ها) منجر گردد.

در سناریو جاری، بازیکن باید سعی کند تعداد خطاها را به صفر برساند. برای حل این سناریوی ساده کافی است بازیکن هر دو خط نقاله آبی‌رنگ و زردرنگ را هم‌عرض کالای ورودی کند:



شکل (۵-۳) مثال اول: راه‌حل اول

با این کار وضعیت هردو خط نقاله به وضعیت «امن» تبدیل می‌گردد و امکان اعمال پاسخ (تیک سبزرنگ گوشه صفحه) فعال می‌گردد. گزارش حاصل از این سناریو درواقع یک اثبات درستی کد ازلحاظ سیستم نوع نوع‌های داده‌ای خواهد بود، چراکه عرض‌های اعمال‌شده توسط بازیکن متناظر با نوع‌های داده‌ای در نظر گرفته‌شده در خود کد اصلی بوده است.

البته در این سناریو ممکن است یک بازیکن با انتخاب یک تسمه با عرض بزرگ‌تر به حالت بی‌خطا دست پیدا کند. این حالت به ازای تسمه آبی‌رنگ در شکل ۵-۴ نمایش داده‌شده است.

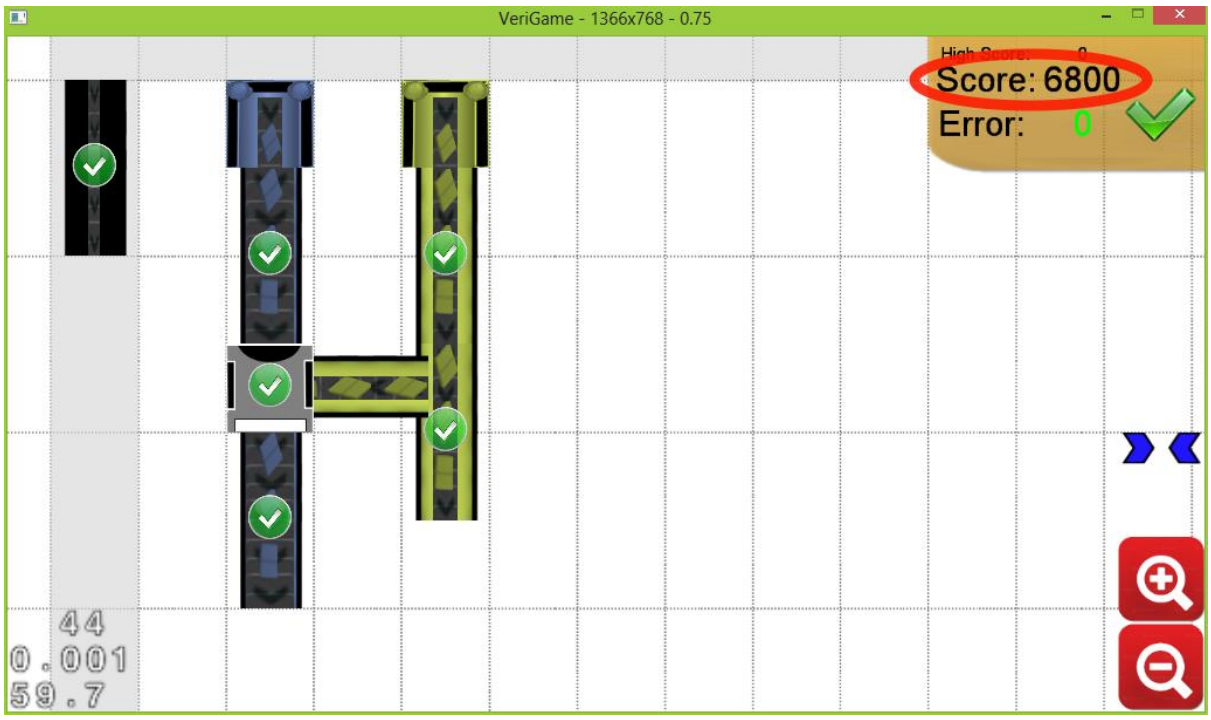
این حالت گرچه یک وضعیت امن در تمام خطوط نقاله ایجاد می‌کند، اما درواقع استفاده نادرست از نوع داده‌ای خواهد بود. این اتفاق در گزارش به‌صورت شکل ۵-۵ نمایش داده می‌شود.

این گزارش از برنامه‌نویس می‌خواهد که متغیر p1 (متناظر با تسمه‌نقاله آبی‌رنگ) را از نوع داده‌ای Book انتخاب کند. همچنین این گزارش برای کمک به انجام این کار، نام فایل و شماره خطی که در آن متغیر p1 تعریف گشته است را مشخص می‌کند.

البته همان‌طور که مشخص است، گزارش ارائه‌شده توسط بازیکن دوم آنچه باید گزارش بشود نیست و گزارش از عمل ناصحیحی را به برنامه‌نویس می‌دهد.

درواقع در اینجا نقش سیستم امتیازدهی بازی پررنگ می‌گردد. چراکه بازیکن دوم در اثر انتخاب غیر بهینه خود به امتیاز پایین‌تری نسبت به بازیکن اول رسیده است. در این سیستم مراحل به‌صورت گسترده و

در اینترنت پخش می‌گردد و گزارش‌ها تنها از نتایج مراحل حل‌شـــده با بالاترین امتیاز به برنامه‌نویس گزارش داده خواهد شد.



شکل (۵-۴) مثال اول: راه‌حل دوم

result.log

```
1  ************************************************
2  *************     RESULT     ********************
3  ************************************************
4
5
6  > Function Name : main
7  1> code.cpp(line 20)      variable "p1" ==> Book
8
```

شکل (۵-۵) گزارش حاصل از راه‌حل دوم

درنتیجه گزارش دوم به سـیسـتم بازتاب نخواهد شـد و تنها «اثبات درستی کد» به سیستم و در ادامه به برنامه‌نویس گزارش داده خواهد شد.

۵-۴- مثال دوم: گزارش خطا در کد

همان‌طور که پیش‌تر نیز اشـــاره شـــد، درســـتی‌یابی یک قطعه کد می‌تواند منجر به اعلام خطا در کد ورودی

باشـد که در این پژوهش معادل وجود ناسازگاری در کد با توجه به سیستم نوع موردنظر است. یکی از اهداف اصلی در این پژوهش، ارائه راهکاری خودکار برای رفع هشدارهای[1] احتمالی در کد است.

عموما این هشـدارها از دیدگاه برنامه‌نویسـان هشـدار های بی موردی از سـوی مترجم (کامپایلر) بوده و توانایی‌های ضـعیف ماشـین های امروزی در تشـخیص این موضوع باعث اعلام ازاین‌دست هشدارها می‌گردد. هرچند گاهی اوقات نیز این هشـدار ها بدرسـتی یک خطا در کد را مشـخص کرده ولی متاسـفانه از دید برنامه‌نویس به اشتباه زائد فرض شده است.

در این بخش با ارائه یک مثال قصـد داریم که یک سـناریو فرضـی از یک اشـتباه را به کمک پژوهش ارائه‌شـده رفع کنیم و نحوه انجام این کار را توسـط یک انسـان غیر متخصـص در حوزه کامپیوتر (یک فرد عادی) شبیه سازی کنیم.

در این سناریو از قطعه کد ساده شکل ۵-۶ استفاده گشته است:

```
function void main()
{
    var Wood w = Wood();
    var Paper p = Paper();
    var Book b = Book();
    p = b ;
}
```

شکل (۵-۶) کد مثال دوم

تمام نوع‌های داده‌ای که هدف اصـلی در درسـتی‌یابی این قطعه کد هسـتند، توسـط برنامه‌نویس تعریف گشته‌اند. علت این انتخاب، نمایش یک سناریوی پیچیده‌تر نسبت به بخش قبل است.

حال فرض کنید نوع‌های داده‌ای غیر اصـلی Wood و Paper و Book که در این کد اسـتفاده گشته‌اند به شکل ۵-۷ تعریف‌شده باشند:

```
type Wood {
};
type Paper : Wood {
};
type Book:Paper {
};
```

شکل (۵-۷) نحوه تعریف نوع‌های داده‌ای استفاده‌شده در مثال

همان‌طور که در بخش ۴-۳-۴ توضیح داده‌شده است، سلسله‌مراتب خطی در ارث‌بری این سه نوع داده‌ای، کمک خواهد کرد که این سـه نوع داده‌ای در کنار یکدیگر یک گروه نوع داده‌ای را تشـکیل دهند. این گروه

[1] Warning

نوع داده‌ای در مراحل «تحلیل» و «ایجاد مرحله» به مانند شکل ۵-۸ می‌گردند:

```
<Group id="3">
    <Wood size="1"/>
    <Paper size="2"/>
    <Book size="3"/>
</Group>
```

Code.xml

```
<ProductGroup id="3" widthCount="3">
    <Product size="1" name="Wood"/>
    <Product size="2" name="Paper"/>
    <Product size="3" name="Book"/>
</ProductGroup>
```

Level.xml

شکل (۵-۸) نحوه تبدیل نوع‌های داده‌ای

کد فوق درنهایت پس از گذشت از مراحل «تحلیل» و «ایجاد مرحله بازی» به مانند شکل ۵-۹ در بازی به نمایش درخواهد آمد:

شکل (۵-۹) نمایش مثال دوم در بازی

همان‌طور که انتظار می‌رود یک خطا به دلیل قطعه کد p=b; نمایش داده‌شده است. این خطا با نمایش یک علامت خطا بر روی خط تسمه‌نقاله زردرنگ مشخص‌شده است. این خطا حاصل عدم مطابقت نوع داده‌ای متغیر p و b است. این عدم مطابقت در بازی با بزرگ‌تر بودن کالاهای قرارگرفته بر روی تسمه‌نقاله نمایش داده‌شده است.

سناریو فوق به چندین روش قابل‌حل است که در این بخش به بررسی تمام راه‌حل‌های ممکن و مقایسه نتیجه حاصل از این راه‌حل‌ها در کنار یکدیگر پرداخته می‌شود.

◘ راه‌حل اول

یکی از راه‌های ممکن برای حل خطای فوق تغییر عرض خط نقاله زردرنگ است. حاصل این عمل در بازی به مانند شکل ۵-۱۰ است.

شکل (۵-۱۰) مثال دوم: راه‌حل اول

گزارش ارائه‌شده از این حل اعلام می‌کند که متغیر p (متناظر با نقاله زردرنگ) نیاز به تغییر نوع داده‌ای دارد. این گزارش کاملاً درست بوده و درواقع برای جلوگیری از دست دادن اطلاعات متغیر b (متناظر با نقاله بنفش‌رنگ) نیازمند تغییر نوع داده‌ای مقصد انتساب (p) به نوع داده‌ای هم‌رده یا بزرگ‌تر هستیم.

result.log

```
1  ************************************************
2  *************     RESULT     ********************
3  ************************************************
4
5
6  > Function Name : main
7  1> code.cpp(line 21)      variable "p" ==> Book
```

شکل (۵-۱۱) گزارش حاصل از راه‌حل اول (مثال دوم)

در ادامه راه‌حل دیگری را بررسی می‌کنیم.

راه‌حل دوم

در این راه‌حل از تبدیل‌کننده جادویی برای حل ناسازگاری کمک گرفته می‌شود. بدین منظور در محل بروز خطا (محل اتصال دو نقاله به‌واسطه دستگاه ادغام کننده) یک تبدیل‌کننده جادویی قرار داده می‌شود. این روش نیز به از بین رفتن خطاهای مرحله منجر می‌گردد. چراکه تبدیل‌کننده جادویی توانایی کوچک کردن اندازه کالای عبوری را دارد و این باعث حل خطای ناامنی خط نقاله زردرنگ خواهد گشت. حاصل این عمل در شکل ۵-۱۲ آورده شده است:

شکل (۵-۱۲) مثال دوم: راه‌حل دوم

گزارش حاصل از این حل (با توجه به سیستم نوع، نوع داده‌ای) به‌عنوان یک تبدیل نوع[1] در انتساب p=b; خواهد بود:

result.log

```
1  ************************************************
2  *************     RESULT     ********************
3  ************************************************
4
5
6  > Function Name : main
7  1> code.cpp(line 23)      Cast ==> Paper
```

[1] Type Casting

شکل (۵-۱۳) گزارش حاصل از راه‌حل دوم (مثال دوم)

◘ مقایسه دو راه‌حل

هرچند در این سناریو دو راه‌حل کاملاً متفاوت ارائه گشت، اما از دیدگاه فنی راه‌حل اول ارزشمندتر و منطقی‌تر است و از سیستم انتظار می‌رود تنها راه‌حل اول برای رفع گزارش به برنامه‌نویس بازتاب شود و وقت ارزشمند برنامه‌نویس برای بررسی تمام راه‌حل‌های ممکن اتلاف نگردد.

در اینجا نیز سیستم امتیازدهی بازی به امر تا حد زیادی کمک می‌کند. همان‌طور که در تصاویر هر دو راه‌حل مشخص گشته بود، امتیاز دو راه‌حل به صورت جدول ۵-۱ می‌باشد.

جدول (۵-۱) مقایسه دو راه‌حل (مثال دوم)

راه‌حل	اول	دوم
روش اعمالی در کد	تغییر نوع متغیر	تبدیل نوع داده‌ای محتوا
امتیاز کسب‌شده	۶۸۰۰	۵۰۰۰

زمانی که تعداد زیادی بازیکن بتوانند با ارائه راه‌حل بهتر به بالاترین امتیاز دست پیدا کنند، سایر راه‌حل‌ها به‌عنوان راه‌حل قطعی و یا گزارش خطا به برنامه‌نویس بازتاب نمی‌گردد. در سناریو فوق نیز راه‌حل دوم با کسب امتیاز پایین به برنامه‌نویس گزارش نخواهد شد.

۵-۵- مثال سوم: گزارش خطا در کد

در سناریو سوم یکی دیگر از وضعیت‌های ممکن در رابطه باوجود ادعا[1] در کد پوشش داده می‌شود. بدین منظور از قطعه کد شکل ۵-۱۴ استفاده‌شده است:

```
1  function void main()
2  {
3      var Wood w = Wood();
4      var Paper p = Paper();
5      var Book b = Book();
6      p = b ;
7      assert(p, Paper);
8  }
```

شکل (۵-۱۴) کد مثال سوم

در خط هفتم این قطعه کد، برنامه‌نویس ادعایی در رابطه با نوع داده‌ای متغیر p کرده است که طبق آن مشخص می‌گردد که محتوای متغیر p در این قسمت از جریان اجرای برنامه باید از نوع Paper باشد.

[1] assertion

این نوع فرضیات و ادعاها در بازی به کمک نقاله‌های قفل‌شده نمایش داده می‌شود. شکل ۵-۱۵ نمای بازی مرتبط با کد فوق است.

شکل (۵-۱۵) نمایش مثال سوم در بازی

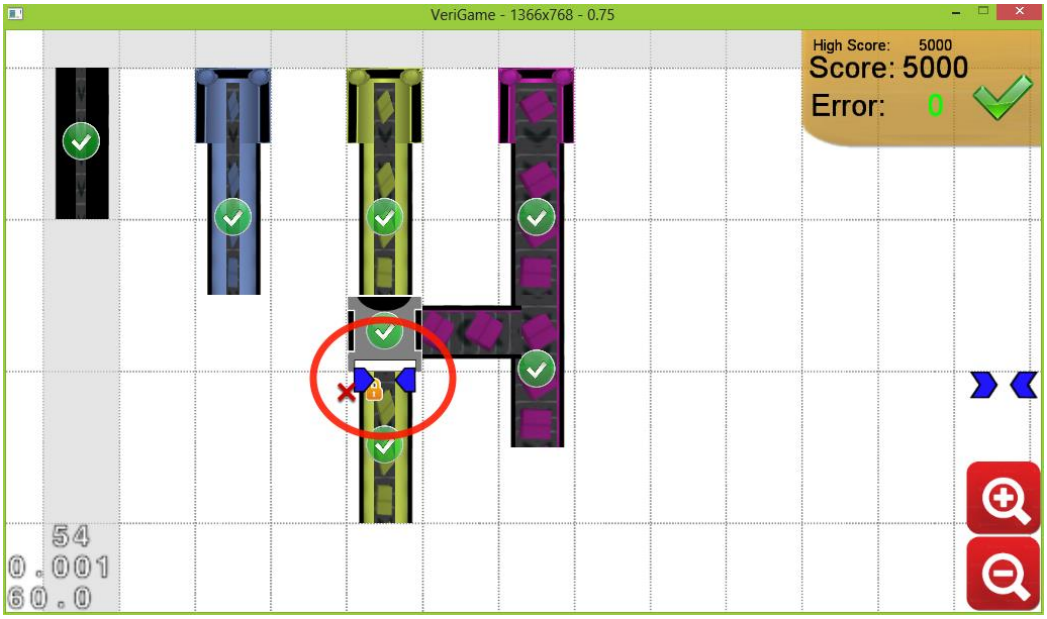
در قطعه نقاله بعد از دستگاه ادغام کننده، نماد قفل نشان داده‌شده است. این به معنی عدم تغییر عرض تسمه این نقاله است. همچنین نماد هشدار در همین قطعه اشاره به ناامنی آن دارد.

بازیکن برای حل این معما برخلاف سناریو بخش قبل امکان تغییر عرض تسمه‌نقاله را نخواهد داشت. چراکه در صورت انجام این کار در سایر بخش‌های این خط نقاله، همچنان در قطعه ذکرشده عرض ثابت می‌ماند و وضعیت ناامن باقی خواهد ماند.

در این سناریو بازیکن مجبور به استفاده از تبدیل‌کننده جادویی در قسمت ناامن خواهد بود. حاصل این عمل یک تبدیل نوع داده‌ای در انتساب خط ششم خواهد بود.

گزارش حاصل از این عمل نیز در شکل ۵-۱۷ آورده شده است.

گزارش ارائه‌شده درواقع اشاره به تبدیل نوع داده‌ای در خط ششم قطعه کد ورودی دارد که برنامه‌نویس در مورد آن تصمیم خواهد گرفت.



شکل (۵-۱۶) مثال سوم: راه‌حل اول

result.log

```
1  ************************************************
2  *************     RESULT     ********************
3  ************************************************
4
5
6  > Function Name : main
7  1> code.cpp(line 6)  Cast ==> Paper
```

شکل (۵-۱۷) گزارش حاصل از راه‌حل اول (مثال سوم)

۵-۶- جمع‌بندی

همان‌طور که مشاهده شد، در این فصل سه سناریو متفاوت ارائه گشت و هرکدام از این سه سناریو به تفضیل شرح داده شدند. همچنین برای ارائه هرچه بهتر سناریوها یک سیستم نوع مشخص گشت و از سناریوها حول یک قطعه کد مشابه ارائه داده شدند.

هدف کاربردی تمام این سه سناریو درواقع ارائه گزارش خطاهای قطعه کد ورودی و یا در نقطه مقابل، اثبات درستی آن قطعه کد ازنقطه‌نظر سیستم نوع مورد فرض فصل بود و این کار (نمایش گزارش خروجی) در تمام سناریوها انجام شد. نکته مهم گزارش‌های فوق اطلاعات کاملاً دقیق و درست به ازای تمام سناریوها است به‌طوری‌که از دیدگاه عملی سیستم ارائه‌شده در این پژوهش توانایی استفاده در صنعت را دارا است و خطای آن صفر است.

فصل ٦:

جمع‌بندی و کارهای آینده

۶-۱- جمع‌بندی

این پژوهش قصد داشت تا با به‌کارگیری راه‌های غیرتخصصی، مانند بازی، انجام بخشی از فعالیت‌های درستی بابی که نیازمند صرف وقت و هزینه بالا است را به شکل روندی جمع‌سپاری دربیاورد. این کار باعث می‌شود گستره نیروی کار ممکن به دلیل عدم نیاز به دانش تخصصی در دامنه جدید، افزایش چشم‌گیری داشته باشد.

راهکار ارائه‌شده در این پژوهش تبدیل مراحل آزمون نرم‌افزار به یک جورچین بصری است که می‌تواند توسط مردم عادی حل گردد. درواقع راه‌حل‌های بازی درنهایت به اثباتی برای درستی نرم‌افزار یا به گزارشی از نحوه حل خطای موجود در نرم‌افزار تبدیل می‌گردد.

هرچند سیستم ارائه‌شده در این پژوهش کامل نبوده و برای تکمیل آن در ادامه پیشنهادهای گوناگونی ارائه‌شده است، اما ایده اصلی پشت این سیستم می‌تواند مسیر تازه‌ای در حوزه درستی بابی نرم‌افزارها مشخص کند.

ارائه گزارش فوق و نتایج حاصل از آن می‌تواند انگیزه لازم برای ادامه پژوهش را در علاقه‌مندان نه‌تنها حوزه آزمون و درستی‌یابی نرم‌افزار، بلکه حوزه بازی‌سازی را ایجاد کند و گام‌های بعدی در تکمیل این سیستم را شاهد بود.

۶-۲- کارهای آینده

در این بخش کمبودهای سیستم فعلی که نیازمند صرف پژوهش‌های آتی است ارائه گشته است. امید می‌رود که پژوهشگران علاقه‌مند حوزه آزمون نرم‌افزار با همکاری فعالان صنعت بازی‌سازی، در انجام کارهای پیشنهادی زیر تلاش کنند:

- **پشتیبانی از زبان‌های غیر تابعی و شی‌گرایی:**

همان‌طور که از ابتدا بیان شد، مبنای سیستم ارائه‌شده وجود یک کد ساخت‌یافته به‌عنوان ورودی است. درواقع توابع و کلاس‌ها به‌نوعی باعث ایجاد یک نوع تقسیم‌بندی از پیش تعریف‌شده برای قالب دادن به بازی می‌شوند و درصورتی‌که یک‌زبان از این ویژگی مستثنا باشد، باید برای نحوه ایجاد مراحل بازی کاری متفاوت انجام داد که در این پژوهش به این موضوع توجهی نگشته است. همچنین توابع باعث می‌شوند تا متغیرها به‌عنوان یکی از اجزای اصلی درستی‌یابی به شکلی مشخص بررسی گردند. درنتیجه برای پشتیبانی از سایر زبان‌ها نیاز است تا چهارچوبی مشخص ایجاد گردد.

◘ استفاده از مسیرهای اجرایی موجود در کد باهدف کوتاه کردن مراحل بازی:

می‌توان با اســـتفاده از پردازش کد در ابتدا و تبدیل توابع به چندین مســـیر، باعث شـــد تا مراحل به‌مراتب کوتاه‌تر گردند. این کار درروند فکری بازیکن تأثیر مثبت خواهد داشت.

◘ پشتیبانی از توصیف‌گرهای غیرقابل مقایسه:

یکی دیگر از کمبودهای ســیســتم فعلی عدم پیاده‌ســازی بخشــی از بازی اســت که بتوان به کمک آن ظاهر کالاها را با توجه به جنس توصیف‌گر متفاوت نمایش داد. این کار اگرچه در طراحی بازی لحاظ گردیده است، اما بخش پیاده‌سازی آن انجام‌نشده است.

◘ پیاده‌سازی زیر سالن:

یکی دیگر از مفاهیم انتزاعی پیاده‌ســازی نشــده در ســیســتم فعلی، پشــتیبانی از فراخوانی‌ها در کد به‌صورت یکپارچه و مشــابه آنچه در بازی PipeJam مشاهده شد، است. بدین منظور طراحی و پیاده‌سازی هر دو عنصر مرحله و جهان در کنار برد بازی در جذابیت و کارایی سیستم تأثیر بسزایی خواهد داشت.

مراجع

مراجع

[1] Paul Ammann, Jeff Offutt, Introduction to Software Testing, 1st ed. Reading, Cambridge University Press, 2008.

[2] Benjamin C. Piercem, Types and Programming Languagesv, 1st ed. Reading, The MIT Press, 2002.

[3] Cem Kaner, Jack Falk, Hung Q. Nguyen, Testing Computer Software, 2nd ed. Reading, Wiley, 1999.

[4] Rex Black, Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing, 3rd ed. Reading, Wiley, 2009.

[5] Steven R. Rakitin, Software Verification and Validation for Practitioners and Managers, 2nd ed. Reading, Artech House Print on Demand, 2001.

[6] Werner Dietl, Stephanie Dietzel, Michael D. Ernst, Nathaniel Mote, Brian Walker, Seth Cooper, Timothy Pavlik, Zoran Popovic, "Verification Games: Making Verification Fun", ACM 978-1-4503-1272-1/12/06 (June 2012)

[7] Wenchao Li, Sanjit A. Seshia, Somesh Jha, "CrowdMine: Towards Crowdsourced Human-Assisted Verification", ACM 978-1-4503-1199-1/12/06, June 2012.

[8] Colin S. Gordon, Michael D. Ernst, Dan Grossman, "Rely-Guarantee References for Refinement: Types Over Aliased Mutable Data", ACM 978-1-4503-2014-6/13/06, June 2013.

[9] Wei Huang, Ana Milanova, Werner Dietl, Michael D. Ernst, "ReIm & ReImInfer: Checking and Inference of Reference Immutability and Method Purity", ACM 978-1-4503-1561-6/12/10, October 2012.

[10] Todd W. Schiller, Michael D. Ernst, "Reducing the Barriers to Writing Verified Specifications", ACM 978-1-4503-1561-6/12/10, October 2012.

[11] Colin S. Gordon, Werner Dietl, Michael D. Ernst, and Dan Grossman, "JavaUI: Effects for Controlling UI Object Access"

[12] Wei Huang, Werner Dietl, Ana Milanova, and Michael D. Ernst, "Inference and Checking of Object Ownership"

[13] Kedar S. Namjoshi1, Lenore D. Zuck, "Witnessing Program Transformations"

[14] P.Grubb, Software Maintenance: Concepts and Practice, 2nd ed. Reading, World Scientific Publishing Company, 2003.

[15] Programming Language Popularity: http://langpop.com/

[16] Bill Evjen, Kent Sharkey, Thiru Thangarathinam, Professional XML, 1st ed. Reading, Wrox, 2007.

پیوست‌ها

پیوست الف: گرامر زبان میانی

قوانین گرامر طراحی شده برای زبان میانی بدین شرح است.

Program → DeclarationList
DeclarationList → Declaration DeclarationList | ε
Declaration → FunctionDeclaration | TypeDeclaration | VariableDeclaration
TypeDeclaration → TYPE ID OptionalSuperclass { MemberDeclarationList };
OptionalSuperclass → : ID |
MemberDeclarationList → MemberDeclaration MemberDeclarationList | ε
MemberDeclaration → VariableDeclaration | FunctionDeclaration
FunctionDeclaration → FUNC FunctionType ID (OptionalParameterList) Block
FunctionType → S_VOID | TypeName
OptionalParameterList → Parameter MoreParameters | ε
Parameter → TypeName Variable
MoreParameters → , Parameter MoreParameters | ε
Block → { DeclarationOrStatementList }
DeclarationOrStatementList → DeclarationOrStatement DeclarationOrStatementList | ε
TypeName → PrimitiveType | ID
PrimitiveType → bool | int | float | string
VariableDeclaration → VAR TypeName RestOfVariableDeclaration
RestOfVariableDeclaration → Variable MoreVariables;
OptionalPointer → * | ε
Dimensions → [NUM] Dimensions | ε
InitialValue → ID() | NUM | S_TRUE | S_FALSE | S_TEXT
OptionalInitial → = InitialValue | ε
Variable → OptionalPointer ID Dimensions OptionalInitial
MoreVariables → ,Variable MoreVariables | ε
MoreExpressions → , Expression MoreExpressions | ε
OptionalExpression → Expression | ε
Expression → Primary MoreOperand
MoreOperand → * Primary | ε

Primary → ID MorePrimary | NUM | TRUE | FALSE | TEXT
MorePrimary → ArrayAccess | FieldAccess | FunctionCall | ε

ArrayAccess → [Expression] MorePrimary
FieldAccess → .ID MorePrimary

FunctionCall → (OptionalArgumentList) MorePrimary
OptionalArgumentList → Expression MoreExpressions | ε
DeclarationOrStatement →VariableDeclaration | KeywordStatement | OtherStatement
KeywordStatement → Jump | Asserts
Jump → S_RETURN OptionalExpression;
Asserts → DataTypeAssert
DataTypeAssert → S_ASSERT(ID, TypeName);
OtherStatement → ID RestOfExpressionStatement
RestOfExpressionStatement → OptionalAssignment';'
OptionalAssignment → = Expression | ε
DeclarationOrStatement → VariableDeclaration | KeywordStatement | OtherStatement
KeywordStatement → Jump | Asserts
Jump → return OptionalExpression;
Asserts → DataTypeAssert
DataTypeAssert → S_ASSERT (ID, TypeName);
OtherStatement → ID RestOfExpressionStatement
RestOfExpressionStatement → OptionalAssignment;
OptionalAssignment → = Expression | ε

<DONE>

Abstract:

Life digitization with smart hardware and software have enhanced the speed of software evolution in recent decade dramatically. Thus, various software has come in our life and this has increased people expectation of software functionality and their reliability. In addition, entering critical areas such as flight control have increased the need of software reliability.
Software verification is the solitary way to be certain that a code has no fault. Hence, in recent decades, various method has developed which all require software engineering and this cause huge cost in heavy and critical tasks.
In this project, our goal is to make verification castles by reducing the required knowledge in the verification process and therefore increasing labour. Presented approach will map a piece of code and its verification process into the a puzzle (game) automatically. Since playing a game need no special knowledge, we could distribute verification process between a million game players.

Keywords: crowdSourcing, verification, game, type system

Iran University of Science and Technology
School of Computer Engineering

Crowdsource Software Verification by Converting the Verification Process Into a Game

A Thesis Submitted in Partial Fulfillment of the Requirement for the Bachelor Degree of Computer Engineering - Software

By:
Emad Aghajani

Supervisor:
Behrouz Minaei Bidgoli

September 2014