DOR :20.1001.1.2538161.1402.8.2.3.7

تابستان ۱۴۰۲، دوره ۸، شماره ۲، پیاپی ۲۹

دریافت مقاله: ۱۴۰۲/۰۳/۰۲

پذیرش مقاله: ۱۴۰۲/۰۴/۱۴

نوع مقاله: پژوهشی

رویکردی جدید جهت ترمیم نرم‌افزار مبتنی بر بازی‌های هدفمند

شرمین موسوی*

دانشجوی دکتری، دانشکده مهندسی کامپیوتر و علوم کامپیوتر، دانشگاه شهیدبهشتی، تهران، ایران

پست الکترونیکی: sharmin_moosavi@yahoo.com

مجتبی وحیدی‌اصل

استادیار، دانشکده مهندسی کامپیوتر و علوم کامپیوتر، دانشگاه شهیدبهشتی، تهران، ایران

پست الکترونیکی: mo_vahidi@sbu.ac.ir

حسن حقیقی

استاد دانشکده مهندسی کامپیوتر و علوم کامپیوتر، دانشگاه شهیدبهشتی، تهران، ایران

پست الکترونیکی: h_haghighi@sbu.ac.ir

چکیده

یکی از مراحل مهم در فرآیند تولید نرم‌افزار، ترمیم نرم‌افزار است. به دلیل مشکلاتی که روش‌های خودکار درترمیم نرم‌افزار دارند، در صنعت همچنان انسان‌ها خطاهای نرم‌افزارها را اصلاح می‌کنند. زمان و هزینه بالای فرآیند ترمیم نرم‌افزار توسط انسان، منجر شده در این مقاله، یک راهکار جدید جهت استفاده از قابلیت‌های انسان پیشنهاد شود، که در وهله اول زمان و هزینه ترمیم نرم‌افزار توسط انسان را کاهش دهد و در وهله دوم این مرحله را به فعالیتی جذاب‌تر در فرآیند تولید نرم‌افزار تبدیل نماید. روش پیشنهادی مبتنی بر جمع‌سپاری و تعامل انسان و ماشین جهت ترمیم نرم‌افزار است. یکی از راه‌کارهای موثر جهت سپردن یک مسئله به جمعیت کثیر و بهره بردن از توانایی‌های انسان با هزینه اندک، استفاده از بازی هدفمند است که از قابلیت‌های بازیکنان جهت حل مسائل جدی بهره می‌برد. در بررسی فعالیت‌های صورت گرفته در زمینه ترمیم نرم‌افزار، فقدان راهکاری جهت استفاده موثر از جمع‌سپاری و بازی هدفمند به شدت احساس شد. بنابراین ما راهکاری کاملا جدید ارائه داده‌ایم، که در آن یک بازی هدفمند جدید با هدف ترمیم نرم‌افزار طراحی شده است و با سپردن بازی به جمعیت فراوان و حل آن توسط بازیکنان، در اصل ترمیم نرم‌افزار صورت می‌گیرد.

در بازی مطرح شده به عنوان باراپاپا، چندین شخصیت وجود دارند که جهت استفاده از دستگاه‌های یک شهربازی، بایستی از مسیرها و شرط‌های آن مسیرها عبور کنند. در این بازی ممکن است که شرط‌ها و مسیرها به درستی تنظیم نشده باشد، بنابراین وظیفه بازیکنان است که آنها را به درستی اصلاح کنند تا تمام شخصیت‌ها به‌درستی از وسایل شهربازی استفاده کنند. عناصر بازی بر اساس کد مشکوک به خطا ساخته شده و حل معماهای بازی معادل با ترمیم نرم‌افزار است.

برای ارزیابی روش پیشنهادی، مراحل بازی در اختیار ۲۰ بازیکن قرار داده شده است. از سوی دیگر، کد اولیه برنامه‌های محک، برای ترمیم نرم‌افزار به پنج برنامه‌نویس

* نویسندهٔ مسئول

Email: sharmin_moosavi@yahoo.com

ارائه شـد. در مقایسـه نتایج ترمیم نرم‌افزار بین بازیکنان و برنامه نویسـان، مشـخص شـد، بازیکنان سریع‌تر از برنامه‌نویسان وصله‌های صحیح را ایجاد نمودند. به‌منظور درک میزان جذابیت و توانایی بازی در مخفی کردن مسئله فنی، پرسـش‌نامه‌هایی در اختیار بازیکن‌ها قرار گرفت که بررسی پاسخ‌ها، نشان از موفقیت بازی در این زمینه‌ها را دارد. همچنین نتایج حاصل از بازی با ابزار خودکار ترمیم نرم‌افزار مقایسه شده و می‌توان بیان کرد که بازی برتری مطلق نسبت به GenProg درترمیم کدها دارد.

واژه‌های کلیدی: ترمیم نرم‌افزار، جمع سـپاری، بازی محاسباتی مبتنی بر انسان، بازی‌های هدفمند.

۱. مقدمه

در دنیـای امروز، نرم‌افزارها در تمام سـطوح زندگی وارد شـده‌اند. با این‌حال، خطاها بخشـی جدایی‌ناپذیر از برنامه‌های نرم‌افزاری هسـتند که گاهـی اوقات منجر به ایجاد پاسـخ نادرست، یا شکسـت نرم‌افزار می‌شوند که می‌تواند خسارات جبران‌ناپذیر مالی و جانی در پی داشته باشد. اگرچه در بیشتر مراحل تولید نرم‌افزار، سعی بر آن است که تعداد خطاها کاهش داده شوند، اما همچنان تعداد قابـل توجهی از آنها در نسـخه نهایی باقی می‌مانند. حتی اگر نسخه اولیه نرم‌افزار بدون خطا باشد، تغییراتی که در نسخه‌های بعدی ایجاد می‌شـوند، می‌تواند منجر به ایجاد خطای جدید در نرم‌افزار شود.

رفـع خطاهـا و تعمیـر برنامه‌ها به صورت دسـتی، فرآیندی دشـوار اسـت. با این‌حال رفع اشکالات و ترمیم نرم‌افـزار در صنعت همچنان متکـی بر نیروی متخصص انسـانی است. انسـان مزیت‌هایی همچون قدرت تحلیل و اسـتدلال، هوشمندی، و تسلط بر کدِ نوشته شده نسبت به ماشین دارد که این مزایا سبب می‌شوند حل مسائلی نظیر ترمیم نرم‌افزار همچنان توسـط انسـان‌ها صورت گیرد. اما انسـان درترمیم نرم‌افزار با چالش‌هایی نظیر خستگی، تکرار فراوان، بزرگ و دشوار بودن مسئله مواجه است، که منجر می‌شـود این عملیات برای نیروی متخصص خسته کننده و هزینه‌بر باشد. به همین دلیل، امروزه روش‌هایی به منظور ترمیم خودکار نرم‌افزار[1] ارائه شـده‌اند که به دنبال راه‌کارهایی جهـت اصلاح خودکار خطاهـای نرم‌افزاری هسـتند و این اصلاحـات خودکار روی کـد برنامه، حتی اگر اندک باشـند، با ارزش بوده و سبب کاهش قابل توجه هزینه‌ها می‌شـوند. بیشتر روش‌های خودکار معرفی شده مبتنی بر تولید تعمیرهایی[2] برای کد مشکوک به خطا بوده و مطابق با یکی از الگوهای زیر عمل می‌کنند:

- تولیـد و ارزیابـی[3]: ایـن روش مبتنی بر یـک فرآیند تکراری تا حصول نتیجه مناسـب است. این فرآیند شامل دو مرحله اسـت: تولید تعمیرها و ارزیابی آنها. در مرحله تولید تعمیرها راه‌حل‌های نامزد ایجاد می‌شوند. در مرحله ارزیابی، درستی یا عدم درستی مجموعه راه‌حل‌های نامزد در مرحله قبل بررسـی می‌گردد. بیشتر روش‌های مبتنی بر تولید و ارزیابی، درستی یک راه‌حل را بر اساس موارد آزمونی که در دسـترس هستند، محاسبه می‌کنند. عملکرد این روش به این صورت اسـت کـه اگر یک برنامه اصلاح شـده از مجموعه راه‌حل‌های نامزد، تمام موارد آزمون را با موفقیت اجرا کند، آنگاه آن برنامه به عنوان برنامه‌ای که احتمالاً درست است، معرفی می‌شود. سپس نیاز است که برنامه ترمیم‌یافته بعداً توسط تولیدکنندگان مورد ارزیابی قرار گیرد تا تصمیم گرفته شود این تعمیر پذیرفته شود یا خیر.
- ترمیـم مبتنـی بـر معناشناسـی: در فنـون مبتنی بر معناشناسـی، مسـئله ترمیم به صورت فرمال یا ضمنی رمزگـذاری می‌شـود. زمانی‌که یک راه‌حل برای مسـئله رمزگذاری شـده پیدا شـود، ترمیم برنامه صورت گرفته است. ترمیمی که بر اساس این روش بیان شده، یک‌ترمیم صد در صد درسـت است و نیاز به مرحله ارزیابی ندارد. درسـت اسـت که ترمیم به وجود آمده با این روش کاملاً درسـت است؛ اما به این معنا نیسـت که منجر به رضایت

1- Automatic program repair (APR)
2- Fix generation
3- generate-and-validate

کامل تولیدکنندگان شـــود. به عنـــوان مثال، فرض کنید در یـــک برنامه، خطایی از نوع همزمانی وجود دارد و ترمیمی که انجام شده، مشـــکل خطای همزمانی را حل می‌کند، اما ممکن اســـت منجر به ایجاد خطای امنیت یا عملکرد شود. بنابراین همچنان نیاز اســـت که ترمیم انجام گرفته توسط تولیدکنندگان مـــورد ارزیابی دقیق‌تر قرار گیرد تا تصمیم گرفته شود این ترمیم پذیرفته شود یا خیر.

هـــر دو دســـته از فنونی کـــه معرفی شـــدند، همچنان مشکلاتی چون رفع خطاهای خاص، فضای بزرگ مسئله، صرف زمان زیاد، و عدم توانایی در رفع چندخطایی دارند. بنابراین دراین مقاله راه‌کاری پیشنهاد می‌شود که همچنان که از مزیت‌های انســـان درترمیم نرم‌افزار اســـتفاده کند، مشـــکلات راه‌کار مبتنی بر انســـان را کمرنگ نماید. هدف دیگر در این راه‌کار، استفاده همزمان از مزیت‌های کامپیوتر و نیروی انسانی است. به عبارت دیگر، دراین تحقیق برای ترمیم نرم‌افزارها به دنبال استفاده از نیروی انسانی ارزان قیمت و به تعداد فراوان هستیم و از سوی دیگر قصد داریم از روش‌های خودکار نیز برای حل قســـمت‌هایی که برای انسان، خسته‌کننده و تکراری بوده و نیاز به هوش انسانی ندارد، استفاده کنیم.

یکـــی از موثرترین راهکارها در اســـتفاده از نیروهای ارزان و به تعداد فراوان، جمع سپاری است. در زمان‌هایی که نیاز به هوش جمعی جهت حل مسئله وجود دارد، مسئله به یک جمعیت انسانی سپرده می‌شود که منجر به بهره‌وری از مزیت‌های هوش انســـانی و روش جمع‌سپاری می‌شود. یکی از روش‌های مطرح شده به هدف جمع‌سپاری، استفاده از بازی‌های هدفمند[4] است.

امروزه با گســـترش شـــبکه‌های اجتماعـــی و همچنین فراگیر شدن استفاده از ابزارهای هوشمند، انجام بازی‌های کامپیوتری تبدیل به تجربه‌ای جهان‌شـــمول شده است. در دوران حاضر، نفوذ بازی‌ها در بین نسل‌ها و افراد مختلف در حال افزایش اســـت و انسان‌ها بیش از هر زمان دیگری وقت خود را صرف بازی می‌کنند [۱]. وجود چالش فکری، یکی از اصلی‌ترین عناصر بازی و ســـرگرمی است. قدرت محاسباتی مغز انسان برای رفع چالش‌های بازی می‌تواند علاوه بر ســـرگرمی، در خدمت یک هدف ضمنی نیز قرار گیرد. به‌عبارت‌دیگر، بازیکن هنگام فکر و اســـتنتاج درباره چالش‌های بـــازی به‌طور ضمنی و بدون آنکه خود متوجه باشد، می‌تواند مسئله‌ای محاسباتی یا فنی را برای اهدافی غیر از ســـرگرمی (برای مثال، مسئله تولید داده آزمون یا مسئله ترمیم خطا) حل نماید.

در راه‌کار پیشـــنهادی در این مقاله، برای اولین بار یک بازی برای ترمیم خودکار برنامه‌ها ارائه می‌شـــود. در این بازی، یک معما براســـاس درخت نحوی انتزاعیِ یک عبارت مشکوک به خطا در برنامه ساخته می‌شود. بدین‌ترتیب که از برگ‌هایِ درختِ نحوی انتزاعی به سمت ریشه، به ازای هر گره، عنصر بازی معادل آن گره قرار داده می‌شـــود. سپس از بازیکنان درخواســـت می‌شود عناصر بازی را به‌گونه‌ای تغییر دهند که معما بازی حل شود و به این صورت، مسئله مربوط به ترمیم عبارت مشکوک به خطا حل می‌شود.

اســـتفاده از این رویکرد، برخـــی از معایب روش‌های ترمیم نرم‌افزار مبتنی بر انســـان را رفع می‌نماید. در ابتدا از آنجایی‌که شکل مســـئله دیگر تخصصی نیست، مسئله بـــرای جمعیت کثیری از انســـان‌ها، چـــه متخصص و چه غیرمتخصص، قابل‌فهم شده و می‌توان از قدرت محاسباتیِ تعداد زیادی از انسان‌ها (به شکل کم‌هزینه و یا حتی بدون هزینه) برای حل مسئله استفاده نمود. بدین‌ترتیب می‌توان با اســـتفاده از جمع‌ســـپاری[5] و هوش جمعی، مســـائل با اندازه‌های بزرگ‌تر را در زمـــان کوتاه‌تر و با هزینه کم‌تر (و گاهی بدون هزینه) حل نمود. همچنین مشکل نبودِ انگیزه افراد برای ترمیم دستی، با تغییر شکل مسئله به یک بازی جذاب، مرتفع می‌گردد. بنابراین مشـــکلات عمده تولید داده آزمـــون و ترمیـــم نرم‌افزار توســـط متخصصین برطرف می‌گردد.

علاوه بر مزیت‌هایی که اســـتفاده از بازی‌های هدفمند در حوزه‌ترمیم نرم‌افزار دارد، با چالش‌ها و محدودیت‌هایی

4- Game With A Purpose (GWAP)

5- Crowdsourcing

مواجه هستیم و سؤالات زیر مطرح می‌شوند:

۱. چگونه می‌توان یک بازی هدفمند، طراحی و پیاده‌سازی نمود که در عین جذابیت، به ترمیم نرم‌افزار کمک کند؟

۲. چگونه می‌توان بازی مذکور را به نحوی طراحی نمود که بازیکن متوجه هدف فنیِ پشت بازی نشود و به‌عبارت دیگر، یک بازیکن غیرفنی که اساساً با زمینه تولید نرم‌افزار آشنایی ندارد، بتواند بازی را به‌سادگی انجام دهد؟

۳. آیا ترمیم نرم‌افزار برای یک برنامه خاص از طریق انجام بازی توسط بازیکن(های) غیرفنی در مقایسه با ترمیم نرم‌افزار بر مبنای تحلیل و بازبینی برنامه توسط برنامه‌نویس در زمان کمتری انجام می‌شود؟

٤. آیا ترمیم نرم‌افزار برای یک برنامه خاص از طریق انجام بازی توسط بازیکن(های) غیرفنی در مقایسه با ترمیم نرم‌افزار بر مبنای روش‌های خودکار دارای عملکرد بهتری است؟

در ادامه این مقاله ابتدا به بیان پیش‌زمینه‌های مورد نیاز جهت مطالعه مقاله پرداخته می‌شود. سپس کارهای انجام شده در زمینه ترمیم نرم‌افزار و بازی‌های هدفمند بررسی می‌گردند. در بخش اصلی این مقاله طرح بازی پیشنهادی جهت ترمیم نرم‌افزار بیان می‌شود و سپس ارزیابی بازی پیشنهادی صورت می‌گیرد. در انتها در مورد نتایج به دست آمده بحث می‌شود.

۲. پیش زمینه

با توجه به محدود بودن فضای مقاله در این بخش فقط به بررسی روش مکان‌یابی خطا مبتنی بر طیف برنامه که در این مقاله استفاده شده است، پرداخته می‌شود.

۱,۲. مکان‌یابی خطا

همانگونه که در بخش قبل بیان شد، اشکال‌زدایی به معنی مکان‌یابی و ترمیم خطا است. در روش پیشنهادی در این مقاله، تمرکز برترمیم نرم‌افزار است و جهت مکان‌یابی خطا از یکی از روش‌های معروف در این حوزه که مبتنی بر طیف برنامه است، استفاده می‌شود. در ادامه، این روش به صورت مختصر شرح داده می‌شود.

یکی از ابتدایی‌ترین و معروف‌ترین روش‌های مبتنی بر طیف برنامه، در [۲، ۳، ٤] مطرح شده است. نویسندگان مقاله سه روش اشتراک مجموعه، اجتماع مجموعه و نزدیک‌ترین همسایگی را برای مقایسه یک طیف اجراییِ ناموفق با چند طیف اجراییِ موفق، به هدف یافتن کد مشکوک به خطا ارائه کرده‌اند. پیاده‌سازی و فهم الگوریتم‌های بیان شده نسبتاً ساده بوده و ایده‌های مطرح شده در آن‌ها پایه و اساس تحقیقات بیشتر در مقوله خودکارسازی فرآیند مکان‌یابی خطای برنامه‌ها قرار گرفته‌اند.

یکی از متداول‌ترین معیارهای امتیازدهی خطا به جملات برنامه در [۳] با نام تارانتولا ارائه شده است. با در اختیار داشتن ماتریس طیف‌های برنامه، متریک تارانتولا به نام hue میزان سالم بودن (عدم ارتباط با خطا) جمله s را به صورت زیر بیان می‌کند (معادله ۱):

$$hue(s) = \frac{\frac{passed(s)}{totalpassed}}{\frac{passed(s)}{totalpassed} + \frac{failed(s)}{totalfailed}} \quad \text{(معادله ۱)}$$

در رابطه بالا، منظور از passed(s) و failed(s)، به ترتیب تعداد اجراهای موفقی و ناموفقی است که جمله s در آن‌ها اجرا شده و totalpassed (totalfailed) تعداد کل اجراهای موفق (ناموفق) است. در حقیقت totalpassed و totalfailed به ترتیب تعداد طیف‌های موفق و ناموفق در ماتریس طیف‌های برنامه هستند که جمله s در آن‌ها دارای مقدار یک است (پوشش داده شده است). همان‌طور که از این رابطه مشخص است، hue(s) مشخص‌کنندهٔ میزان حضور s در طیف‌های موفق نسبت به حضور آن در همه طیف‌های برنامه می‌باشد. بدیهی است که هرچه این مقدار بیشتر باشد، احتمال وجود خطا در جمله s کمتر است.

۳. کارهای انجام شده

در این بخش، ابتدا چند نمونه از کارهای معروف

صورت گرفته در حوزه ترمیم نرم‌افزار معرفی می‌شوند. سپس در ادامه بازی‌های هدفمند بررسی می‌گردند و به دلیل اینکه ما موفق به یافتن بازی هدفمند در حوزه‌ترمیم نرم‌افزار نشدیم، در ادامه این بخش به بررسی چند بازی هدفمند در حوزه تولید نرم‌افزار می‌پردازیم.

۳-۱- ترمیم نرم‌افزار

ترمیم خودکار نرم‌افزار پیشینه طولانی ندارد، با این حال به دلیل اهمیت بالایی که در فرآیند تولید نرم‌افزار و به خصوص اشکال‌زدایی نرم‌افزار دارد، مورد توجه پژوهشگران قرار گرفته است. تاکنون روش‌های مختلفی نیز در این حوزه، ارائه شده است و در این بخش به بیان چند نمونه از کارهای مهم انجام شده در این زمینه پرداخته خواهد شد.

کیم و همکارانش در مقاله خود اولین راه‌کار برای ترمیم خطا را ارائه داده‌اند [۵]. کیم معتقد است که خطاهای برنامه‌نویسان اغلب از الگوی خاصی پیروی می‌کنند که با شناسایی این الگوها می‌توان آنها را به طور خودکار ترمیم نمود. در این مقاله، ده خطایی که به طور معمول در برنامه‌ها رخ می‌دهند اما برنامه‌نویسان به آنها توجهی نمی‌کنند، شناسایی شده و برای هر یک روش ترمیمی ارائه شده است که به الگوهای پار معروف هستند. اگر چه به روش او ایرادات اساسی وارد است، همچنان جزء مقالاتی است که در حوزه‌ترمیم خودکار نرم‌افزار مورد توجه قرار گرفته‌اند. پس از آن که الگوهای پار معرفی شدند، دیگر محققان به تولید هر یک از این الگوها پرداختند و روش‌های جدیدی برای شناسایی خطا و ترمیم آن پیشنهاد داده‌اند [۶، ۷]. اما همچنان این روش‌ها فقط قادر به رفع تعداد محدودی از خطاها بودند. همچنین از مشکلاتی چون بزرگ بودن فضای جستجو، وجود چند خطایی رنج می‌برند.

گروهی دیگر از فنون ترمیم خطا، با جستجو در فضای مسئله سعی می‌کند وصله محتملی را بیابد که به ازای آن، موارد آزمون با موفقیت اجرا شوند [۸، ۹، ۱۰]. در سال ۲۰۰۹، ویمر و همکارانش الگوریتم GenProg را معرفی کردند [۸]. این روش سعی دارد با استفاده از الگوریتم ژنتیک و با جستجو در فضای مسئله، وصله محتمل را بیابد. این الگوریتم ابتدا با ایجاد جهش تصادفی، جمعیت اولیه‌ای را به وجود می‌آورد. بدیهی است که هر نمونه از جمعیت، یک وصله نامزد محسوب می‌شود. در مراحل بعد وصله‌ها مورد ارزیابی قرار گرفته، درصورتی که وصله محتمل یافت نشود، ایجاد جهش و ارزیابی تکرار می‌گردد. الگوریتم سعی می‌کند تا با ایجاد جهش در نمونه‌های مناسب، وصله محتمل را تولید کند. این کار تا زمانی تکرار می‌شود که یا یک جهش محتمل یافت شود، یا محدودیت زمانی به پایان برسد. چالشی که همواره روش‌های مبتنی بر الگوریتم ژنتیک از آن رنج می‌برند، نیاز به مجموعه داده بسیار بزرگ بابت جستجو و شناسایی وصله مناسب است و این چالش در روش‌های ژنتیک قابل حل نیست.

در سال‌های بعد از ۲۰۰۹، دیگر الگوریتم‌های مبتنی بر جستجو برای تولید وصله به کار گرفته شدند [۱۰، ۱۱]. همچنین در سال ۲۰۱۲، الگوریتم GenProg توسط ارائه دهندگان آن بسط داده شد و موفقیت خوبی نسبت به نسخه اولیه داشت. آن‌ها برای ارزیابی الگوریتم خود از برنامه‌های که به زبان C نوشته شده بود استفاده کردند. بسط الگوریتم GenProg توانست بیش از نیمی از خطاهای موجود در برنامه را ترمیم کند. اما همچنان با چالش‌هایی چون فضای بزرگ مسئله، مقیاس‌پذیری و مقدار پایین نرخ ترمیم مواجه هستند.

دسته دیگر رویکردها، ترمیم خودکار مبتنی بر روش جستجوی وصله با به کارگیری الگوریتم‌های مبتنی بر جستجو است. در این دسته از الگوریتم‌ها، خطی از کد مورد هدف قرار می‌گیرد و الگوریتم بکارگرفته شده سعی می‌کند، با ایجاد تغییر در کد مبدا، وصله محتمل را تولید کند. هر وصله نامزد توسط موارد آزمون و تابع شایستگی مورد ارزیابی قرار می‌گیرد [۱۲]. تعداد زیادی از الگوریتم‌های جستجو وجود دارند که می‌توان از آنها برای تولید وصله محتمل استفاده نمود. آنچه که در به کارگیری الگوریتم‌ها برای تولید وصله حائز اهمیت است، بحث زمان است. به

عبـــارت دیگر کدام الگوریتم در مدت زمان کمتر به پاســـخ دست می‌یابد. به طورکلی الگوریتم‌های مبتنی بر جستجو، به دو دســـته کلی الگوریتم‌های سراسری و الگوریتم‌های محلی دســـته‌بندی می‌شـــوند [۱۳]. در این میان، الگوریتم مِمتیک [۱۳] با تلفیق الگوریتم‌های جســـتجوی سراسری و الگوریتم‌های جســـتجوی محلی، سعی دارد در مدت زمان کمتری به جواب برســـد. در تمام این رویکردها مشکلاتی چون نحوه انتخاب تابع شایســـتگی، نیـــاز به تعداد فراوان مورد آزمون، فضای بزرگ جســـتجو و ایجاد وصله‌های بیش برازش شده، را دارند.

اگـــر روش‌های ترمیم خطا را بـــا توجه به زمان ترمیم دسته‌بندی کنیم، به دو دسته ترمیم پویا و ایستا می‌رسیم.

در روش‌هـــای پویـــا، ترمیم نرم‌افـــزار در حین اجرای برنامه انجام می‌شود. هدف اصلی از این کار، جلوگیری از توقف اجرای نرم‌افزار می‌باشد. روند کار این گونه است که اگر خطایی در هنگام اجرای دســـتورات رخ داد، به گونه‌ای از این خطا چشم‌پوشی شود و اجرای دستورات ادامه یابد. هنگام رخ دادن خطا، می‌توان کنترل برنامه به قسمتی دیگر منتقل شود و در آنجا خطای رخ داده شده بررسی و کنترل شود و سپس اجرای دستورات قبلی ادامه یابد. دو راهکار معروف در این زمینه، تزریق کد و داده ساختار ترمیم‌پذیر است [۱٤].

در روش‌های ترمیم ایســـتا، در هنگام تولید سیســـتم بـــه ترمیم خطای آن می‌پردازند. ترمیم ایســـتا خود دارای روش‌ها و دسته‌بندی‌های مختلفی است. ترمیم‌های مبتنی بر جســـتجو مانند الگوریتـــم GenProg همگی جزء ترمیم ایستا هستند. زیرا، در هنگام تولید نرم‌افزار سعی در ترمیم خطای موجـــود در آن دارند. یکی دیگـــر از الگوریتم‌های ترمیم ایستا DeepFix نام دارد [۱۵]. این روش سعی دارد خطاهـــای نحوی را که به دلیل بی‌دقتی برنامه‌نویس در کد وجود دارد، به طور خودکار ترمیم کند. این درحالی اســـت که کامپایلرهای امروزی گاه در هنگام تشخیص محل خطا دچار اشـــتباه شده و محل دیگری را به عنوان محل خطای نحوی گزارش می‌کنند. برای مثال اگر در قسمتی از کد نویسهٔ "}" برای بستن بلوک شـــرطی فراموش شود، کامپایلر به اشتباه آخرین خط کد را محل بستن "{" تشخیص می‌دهد. در روش DeepFix به کمـــک یادگیری، چنین خطاهایی به طور خودکار تشـــخیص داده شده و ترمیم می‌گردند. تمام این رویکردها با چالش‌های بزرگی چون زمان اجرا و نرخ ترمیم پایین و وصله‌های بیش‌برازش شده مواجه هستند، به این دلیل همچنان نیازمند متخصصین در حوزه هستند تا هم درســـتی وصله‌های ایجاد شده را بررسی کنند و هم برای خطاهایی که وصله‌ای برای آنها ایجاد نشـــده است، وصله مناســـب را ایجاد کنند. اما در رویکرد ما چون هدف اســـتفاده از نیروی انســـانی فراوان و هوش جمعی است، بسیاری از مشـــکلات روش‌های خودکار چون زمان اجرا و نرخ ترمیم پایین، را دارا نیست.

۳-۲- بازی هدفمند و تولید نرم‌افزار

یک دســـته از بازی‌ها، بازی‌های جدی هستند که هدف این دسته فراتر از ســـرگرمی است. در این دسته، طراحی بازی شـــامل تمـــام عناصر و جزئیات یـــک بازی معمولی است. ترکیب بازی‌ها با محاســـبات مبتنی بر انسان منجر به ظهور بازی‌های محاسباتی مبتنی بر انسان یا بازی‌های هدفمند شده است که زیرمجموعه‌ای از بازی‌های جدی را تشـــکیل می‌دهند [۱٦، ۱۷]. اخیرا بازی‌های جدی و هدفمند در زمینه های مختلف مهندســـی نرم افزار مورد اســـتفاده قرار گرفته‌اند. در ادامه برخـــی از این بازی‌ها به اختصار توضیح داده شده‌اند.

سیم ســـی[6] [۱۸] یک بازی هدفمند کامپیوتری است که آموزش مهندســـی نرم‌افزار را به هـــدف درک و مدیریت فرآیند تولید نرم افزار به دانش‌آموزان شبیه‌سازی می‌کند. در این بازی، یک پروژه نرم‌افزاری شبیه‌ســـازی شده که در آن هر بازیکن، نقش یک مدیر پروژه را دارد که موظف است تولیدکنندگان را برای انجام وظایف و فعالیت‌هایشان مدیریت نماید.

پِکس فورفان[7] [۱۹] یکی دیگر از بازی‌های جدی مبتنی

6- SimSE
7- Pex4Fun

بر وب برای آموزش برنامه‌نویسی به دانشجویان کامپیوتر است. این پروژه، دانشـــجویان را تشویق می‌کند تا یک کد برنامه مفروض را در مرورگرهای مختلف ویرایش نموده و این کد سپس برای اجرا، تجزیه و تحلیل به موتور پکس فور فان داده می‌شود.

کـد هانت[۸] [۲۰] یک افزونه بـــرای پکس فور فان و یک بن‌ســـازهٔ مبتنی بر بازی جدی اســـت که هدف آن، تمرین مهارت‌های برنامه‌نویسی است. کدهانت در واقع یک موتور اجرای جعبه سفید نمادین است که معماهایی از جمله موارد آزمـــون را به بازیکنان ارائه می‌دهـــد و آنها باید برنامه‌ای بنویسند که تمام موارد آزمون داده شده را تایید کند.

تأیید رسمی[۹] یک حوزهٔ مهم در مهندسی نرم‌افزار است که می‌تواند قابلیت‌های بازی را برای بهبود اطمینان‌پذیری نرم‌افزار به کار گیرد. تأیید رسمی برای مدل‌سازی کد منبع و همچنیـــن اثبات ویژگی‌های موجود در برنامه اســـتفاده می‌شود. اگرچه اثبات ویژگی‌ها معمولاً به صورت خودکار انجام می‌پذیرند، اما مدل‌سازی رسمیِ کد منبع، یک فعالیت پرهزینه است که به میزان قابل‌توجهی به مشارکت انسانی نیـــاز دارد. در ادامه به برخی از فعالیت‌هایی که برای تأیید رسمی پیشنهاد شده، اشاره می‌کنیم.

زایلم[۱۰] یک بازی هدفمند جهت تایید رسمی است که در آن بازیکنان با رشـــد گیاهان عجیب و غریب درگیر هستند [۲۱]. این بازی از بازیکنان می‌خواهد که عوامل رشد گل‌ها را مدل‌ســـازی کنند، که این عوامـــل در اصل نامتغیرهای حلقه‌های برنامه هستند. پس از مطالعه مقادیر رشد گل‌ها، از بازیکنان خواسته می‌شود تا معادلات جبری برای رشد گل‌ها بنویسند. این معادلات، مشخصات نامتغیرهای حلقه‌ها هستند. چالش اساســـی در طراحی بازی، پنهان کردن کد منبع برنامه هدف از بازیکنان است.

در پایپ جم[۱۱][۲۲]، هدف این اســـت که هزینه تأیید نرم افـــزار کاهش یابد. در این بازی، فرآینـــد تأیید نرم‌افزار به یک معما تبدیل شـــده و از بازیکنان خواســـته می‌شود که معماهای حاصل را حل کنند. پس از آن، راه‌حل‌ها به اثبات صحت تبدیل می‌شـــوند. محدودیت‌ها در انـــواع برنامه و وابستگی‌های بین نامتغیرهای برنامه با روابط بین عناصر بازی نشان داده می‌شوند.

فاوا و همـــکاران [۲۳] رویکرد دیگری برای اســـتفاده از بازی هوشـــمند به هدف بهبود تشـــخیص نامتغیرهای برنامه ارائـــه کردند. این رویکرد تشـــخیص خصوصیات نامتغیر، برنامه را به یک بـــازی کامپیوتری تبدیل می‌کند، که به آن باینزی فیژن[۱۲] می‌گویند. در این بازی که براساس استخراج پیش شرط‌ها است ، بازیکن به عنوان یک رده‌بند عمل می‌کند و با استفاده از فیلترها در یک نمایش گرافیکی، خصوصیات نامتغیر را شناســـایی می‌کنـــد. در این بازی فرض بر این اســـت کـــه بازیکنان با روش‌های رســـمی و برنامه نویسی کامپیوتری آشـــنا هستند و از قصد واقعی بازی مطلع هستند.

پوجـــاس و همکاران [۲۴] یک بازی بـــه نام کد دِفندِر[۱۳] معرفی کردند که از عناصر بازی در فرآیند آزمون استفاده می‌کنـــد. این بازی، دانشـــجویان را به شـــیوه‌ای رقابتی و سرگرم‌کننده برای انجام آزمونِ جهش درگیر می‌کند و در عین حال، ســـرگرمی را به تجربه یادگیری اضافه می‌کند. در ایـــن بازی، بازیکنان بـــه عنوان مهاجم یـــا مدافع عمل می‌کنند. مهاجمان، جهش‌ها را در یک کد منبع تولید می‌کنند و مدافعان، آزمایش‌های واحدی را برای کشـــتن جهش‌ها ایجاد می‌کنند.

نینگ چن و ســـونگون کیـــم [۲۵] یک محیـــط آزمون خودکار مبتنی بر معما (PAT) را برای تجزیه مسائل جهش اشـــیاء و حل محدودیت‌های پیچیده بـــه معماهای کوچک پیشـــنهاد کردند که این معماها توســـط انسان حل شوند. نتایج ارزیابی نشـــان داد که انســـان‌ها می‌توانند مشکلات را بـــه طور موثر حل کنند. آن‌ها همچنین نشـــان دادند که انسان‌هایی که مهارت‌های برنامه‌نویسی یا دانش حوزه‌ای از موضوعـــات را ندارند، می‌تواننـــد معماهای PAT را حل کنند. سیســـتم پیشنهادی صرفاً مبتنی بر حل معما بوده و

8- Code Hunt
9- Formal verification
10- Xylem, The Code of Plants
11- Pipe Jam
12- Binary Fission
13- Code Defenders

نقش عناصر بازی در آن پررنگ نبود. علاوه بر این، محیط بازی بیشتر شبیه به یک سیستم بازی‌گونه شده است تا این که مبتنی بر بازی هدفمند باشد.

اما شبیه‌ترین بازی‌ها به روش پیشنهادی ما بازی‌های رینگز و گرینیفای هستند که در هر دو بازی، هدف، تولید داده آزمون نرم‌افزار است. در این بازی‌ها مسیرهای گراف جریان کنترلی برنامه با عناصر بازی به بازیکنان نشان داده می‌شوند و بازیکنان با حل معماهای بازی در اصل، داده‌های آزمون را تولید می‌کنند.

در بازی رینگز [۲۶]، هدف، تولید داده‌های آزمایشی برای واحدهای برنامه براساس اجرای نمادین است. در این بازی، فرض بر این است که در هر معما، یک مسیر مشخص از گراف جریان کنترلی یک برنامه واحد توسط شبکه‌ای از لوله‌ها به بازیکنان نمایش داده می‌شود. در ورودیِ سیستم لوله‌ها، چندین حلقه است که می‌توانند در شبکه لوله سقوط کنند. لوله‌ها حاوی چندین فیلتر هستند که نمایشی از محدودیت‌های برنامه هستند و حلقه‌ها شبیه پارامترهای ورودی کد منبع هستند. اگر ویژگی‌های حلقه‌ها (به عنوان مثال، اندازه، شکل یا رنگ) به درستی تنظیم نشود، حلقه‌ها نمی‌توانند از فیلترها عبور کنند. هنگامی که یک بازیکن یک معما را حل می‌کند، به طور ضمنی مقادیر مناسبی را برای برآورده شدن محدودیت مسیر مربوطه ایجاد می‌کند. این بدان معنی است که بازیکن در حال تولید مقادیری است که ممکن است منجر به اجرای مسیر انتخابی از گراف کنترلی یک برنامه واحد شود.

در گرینیفای [۲۷]، مسیرهای گراف جریان کنترلی یک برنامه واحد از طریق رشته‌ای از لامپ‌های نور متصل نشان داده می‌شوند و پارامترهای ورودی واحد از طریق اسلایدرها و چک‌باکس‌ها نشان داده می‌شوند. بازیکنان بایستی مقادیر اسلایدرها و چک‌باکس‌ها را تغییر دهند تا زمانی که همه لامپ‌های سبز در یک رشته، روشن شوند. نتایج نشان داد که گرینیفای از رینگز، رویکرد مبتنی بر انسان و روش تولید داده‌های آزمون به صورت تصادفی از نظر درصد پوشش مسیر و زمان سپری شده برای تولید داده‌های آزمون، عملکرد بهتری دارد. اما همچنان با چالش‌هایی مواجه بود که در کوته [۳۰] این چالش‌ها با بیان داستانی جدید برطرف شد. کوته گسترده‌ترین بازی به هدف تولید داده آزمون است که تمام چالش‌های رینگز و گرینیفای را برطرف کرده است.

تمام بازی‌های مطرح شده در حوزه خود، بازی‌های موفقی بودند و نشان از توانایی بازی و بازیکنان در حل مسائل جدی را دارد. علی‌رغم قابلیت‌های بازی هدفمند و با وجود این که از این نوع بازی‌ها در مراحل مختلف از تولید نرم‌افزار استفاده شده است، اما در بررسی‌های صورت گرفته مشخص شد که بازی هدفمندی جهت ترمیم نرم‌افزار طراحی نشده است. بنابراین در این مقاله به دنبال رویکردی بابت ترمیم نرم‌افزار مبتنی بر بازی‌های هدفمند هستیم.

۴- طرح بازی به هدف ترمیم نرم‌افزار

در طراحی بازی بایستی دو دیدگاه به دقت بررسی شود: ۱) دیدگاهی که به حل مسئله ترمیم و چگونگی ایجاد ارتباط بین مسئله ترمیم و بازی می‌پردازد و ۲) دیدگاهی که به جذابیت بازی و جریان بازی[14] می‌پردازد؛ زیرا اگر بازی به اندازه کافی جذاب نباشد، نمی‌تواند بازیکنان را متعاقد کند که در انجام بازی مشارکت کنند.

در طراحی این بازی، ما به دنبال تعامل بین سیستم و کاربر هستیم، به گونه‌ای که بعضی مراحل را سیستم و بعضی دیگر را کاربران انجام دهند. بنابراین از دیدگاه حل مسئله ترمیم، بازی پیشنهادی دارای چندین مرحله است که در شکل (۱) نشان داده شده است. این مراحل تعامل بین انسان و کامپیوتر را به خوبی نشان می‌دهند. بعضی مرحله‌ها که انجام آن برای کامپیوتر دشوار است، به بازیکنان سپرده می‌شود و بعضی مرحله‌ها که برای انسان خسته‌کننده است به سیستم سپرده می‌شوند.

۱. مکان‌یابی خطا توسط سیستم: در این مرحله از روش‌های مکان‌یابی خطا به صورت خودکار استفاده شده

14- game flow

شکل ۱: مراحل بازی پیشنهادی

است. یکی از روش‌های معروف در این زمینه یک روش آماری موسوم به تالانتولا است (بخش ۲-۱) که در این مقاله از این راهکار، جهت پیدا کردن مکان‌های مشکوک به خطا استفاده شده است.

۲. نمایش کد مشکوک به خطا با کمک عناصر بازی: قسمت اصلی و چالش برانگیز در طراحی بازی، طراحی این مرحله است. در این مرحله، عناصر مشکوک به خطا در برنامه بایستی با کمک عناصر بازی بسته‌بندی شده و به بازیکنان نشان داده شود. عناصر بازی بایستی به گونه‌ای طراحی شوند که برای همه افراد، بدون داشتن تخصص در زمینه کامپیوتر، قابل درک باشد. بنابراین یکی از بخش‌های اصلی، طراحی عناصر بازی و انتقال عناصر برنامه به عناصر بازی است (بخش ٤-۲-۲).

۳. توزیع بازی بین بازیکنان: یک راه مناسب، انجام بازی به صورت گروهی است. بدین گونه که یک سطح از بازی به یک گروه داده شده و اولین نفری که به جواب صحیح دست یافت، بالاترین امتیاز را می‌گیرد (رقابت بین اعضای گروه). از سوی دیگر، می‌توان بین گروه‌ها نیز مسابقه برگزار نمود. در این حالت، چند سطح از بازی بین دو گروه توزیع شده و گروهی که بیشترین سطح را حل کند، برنده اعلام می‌شود (رقابت بین گروه‌ها). مزیت توزیع بازی در گروه‌ها افزایش حس رقابت و یادگیری روش‌های حل مسئله از هم گروهی‌ها است.

٤. پیشنهاد قالب مناسب جهت ترمیم کد توسط بازیکنان: در این مرحله، بازیکن به بازی کردن می‌پردازد و سعی در حل مسئله دارد. در این مرحله، بازیکن از بین عناصر بازی، عنصری که تشخیص می‌دهد عامل بروز خرابی برنامه است را انتخاب می‌کند. آنگاه بازی چندین قالب مرتبط با عنصر مورد نظر را جهت اصلاح آن پیشنهاد می‌دهد. در این مرحله، وظیفه بازیکن این است که بهترین قالب را جهت اصلاح بازی انتخاب کند (بخش ٤-۲-٤).

۵. ایجاد تمام حالت‌های ممکن برای قالب پیشنهادی بازیکن، توسط سیستم: قالب پیشنهادی را بازیکن انتخاب می‌کند، اما وارسی تمام حالت‌های ممکن برای قالب پیشنهادی برای یک انسان، عموماً خسته‌کننده است، بنابراین، این وظیفه به سیستم سپرده می‌شود. به عنوان مثال، بازیکن اعلام می‌کند متغیر A بایستی عوض گردد، در اینجا دیگر این وظیفه بر عهده سیستم است که تمام متغیرهای موجود در حوزه کد خطادار را جایگزین متغیر A کند و نتیجه را به بازیکن اعلام نماید.

٦. اجرای موارد آزمون و بررسی تایید شدن تمام موارد آزمون: وظیفه دیگری که بر عهده سیستم است اجرای تمام موارد آزمون برای کد اصلاح شده است. در این بخش، سیستم بررسی می‌کند که آیا با توجه به تغییرِ اعمال‌شده، تمام موارد آزمون تایید شده‌اند یا خیر. اگر تمام موارد آزمون به درستی اجرا شده باشند، آنگاه بیان می‌شود که بازیکن برنده است و تغییرات در بازی به برنامه اعمال شده و به عنوان کد ترمیم شده، ثبت می‌گردد. در صورت درست اجرا نشدن موارد آزمون، موردهای درست اجرا نشده به بازیکن نشان داده شده و از او خواسته می‌شود مجدد تلاش کند.

٤-۱- داستان بازی

داستان‌سرایی و ایجاد هدف برای بازیکنان جهت انجام بازی، در ایجاد جذابیت و جریان بازی بسیار تاثیرگذار است. در این پروژه علاوه بر جذابیت، داستان بازی این کمک را به ما می‌کند که عناصر برنامه را در قالب داستان و عناصر بازی مخفی نماییم. به این منظور، برای این بازی یک داستان در نظر گرفته شده که عناصر بازی براساس داستان تعریف شده‌اند.

داستان این بازی در محیط یک شهربازی انجام

می‌شـود که شامل چندین دسـتگاه بازی است و استفاده از هـر دسـتگاه، قوانینی، مانند محدودیـت در قد و وزن، دارد. مشکلی که وجود دارد این است که هنگام پیاده‌سازیِ قوانین با کمک ابزارهـا، خطاهایی به وجود آمده و نتیجه آن این اسـت که به افرادی که نباید، اجازه ورود داده شده و یـا برعکس به افراد مجاز، اجازه ورود داده نمی‌شـود. خانواده کارتونی باربا‌پاپا به این شـهربازی آمده‌اند و با این مشکل روبه‌رو می‌شوند و حال آنها با ایجاد تغییراتی در ابزارها سـعی می‌کنند این خطاها را برطرف کنند تا که همه دستگاه‌ها به درستی کار کنند. ما این بازی را به دلیل شـخصیت‌های باربا‌پاپایی موجود در بـازی، «باربا‌پاپا» نامیده‌ایم.

٤-۲- نمایش عناصر کد مشـکوک به خطا با کمک عناصر بازی

طراحـی مراحل بازی چند مرحله مهـم دارد که اولین مرحله از آن، تطابق عناصر کد مشکوک به خطا به عناصر بازی اسـت. این فاز یکی از مهمترین مراحل طراحی بازی است. در این بخش، ابتدا ارتباط بین عناصر کد و بازی بیان می‌شود، سـپس نحوه نمایش یک عبارت مشکوک به خطا بررسی می‌شود.

۴-۲-۱. ارتباط بین عناصر بازی و عبارت مشکوک به خطا

عبارت مشـکوک به خطا شـامل عناصری اسـت که هـر کدام از آنها بایسـتی به صورت جداگانه بررسـی و بـه یک عنصر در بازی ارتباط داده شـوند. در جدول (۱) عنصرهای عبارت مشکوک به خطا و عناصر بازی معادل با آنها نمایش داده شده است.

۴-۲-۲. نحوه نمایش یک عبارت

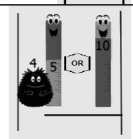
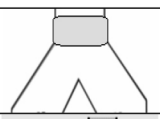
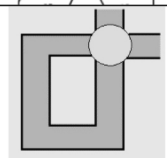
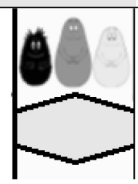
بـرای نمایـش یک عبارت بـا کمک عناصـر بازی از درخت نحوی انتزاعیِ آن استفاده می‌شود. در مسیر بازی باربا‌پاپا، ابتدا برگ‌های درخت نحوی انتزاعی نشـان داده می‌شود. سـپس از برگ به سمت ریشه درخت که حرکت کرده می‌شـود، هر گره داخلـی در درخت نحوی انتزاعی، به صـورت یک بلوک از کل مسیرنشـان داده می‌شـود. عناصری که در هر بلوک اسـت، معادل با دستور مربوطه در برنامه اسـت. به طور معمول در هر بلوک سـه عنصر بـازی وجـود دارد که متناظر با یک عملونـد و دو عملگر مربوطه در برنامه است.

جهـت بیـان بهتر مسـئله به بررسـی مثـال x+y>8 می‌پردازیم. درخت نحوی انتزاعی معادل با این عبارت در شـکل (۲-الف) نشان داده شده است. در تبدیل این درخت نحوی انتزاعی به بازی، به این صورت عمل می‌گردد که در ابتدای مسیر (بلوک اول) به ازای متغیر x و y، دو شخصیت باربابابایی قرار داده می‌شـود و به ازای عملگر جمع، نماد معادل آن قرار داده می‌شـود. گره بعد عملگر مقایسـه‌ای اسـت که با نماد متر نشـان داده می‌شود و دو عملوند آن (عـدد ۸ و حاصل بلوک قبل) در کنار نماد متر نمایش داده می‌شوند (شکل ۲-ب). در انتهای مسیر یک بازی قرار دارد که اگر بازیکـن همه عناصر را به درسـتی تنظیم کند، به شخصیت‌های مجاز اجازه بازی داده می‌شود.

۴-۲-۳. نحوه انجام بازی

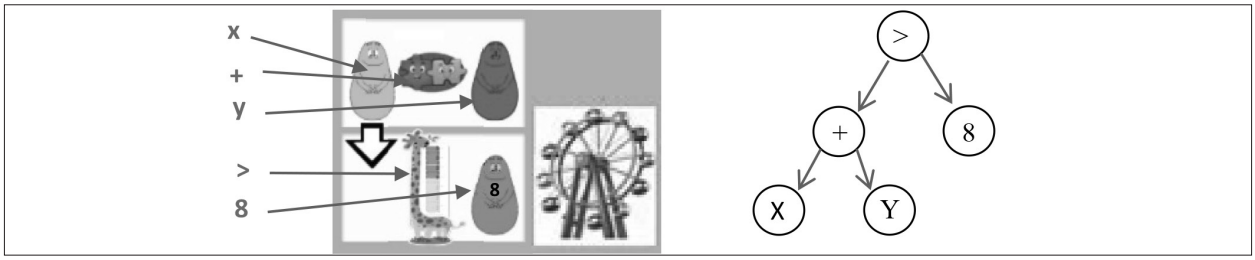
بـرای انجام بازی توسـط بازیکنان، فقـط نمایش کد مشـکوک به خطا توسط عناصر بازی کفایت نمی‌کند. بلکه بازیکنـان نیاز به راهنمایی‌هایی دارنـد که با کمک آنها به سرعت بتوانند دلایل به‌وجود آمدن خطا را تشخیص دهند و ترمیم‌هـای لازم را انجام دهند. به این هدف ما از موارد آزمون و پیشگوی آزمون آنها جهت راهنمایی به بازیکنان استفاده می‌کنیم. روال این گونه است که به ازای هر مورد آزمـون، یک نمونه بازی اجرا شـده و نتیجه، درسـت یا نادرسـت بودن آن نشان داده می‌شود. به عبارتی دیگر به ازای یک سـری مقادیر داده شـده به وزن شخصیت‌های باربا‌پاپا، آنها در مسیر بازی شروع به حرکت می‌کنند و در انتها مشخص می‌شود که اجازه استفاده از دستگاه بازی را دارند یا خیر. اگر مجوز آنها برای استفاده از دستگاه بازی درست باشد، آنگاه آن بخش سبز می‌شود و اگر نادرست باشـد قرمز می‌شود. بنابراین بازیکن با توجه به اجراهای نشان داده شـده به ازای وزن‌های مختلف شخصیت‌های باربا‌پاپـا، تغییراتی در عناصر بازی ایجاد می‌کند و نتیجه این تغییرات را نیز که در اجراها قابل مشاهده است، می‌بیند.

جدول ۱: عناصر برنامه و عناصر بازی معادل با آن

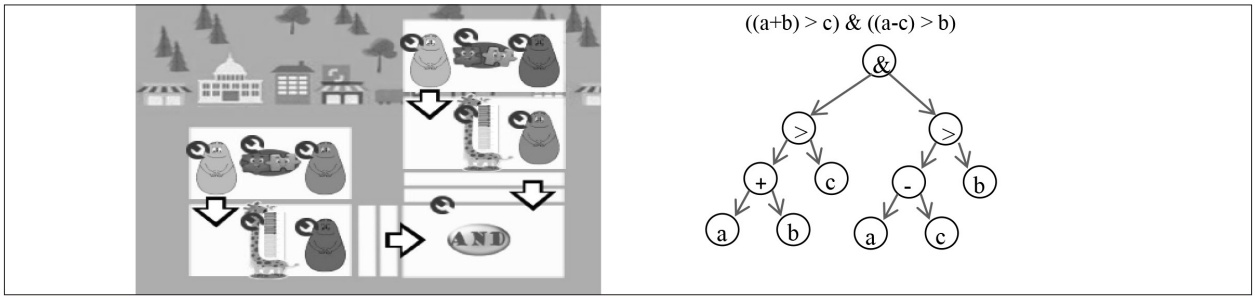
	عنصر در عبارت مشکوک به خطا			عنصر در بازی	توضیحات
۱	متغیرها و لیترال‌ها				هر متغیر و لیترال در کد مشکوک به خطا به وسیله یک شخصیت باربابا‌پا نمایش داده می‌شود که قد شخصیت، مقدار لیترال یا متغیر را نشان می‌دهد.
۲	عملگرها	عملگرهای اصلی ریاضی	جمع		برای نمایش عملگرهای جمع و تفریق از شکلک‌هایی که نشان‌دهنده الحاق و انفصال هستند، استفاده می‌گردد.
			تفریق		
			ضرب		برای نمایش عملگرهای ضرب و تقسیم از نمادهای چند برابر شدن و تقسیم شدن استفاده می‌شود.
			تقسیم		
			مساوی		جهت نمایش عملگر مساوی از دو نماد مشابه همدیگر استفاده می‌شود.
		عملگرهای منطقی	AND		عملگر منطقی «و» سبب می‌شود دو عبارت پشت سر هم قرار بگیرد، به این معنا که برای عبور از این مسیر، بایستی هر دو عبارت درست باشند.
			OR		عملگر منطقی «یا» مابین دو عبارت قرار می‌گیرد و به این معنا است که اگر یکی از این دو مسیر هم درست بوده و اجازه ورود دهد، کفایت می‌کند.
			نقیض (!)		برای نمایش عملگر نقیض، کل مقادیر مجاز، برعکس می‌شوند.
		عملگرهای مقایسه‌ای	< > = => == =!		این دسته عملگرها با نماد خط‌کش نشان داده می‌شوند و محدوده مجاز با رنگ سبز و محدوده غیرمجاز با رنگ قرمز نشان داده شده است.
۳	عبارات شرطی		If-else		برای نمایش یک عبارت شرطی از مسیر دوراهی استفاده می‌شود.
۴	حلقه‌ها		While For		با کمک یک میدان در مسیر حرکت نشان داده می‌شود. حرکت در میدان و انجام تغییرات تا حدی ادامه می‌یابد که به شخصیت‌ها اجازه خروج از میدان داده شود.
۵	فراخوانی روش‌ها و پارامترهای ورودی				هر پارامتر ورودی با کمک یک شخصیت باربابا‌پایی نشان داده می‌شود. روش به مانند یک جعبه مبهم و جادویی است که بعد از خروج از این جعبه، تغییراتی در مقادیر شخصیت‌ها ممکن است رخ داده شود.

این تغییرات را تا زمانی انجام می‌دهد که تمام اجراها سبز رنگ شـــوند. به عبارتـــی دیگر تمام مجوزهـــا برای عبور شخصیت‌های باربابا‌پا درست می‌شود. در این صورت در برنامه نیز تمام موارد آزمون به درستی انجام می‌شوند و خطاها برطرف می‌شوند.

به عنوان مثال، کد شـــکل (۳) را در نظـــر بگیرید، ابتدا



شکل ۲: الف) درخت انتزاعی عبارت x+y>8. ب) بازی ساخته شده معادل عبارت



شکل ۳: یک مثال و بازی معادل با آن

درخت نحوی انتزاعی مربوط به کد تهیه می‌شود. سپس بر اساس این درخت، بازی مربوطه

برای این کد سه مورد آزمون وجود دارد، که در شکل(٤) نشان داده شده است. موارد آزمون، شامل مقادیر زیر است و پیشگوی آزمون آن عبارت نیز مشخص شده است. در مورد اول و دوم بایستی نتیجه نادرست است که در بازی هم نتیجه آن است که به بازیکنان اجازه ورود داده نمی‌شود. اما در مورد سوم، نتیجه بایستی درست شود؛ در حالی که در بازی نتیجه آن است که به بازیکنان اجازه ورود داده نمی‌شود. بنابراین بازیکن بایستی تغییراتی در بازی دهد که هم مورد سوم اصلاح شود و هم دو مورد قبل درست باقی بمانند. به عبارتی دیگر قاب دور هر سه تا مورد سبز شود.

۴-۲-۴. الگوها جهت ایجاد تعمیر

جهت ترمیم نرم‌افزار، الگوهایی بایستی تعریف شوند که هم قابلیت ترمیم مناسب نرم‌افزار را داشته باشند و هم از سوی دیگر قابلیت تبدیل به بازی را داشته باشند. مسئله مهم دیگر در تعیین این ترمیم‌ها این است که بازی بسیار پیچیده و سخت نشود. به عبارتی دیگر بازی برای بازیکنان جذاب باشد. بنابراین یک بخش چالش‌انگیز دیگر در طراحی بازی، تعیین الگوها جهت ترمیم نرم‌افزار است. در ادامه به سه دسته اصلی در الگوی ترمیم که در این مقاله بیان شده است، پرداخته می‌شود.

۱. تغییر در قسمتی از کد

۲. حذف قسمتی از کد

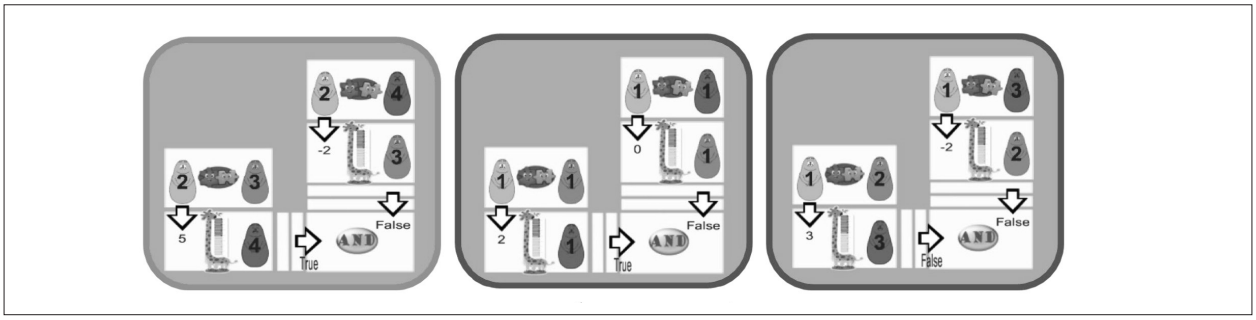
۳. اضافه کردن دستور به بخشی از کد

ایجاد تغییر در کد

در جدول (۲) به بررسی تغییراتی که در بازی امکان انجام آن است پرداخته می‌شود.

برای تبدیل این تغییرات به فرم بازی، بر روی هر شکلک از یک نماد تغییر استفاده می‌شود که با کلیک بر آن و انتخاب گزینه تغییر، تغییرات موردنظر اعمال می‌شود (شکل ٥). جهت انجام بازی توسط بازیکنان می‌توان از دو راهکار استفاده کرد.

- در راهکار اول به صورت دقیق چگونگی تغییر در کد توسط بازیکن تعیین می‌شود. به عنوان مثال بازیکن اعلام می‌کند قصد تغییر عملگر a را دارد، حال او باید دقیقا تعیین کند متغیر a را با چه متغیری بایستی عوض کرد. چالش اساسی در این راهکار، سختی انجام بازی توسط بازیکنان است. در مثال گفته شده، بازیکن باید با تمام متغیرهای حوزه بازی آشنا باشد و تمام آنها را امتحان کند تا متغیر درست انتخاب شود. بازیکنان با انتخاب‌های فراوانی روبه‌رو است که منجر به سختی بازی و بی‌انگیزگی او می‌شود.

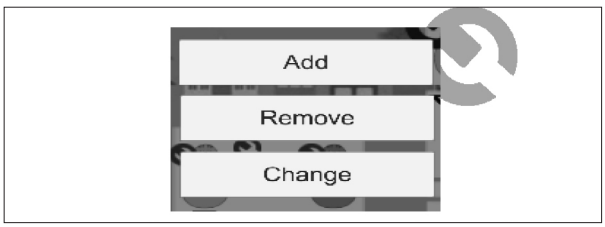


شکل ۴: نحوه نمایش موارد آزمون در بازی

جدول ۲: تغییرات قابل انجام در بازی

عملیات	توضیحات		عملگرهای یک حوزه
جایگزینی متغیرها	با انتخاب یک متغیر، آن متغیر با یکی از متغیرهای حوزه کد جایگزین می‌گردد.		
تغییر عملگرها	در این حالت عملگر با دیگر عملگرهای متناسب با آن عملگر جایگزین آن می‌شود.	عملگر محاسباتی	(+، -، *، / و %)
			(++ و --)
		عملگر ترکیبی	(+=، -=، *=، /=، %=)
		عملگر منطقی	(&& و ||)
		عملگر مقایسه‌ای	(<، >، <=، >=، ==، !=)
تغییر لیترال‌ها	بازیکن قادر است یک لیترال را انتخاب کند و مقدار آنرا تغییر دهد.		
تغییر در ساختار تصمیم‌گیری (عبارت شرطی و حلقه)	نقیض شرط	در این حالت بازیکن قادر است شرط را نقض کند	
	تغییر دستور اصلی در ساختار تصمیم‌گیری	دستور مورد نظر می‌تواند با دستورات مشابه جایگزین شود. در این صورت سیستم تغییرات لازم در ظاهر بازی و کد را به صورت خودکار انجام می‌دهد.	(for، while، do while)
تغییر در فراخوانی روش‌ها	تغییر در نام روش	در این صورت نام روش با نام روشهای موجود در حوزه کد می‌تواند تغییر می‌یابد.	
	تغییر در پارامترهای ورودی	در این حالت بازیکن قادر است مقادیر یا متغیرهای پارامترهای ورودی را تغییر دهد.	

- راهکار دوم که راهکار انتخابی در این مقاله است، مبتنی بر تعامل بیشـــتر بازیکنان و سیستم است. انجام تمام تغییرات بدین گونه است که بازیکن تغییر مورد نظر را انتخاب می‌کند، آنگاه سیستم متغیر یا عملگر مورد نظر را با تمام گزینه‌های ممکـــن جایگزین می‌کند. به عنوان مثال بازیکن تعیین می‌کند که متغیر a بایستی تغییر کند، آنگاه سیســـتم متغیر a را با تمام متغیرهای موجود در حوزه کد، جایگزین می‌کند. سپس سیستم تمام کدهای جایگزین را بررسی می‌کند تا ببیند کدام کد تمام موارد آزمون را به درســـتی اجرا می‌کنـــد. در این راهکار اگر



شکل ۵: عنصر بازی جهت ایجاد تعمیر در کد

هیچ جایگزینی موفقیت آمیز نبود، به بازیکن اختیار داده می‌شـــود که اگر قصد دارد، متغیر یا عملگر مورد نظر را به دلخواه بـــا یکی از گزینه‌های موجود تغییر دهد تا محیـــط را برای تغییرات دیگر آماده کند. به عنوان مثال بازیکـــن قصد دارد دو تغییر همزمان دهد در تغییر اول

یـک متغیر را عـوض کند و در تغییـر دوم یک عملگر محاسباتی را عوض کند، سـپس از سیستم می‌خواهد درستی این تغییرات را بررسی کند.

اضافه کردن به عبارت شرطی

– اضافه کردن یک قید ساده: در این حالت یک قید شبیه به قید مشابه و موجود در عبارت شرطی، اضافه می‌گردد. به عنـوان مثال به عبارت شـرطی (a>10) if، می‌توان قید a<15 را اضافه کرد: if(a>10 && a<15).

– چالشـی که در این حالت ممکن است پیش بیاید این اسـت که عملگر منطقی اضافه شـده همراه بـا قید جدید، چگونه تعیین شـوند. راهکار پیشنهادی ما این است که هر دو عملگر منطقی " و" و "یا" توسط سیستم وارسی شوند. در این حالت سیسـتم یک بار عبارت if(a>10 && a<15) را بررسی می‌کند و بار دیگر عبارت if(a>10 || a<15) را بررسی می‌کند.

– اضافـه کردن یک قید پیچیـده: در این حالت بازیکن قادر است یک قید با چندین متغیر به عبارت شرطی اضافه کنـد. متغیرهایی که اضافه می‌شـوند، از متغیرهای همان حوزه کد انتخاب می‌شـوند. به عنوان مثال عبارت شرطی x>10 می‌تواند به عبارت x>10 && y+z<15 تغییر یابد.

– اضافـه کردن یک پیش‌شـرط: در این حالت به طور معمول بازیکن درخواسـت اضافه شـدن یک پیش‌شرط قبـل از یک عبارت را دارد. در این حالت بازیکن بایسـتی دو نوع عملیات را انجـام دهد. ابتدا مکانی که قصد اضافه کردن پیش‌شـرط را دارد انتخاب می‌کند و سیستم عبارت شرطی if() اضافه می‌کند. سپس بازیکن به عبارت شرطی قید اضافه می‌کند.

– اضافـه کردن یـک فراخوانـی روش: در این حالت بازیکن قادر اسـت یکی از روش‌های حوزه کد را انتخاب و به کد اضافه کند. در این حالت سیسـتم ابتدا مقادیری را جهت پارامترهای ورودی روش تعیین می‌کند، سـپس اگر بازیکن مایل بود قادر اسـت که مقدار پارامترهای ورودی را تغییر دهد.

حذف کردن از یک عبارت

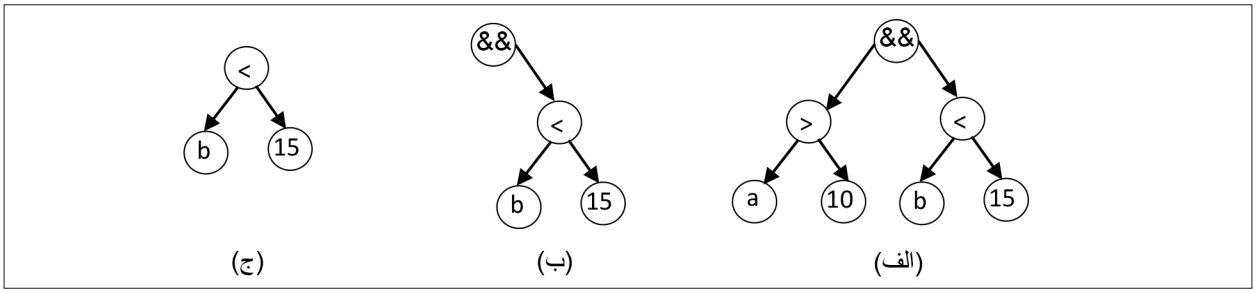
حذف کردن یک قید: در ایـن حالت می‌توان یک قید از یک عبارت شـرطی را انتخاب کرد و آن را حذف نمود. با حذف این قید ممکن است در عبارت شرطی عملگری اضافه باقـی بماند. بنابراین بعد از حذف قید در عبارت شـرطی، سیستم مجدد عبارت را بررسی می‌کند و عملگرهای اضافه را حـذف می‌کند. به عنوان مثال قصـد داریم قید a>10 را در عبـارت شـرطی if(a>10 && b<15) حـذف کنیم، در این صورت بازیکن در بـازی عنصر معادل با عملگر "<" را انتخـاب و آن را حذف می‌کنـد. در این صورت کل قید مربوطه حذف می‌شـود، سپس سیسـتم با بررسی مجدد عبارت شرطی، عملگرهای منطقی && را حذف می‌کند.

ممکن اسـت این سـوال پیش بیاید که سیستم چگونه عبارت شـرطی را پس از حـذف یک قید ترمیـم می‌کند. سیسـتم برای ترمیم مجدد عبارت شرطی از درخت نحو انتزاعی عبارت شرطی استفاده می‌کند و ابتدا فرزندان گره انتخابی حذف می‌شـود و سپس شروع به حذف گره های پدر به سـمت ریشـه می‌کند و هر زمان که توازن درخت برقرار شد، از حذف گره‌ها خودداری می‌کند. در مثال بالا درخت نحو انتزاعی به صورت شکل (٦–الف) است. در این درخت ابتدا گره < و سپس فرزندانش حذف می‌شود (شکل ٦–ب). سپس گره پدر آن، که گره && است و توازن در آن برقرار نیست، حذف می‌شود (شکل ٦–ج).

۱. حذف کـردن عبـارت شـرطی: در این حالـت کاربر درخواست حذف کل عبارت شرطی را دارد که به راحتی کل عبارت حذف می‌گردد.
۲. حـذف کردن یـک فراخوانی روش: در ایـن حالت نیز کاربر درخواست حذف فراخوانی روش را دارد بنابراین فراخوانی روش در عبارت حذف می‌شود.

۵-۲-۴. مثال کاربردی

در ادامه مثال کاربردی بزرگترین مقسوم‌علیه مشترک بین دو عدد را بررسـی می‌کنیم. این مثال و شـش نسخه خطادار آن در شـکل (۷) نشـان داده شده است. این مثال



شکل ۶: یک مثال از حذف یک قید در عبارت شرطی

یکـــی از مثال‌های پرچالش برای روش‌های ترمیم خودکار اســـت که در مقالات مختلف به آن اشـــاره شـــده است. به عنوان مثال در [۲۸] الگوریتم‌های ۱ تا ٦ از شکل‌های (۸ تا ۱۳) را در چند ابزار معروف بررسی کرده و بیان کرده که هر ابزار در ترمیم کدام الگوریتم‌ها دچار مشکل شده است و مشـــخص گشته اســـت که هیچ ابزاری توانایی حل تمام الگوریتم‌هـــا را ندارد. در ادامه ایـــن الگوریتم‌ها و داده‌های آزمون که منجر به اجرای درست و نادرست هر الگوریتم می‌شـــود و بازی طراحی شـــده مربوط بـــه آن را نمایش می‌دهیم و نشـــان می‌دهیم که تمام این الگوریتم‌ها توسط بازیکنان قابل حل هستند.

الگوریتم شـــماره یک کد اصلی و بدون خطا را نشـــان می‌دهد.

```
1: if(a==0){
2:          printf("%d",b);
3:          exit(0);}
4: while(b !=  0){
5:          if(a>b)
6:               a = a – b;
7:          else
8:               b = b-a;}
9: printf("%d",a);
10:exit(0);}
```

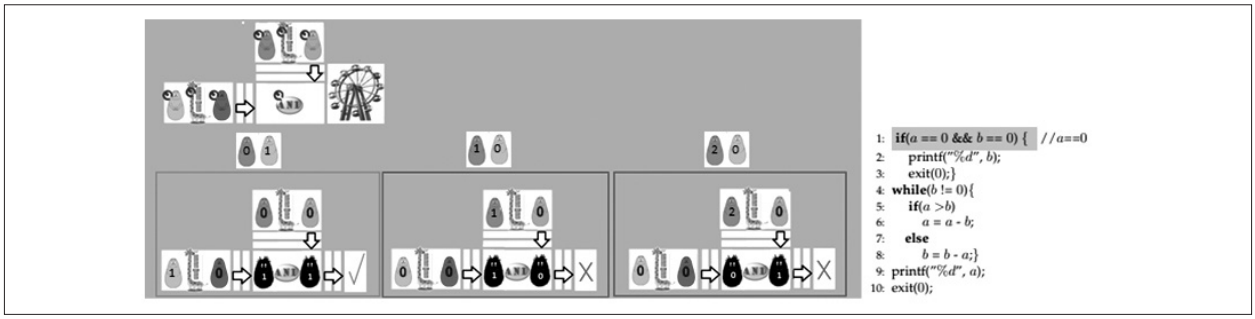
شکل ۷: الگوریتم اصلی بزرگترین مقسوم‌علیه مشترک بین دو عدد

در الگوریتم شماره ۲ همان‌گونه که مشخص شده خطا در عبارت (a==0 && b==0) اســـت. بازی طراحی شده به صورت شـــکل زیر است. سه نمونه آزمون مشخص شده، نشان می‌دهد که داده‌های (۰، ۲) و (۰، ۱) خروجی متناقض با کد اصلی دارد. در صورتی که داده (۱، ۰) خروجی متناسب با کد اصلی دارد. این داده‌ها نشان می‌دهند مقدار مخالف با صفر برای b تاثیر مخربی در نتیجه دارد. بنابراین بازیکن تصمیم به تغییر و یا حذف عبارت b==0 می‌گیرد. که البته حذف این دستور پاسخ به معما مربوطه است.

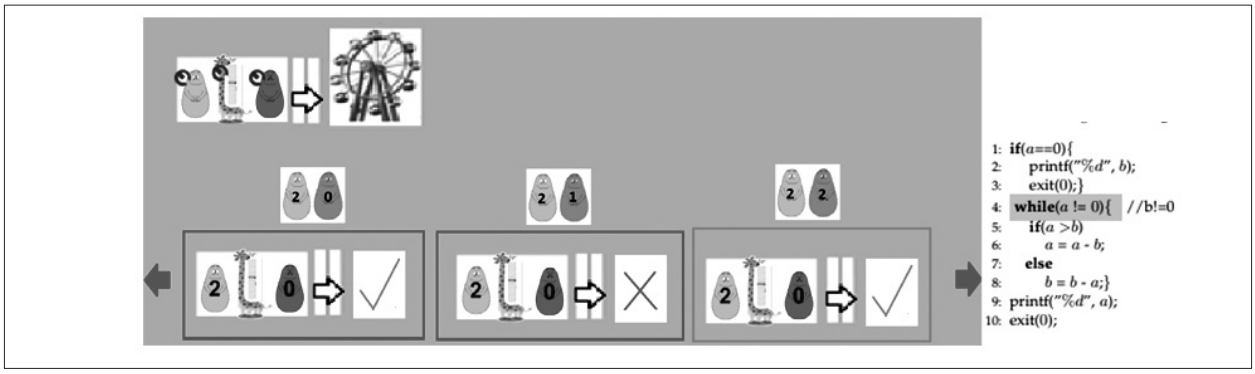
در الگوریتم شماره ۳، خطا در خط ٤ (عبارت a!= 0) قرار دارد. مـــوارد آزمونی که عبارت while(a!=0) را پوشـــش می‌دهند، بر اســـاس دستور شـــرطی خط شماره ۱، مقدار a در آن مخالف با صفر است. مجموعه داده‌های مشخص شده در بازی نشان می‌دهد مقادیر تعیین شده برای متغیر a فقط در صورتی به پاسخ صحیح دست می‌یابد که مقدار آن برابر با b اســـت. بنابراین ساده‌ترین و اولین تصمیمی که یک بازیکن می‌تواند بگیرد، تغییر متغیر a (شـــخصیت باربا‌پاپایی متناظر با آن) با متغیر b است.

یک چالش که در این مرحله با آن مواجه شـــدیم، تعداد موارد آزمون اســـت که بیشـــتر از ۳ مورد است. برای حل ایـــن چالش از علامت‌های فلش در ســـمت چپ و راســـت موارد آزمون جهت جابه‌جا کردن موارد آزمون اســـتفاده شده است.

در الگوریتم شماره ٤، خطا در حذف به اشتباه عبارت exit(0) است. این نوع خطا بیشترین چالش را برای بازیکنان دارد. زیرا، بازیکنان علاقـــه دارند ابتدا عناصری را که در بازی مشاهده می‌کنند، تغییر دهند و اگر به نتیجه نرسیدند آنگاه جهت اضافه کردن عنصر جدید اقدام می‌کنند. در این مرحلـــه نیز بازیکنان بعد از تغییـــر روش print و پارامتر ورودی آن، به این نتیجه می‌رســـد که نیاز به اضافه کردن عنصـــر جدید دارد. مزیتی که بازی در این مرحله دارد این اســـت که وقتی بازی در بین تعداد فراوانی بازیکن توزیع



شکل ۸: الگوریتم شماره ۲ و بازی معادل با آن



شکل ۹: الگوریتم شماره ۳ و بازی معادل با آن

می‌شـــود، هر بازیکن نحوه رفتار وانجام بازی متفاوتی از دیگـــری دارد، یعنی به تعداد بازیکنـــان راه‌حل برای بازی اســـت. به این ترتیب مســـئله‌ای به مانند ایـــن الگوریتم که بسیار چالش دارد می‌تواند توسط بازیکنان به تعداد فراوان به زودی حل شود.

در الگوریتم شماره پنج، نیاز به دو تغییر در بازی است. بازیکن با مشـــاهده داده‌ها به راحتی متوجه می‌شود فقط در صورتـــی اجرای برنامه صحیح اســـت که مقدار a از b کوچکتر نیست. بنابراین تصمیم به تغییر عملگر < می‌گیرد. اگر بازیکن عملگر بزرگتر را انتخاب کند آنگاه همان اجرای آخر دیگر به درستی اجرا نمی‌شود. اما با انتخاب عملگر = فقط یک مورد آزمون (۰ و ۱) درســـت اجرا نمی‌شـــود. اما با ایجاد تغییر دوم و تغییر متغیر b با لیترال ۰ مســـئله حل می‌شود.

در الگوریتم شـــماره شـــش، خطا در فراخوانی روش print تشـــخیص داده شده اســـت. در این سطح از بازی، بازیکنـــان به راحتی ورودی روش را می‌تواند تغییر دهد و به نتیجه مطلوب می‌رسد.

۵. ارزیابی

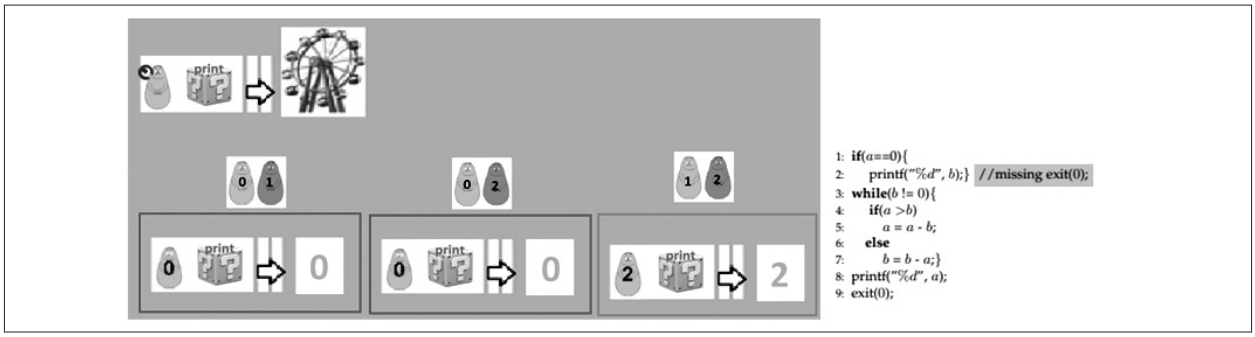
جهت پیاده‌ســـازی بازی و انجام آزمایش‌ها، از موتور بازی‌ســـازی یونیتـــی[۱۵] اســـتفاده شـــده اســـت. از محیط سی‌شارپ[۱۶] نیز جهت مقایسه و بررسی معیارهای ارزیابی در داده‌های حاصل از انجام بازی استفاده گشته است.

در ایـــن مقاله، بـــازی از دو جنبه مـــورد ارزیابی قرار می‌گیرد. از جنبه اول، توانمندی بازی برای حل مسئله فنی اصلی که همان ترمیم نرم‌افزار اســـت، ارزیابی می‌شود. از جنبه دوم معیارهای یک بازی خوب بررسی می‌گردد.

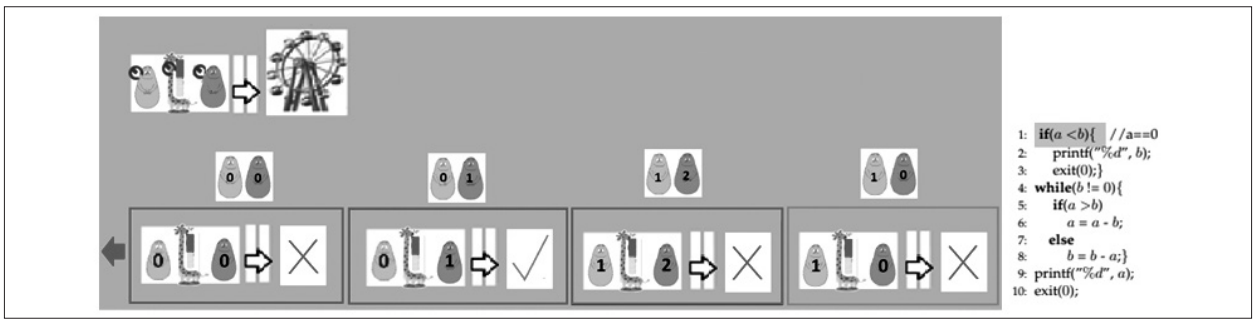
برای بررســـی نتایج مربوط به بخش ترمیم نرم‌افزار، سه معیار میانگین زمان ترمیم کد، تعداد وصله‌های صحیح تولید شـــده و تعداد وصله‌های بیش برازش شـــده، مورد استفاده قرار گرفتند.

از منظر میزان سودمندی[۱۷] بازی (چون اگر بازی خوب نباشـــد، بازیکنی به آن بازی متمایل نمی‌شود و در نتیجه مســـائل فنی پشت پنهان در بازی نیز حل نمی‌شود) از سه معیار به نام‌هـــای توان عملیاتی[۱۸]، میانگین زمان بازی[۱۹] و

15- Unity
16- C#
17- Usefulness
18- Throughput
19- Average lifetime play (ALP)



شکل ۱۰: الگوریتم شماره ۴ و بازی معادل با آن



شکل ۱۱: الگوریتم شماره ۵ و بازی معادل با آن

جدول ۳: برنامه‌های محک

نام برنامه	تعداد خط‌های کد	تعداد نسخه‌های خطادار از برنامه
Replace	۵۶۴	۱۸
Schedule	۴۱۲	۳
Schedule2	۳۷۴	۱
Tcas	۱۷۳	۱۲
Totinfo	۵۶۵	۱۰
Pronttokens2	۵۷۰	۶
Space	۹۵۶۴	۳

همکاری مورد انتظار [20] استفاده می‌شود [۱۷]. این معیارها، معیارهایی استاندارد هستند که میزان جذابیت بازی نیز با کمک آنها سنجیده می‌شود. در این پژوهش علاوه بر این معیارها، پرسش‌نامه‌هایی نیز تهیه و در اختیار بازیکن‌ها قرار می‌گیرد تا میزان علاقه خود را به بازی بیان کنند.

لازم به ذکر است که برای ارزیابی روش پیشنهادی، از مجموعه داده زیمنس (جدول ۳) که در تحقیقات مشابه، به عنوان برنامه محک به کار گرفته شده، در این مقاله استفاده شده است.

۵-۱. نحوه انجام آزمایش

– برای برنامه‌هایی که در جدول (۱) به عنوان معیار محک معرفی شده‌اند، یک نسخه از بازی با ۵۳ مرحله طراحی شده است.

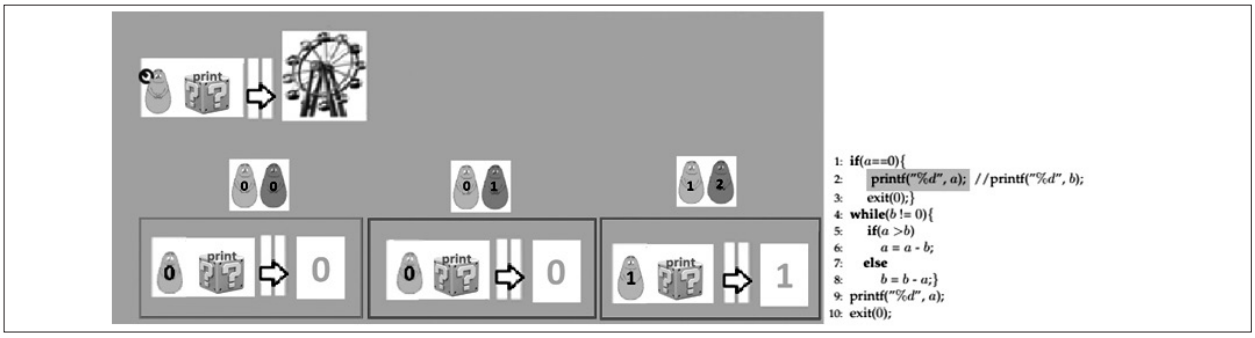
– در این آزمایش، ۲۵ بازیکن در محدوده سنی ۲۰ تا ۲۳، و پنج برنامه‌نویس با میانگین سه سال تجربه برنامه‌نویسی، در آزمایشگاه بازی دانشگاه شهید بهشتی حضور به هم رساندند.

– در آغاز این جلسه، درباره کار تحقیقاتی که انجام می‌شود، بدون اشاره به آزمون نرم‌افزار، به بازیکنان توضیحاتی داده شد و از آن‌ها خواسته شد در صورت رضایت، در این آزمایش‌ها شرکت کنند. سپس مکانیک‌ها و قوانین بازی شرح داده شد و یک نمونه آزمایشی از بازی در اختیار آن‌ها قرار گرفت، تا آن‌ها با محیط بازی آشنا شوند و اگر مشکل یا سوالی داشتند، پاسخ داده شود.

– سپس بازی‌ها به بازیکنان داده شد و از آن‌ها خواسته شد معماهای بازی را حل کنند.

– در نهایت، زمان حل مسئله اندازه‌گیری شد. پرسشنامه‌ای به بازیکنان داده شد و چند سوال درباره جذابیت بازی پرسیده شد، تا مشخص گردد که بازی طراحی‌شده، چقدر برای بازیکنان جذاب است. یک سوال که

20- Expected contribution



شکل ۱۲: الگوریتم شماره ۶ و بازی معادل با آن

پرسیده شد این است که بازی در ارتباط با چه موضوعی اســت؟ هدف از این ســوال این بود که مشخص گردد آیا مسئله فنی در قالب عناصر بازی پنهان شده است یا خیر.

– از ســوی دیگر، برنامه‌ها به تعدادی برنامه‌نویس نیز داده شــد و از آنها خواسته شد، کد مورد نظر را با توجه به اوراکل داده شده، ترمیم کنند. سپس زمان حل مسئله جهت مقایسه با زمان حل مسئله توسط بازیکنان، اندازه‌گیری شد.

– پاســخ‌های گروه اول به پرسشــنامه مورد ارزیابی قرار گرفت تا میزان جذابیت بازی برای بازیکنان مشخص گردد، و از طرف دیگر تعیین شــود که آیا به خوبی مفهوم پیش‌زمینــه بازی پنهان شــده اســت؛ یا به عبــارت دیگر اطمینان حاصل شــود افرادی کــه دانش خاصی در زمینه برنامه‌نویسی ندارند، می‌توانند این بازی را انجام دهند.

– نتایج به دســت آمده از بازی، با نتایج برنامه‌نویسان و ابزارهای خودکار ترمیم نرم‌افزار، از نظر میزان درستی و سرعت، مقایسه شدند.

۵-۲. نتایج

تمامی بازیکنان، همه مراحل بازی را به اتمام رساندند. ســپس نتایج، جمع‌آوری گردید و از دو دیدگاه که به آن‌ها اشاره شد، مورد بررسی قرار گرفتند.

۵-۲-۱. ترمیم نرم‌افزار

پنجاه و سه مرحله از بازی طراحی شد و این مراحل به صورت تصادفی در بین بازیکنان توزیع شد تا هر مرحله شانس مساوی در حل شدن توسط بازیکنان داشته باشند. در ۳۰ دقیقه بازیکنان، ۱۳۲۷ تلاش جهت ترمیم کدها انجام

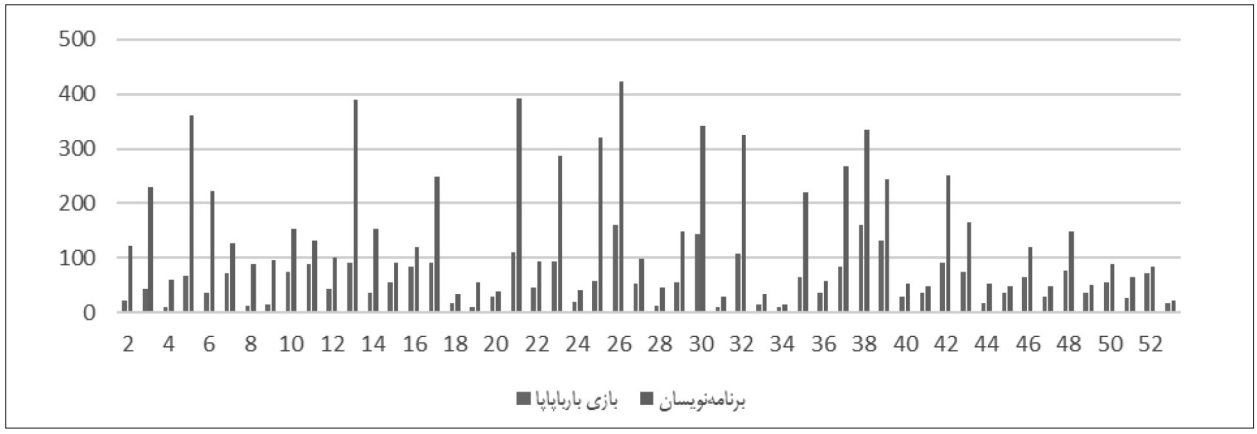
جدول ۴: نتایج حاصل از بازی

برنامه‌ها	تعداد وصله‌های صحیح	تعداد وصله‌های بیش برازش شده
Replace	۱۶	۲
Schedule	۳	۰
Schedule2	۱	۰
Tcas	۱۲	۰
Totinfo	۸	۲
Pronttokens2	۶	۰
Space	۳	۰

دادند، که ٤٩ کد از کدهای موردنظر توسط بازیکنان ترمیم شد و برای ٤ تا کد باقیمانده وصله بیش برازش، تولید شد. یعنی به طور متوســط برای هر کد ۲۵ وصله توسط تمام بازیکنان تولید شد. بازی قادر بود تمام وصله‌های صحیح برای کدهــای pronttokens، Tcas، schedule و Space را تولید کنند. اما برنامه‌نویسان در ۳۰ دقیقه حدود ۱۱٦ وصله پیدا کردند که به طور متوسط برای هر کد، ۲,۱ وصله پیدا کردند. مزیتی که تنوع وصله‌های تولید شده دارد، این است که در ترمیم نرم‌افزار، بعضی از انواع وصله‌ها ممکن است منجر به وصله بیش برازش شده شوند. در چنین مواقعی، در اختیــار داشــتن وصله‌های متنوع جهــت یافتن وصله صحیح راه‌گشا است.

جدول (٤) تعداد وصله‌های صحیح و بیش برازش شده توسط روش پیشنهادی را به ترتیب در ستون‌های دوم و سوم گزارش می کند.

مدت زمان سپری شده در بازی برای تولید اولین وصله نامزد برای یک کد، در حالت میانگین برابر با ٥٦,٤ ثانیه است. همان‌گونه که در شــکل (۱۳) نشان داده شده، بهترین زمان



شکل ۱۳: زمان ایجاد ترمیم توسط بازی و برنامه‌نویسان

برای تولید وصله نامزد ۹ ثانیه است؛ و برای ٤٧ تا از کدها قبل از ۱۰۰ ثانیه وصله نامزد تولید شد. بدترین زمان برای تولید وصله نامزد یک کد ١٦١ ثانیه است (دلیل این که مدت زمان بیشتری برای این کد صرف شد وجود عبارات شرطی پیچیده در آن کد است). برنامه‌نویسان در مدت زمانی طولانی‌تر (به صورت میانگین ۱٤۹٫۸ ثانیه) موفق به پیدا کردن وصله برای کدها شدند. همان‌گونه که در شکل (۱۳) نشان داده شده است، این بازی از لحاظ زمان، بسیار سریع‌تر از روش‌های سنتی تولید وصله توسط انسان، می‌تواند وصله تولید کند.

با هدف این که نشان دهیم تفاوت بین میانگین زمان برای بازی و برنامه‌نویسان معنادار است، یک آزمون آماری را انجام داده‌ایم. اما قبل از تصمیم‌گیری جهت استفاده از آزمون آماری مناسب، بایستی نرمال بودن داده‌ها بررسی شوند. برای هر دو مجموعه داده‌ها از زمان‌ها، از آزمون اندرسون–دارلینگ جهت بررسی فرضیه صفر «مجموعه داده‌های مدت زمان به طور نرمال توزیع شده‌اند» استفاده شد. مقادیر p محاسبه‌شده برای همه مقادیر مجموعه داده‌ها بالاتر از سطح انتخابی معنی‌دار (۰٫۰۵) هستند؛ بنابراین، می‌توان گفت که تمام مجموعه داده‌های مدت زمان از جمعیت‌هایی که به طور نرمال توزیع شده بودند، استخراج شدند. پس از بررسی نرمال بودن داده‌ها، از آزمون t برای رد فرضیه صفر «برنامه‌نویسان در مدت زمان کمتری نسبت بازیکنان ترمیم کدها را انجام می‌دهند» استفاده می‌کنیم. از آن جایی که مقادیر p محاسبه شده کمتر از سطح معنی دار انتخابی (۰٫۰۵) است، فرضیه صفر رد و فرضیه جایگزین «بازیکنان در مدت زمان کمتری نسبت برنامه‌نویسان ترمیم کدها را انجام می‌دهند» پذیرفته می شود.

۵-۲-۲. بازی هدفمند

در این بخش، معیارهایی مربوط به سنجش یک بازی بر اساس GWAP برای روش پیشنهادی محاسبه می‌شود تا تعیین گردد بازی مطرح شده از دیدگاه GWAP چقدر می‌تواند مفید واقع شود.

میانگین تعداد نمونه های مسئله حل شده در هر ساعت انسانی به عنوان توان عملیاتی در نظر گرفته می‌شود. با توجه به مقادیر ذکر شده در جدول (٤)، در هر جلسه (به طور متوسط)، بازیکنان ۵۳ معما را در ۰٫۸۱ ساعت حل کردند. بنابراین، توان عملیاتی بازی برابر است با:

نمونه مسئله در هر ساعت انسانی ٦٥,٤٣ = ۵۳/۰٫۸۱

کل مدت زمانی که یک بازیکن به طور متوسط مشغول بازی است، به عنوان ALP معرفی می‌شود. بازیکنان به صورت میانگین ۰٫۸۱ ساعت را در هر جلسه به بازی کردن اختصاص دادند. بنابراین،ALP برابر است با:

ساعت ۰٫۰۳۲ = ۲۵ / ۰٫۸۱

در نهایت، مشارکت مورد انتظار برابر است با:

توان عملیاتی × ALP = ٦٥,٤٣ * ۰٫۰۳ = ۲٫۱۲

۵-۲-۳. بررسی پرسشنامه

بر ای بررسی برخی از ویژگی‌های بازی از دیدگاه

بازیکنـــان، پرسشـــنامه‌ای در اختیار آنها قرار داده شـــد. پاســـخ‌هایی که بازیکنان به ســـوالات پرسشـــنامه دادند، جمع‌آوری شـــد. در ادامه ســـوالات پرسشنامه و پاسخ‌ها بیان می‌شوند:

۱. در ســـوال اول از بازیکنان خواســـته شد به میزان جذابیت بازی، عددی بین ۱ تا ۱۰ دهند. میانگین نمره‌ای که بازیکنان به این سوال دادند برابر با ۷ بود. این پاسخ نشان می‌دهد که بازی به اندازه کافی، بخش‌های ســـرگرم‌کننده دارد تا بتواند بازیکنان جدیدی را جذب کند.

۲. از ۲۵ داوطلب شرکت‌کننده ۲۰ نفر آنها اعلام کردند که علاقه‌مند به نصب بازی هســـتند. پاسخ بازیکنان به این سوال نشـــان می‌دهد که بازی توانایی نگهداری بازیکنانی قدیمی خود را دارد.

۳. در ســـوال ســـوم از بازیکنان خواسته شد که هدف اصلی از انجام این بازی را تشـــخیص دهنـــد. چهار گزینه در اختیـــار آنها قرار گرفت که یکی را انتخاب کنند. گزینه‌ها شامل «تحقیقات در زمینه روانشناسی»، «ترمیم نرم‌افزار»، «تحقیقات در زمینه آموزش کودکان» و «شبیه‌سازی سیستم الکترونیـــک»، بود. ده نفر از بازیکنـــان «تحقیقات در زمینه روانشناسی» را انتخاب کردند. پنج نفر «شبیه سازی سیستم الکترونیک»، هشت نفر «تحقیقات در زمینه آموزش کودکان» و دو نفـــر دیگر نیز «ترمیم نرم‌افـــزار» را انتخاب کردند. از پاسخ‌های بازیکنان می‌توان دریافت که مسئله فنی به خوبی در قالب عناصر بازی پنهان شـــده اســـت. این پنهان‌سازی مسئله فنی کمک می‌کند که بازیکنان بدون نیاز به تخصص خاصی در زمینه آزمون نرم‌افزار و یا حتی برنامه‌نویسی و علم کامپیوتر، بتوانند به خوبی در انجام بازی شرکت کنند.

۵-۳. مقایسه بازی پیشنهادی با ابزار ترمیم خودکار نرم‌افزار

در این بخش تصمیم گرفته شد که نتایج به‌دست آمده از بازی با یک ابزار معـــروف در ترمیم خودکار نرم‌افزار مقایســـه گـــردد. از آنجـــا که بـــازی بر اســـاس مجموعه کدهای زیمنس ارزیابی شده است، بنابراین ابزار معروف GenProg کـــه نتایج مبتنی بر کدهای زیمنس توســـط آن ارائه شـــده است، جهت مقایسه انتخاب شد [۸]. نرخ ترمیم گزارش شـــده از این ابزار در جدول (۵) نشـــان داده شده اســـت. همان‌گونه که از نرخ ترمیم محاسبه شده در جدول (۵) قابل مشاهده اســـت می‌توان بیان کرد که بازی دارای برتری مطلق نسبت به GenProg در ترمیم کدها دارد.

۶. بحث در مورد نتایج

از ارزیابی این بازی چند نتیجه حاصل شد که در ادامه به صورت خلاصه، ارائه می‌شوند.

در آزمایش‌ها مشـــخص شـــد بازیکنان ترمیم کدها را ســـریع‌تر از برنامه‌نویســـان انجام دادند و همچنین ترمیم ۹۲ درصدی از کدها توســـط بازی، میسر شد و در همان بازه زمانی ۳۰ دقیقه‌ای، برنامه‌نویسان موفق به اصلاح ۸۳ درصدی کدها شده‌اند. همچنین مشخص شد که این بازی به اندازه کافی جذاب اســـت که بتواند بازیکنان جدید را به ســـوی خود بکشـــاند و بازیکنان قدیمی خود را حفظ کند. عـــلاوه بر این نتایج، چند مزیت زیر نیـــز برای بازی، قابل تأمل هستند:

- علاوه بر این که بازی از لحـــاظ کارایی عملکرد بهتری دارد، بازیکنان تعداد وصله بیشتری نسبت به برنامه‌نویسان تولید کردند.
- بازیکنان به صورت مجانی در بازی شـــرکت کردند. در نتیجه بدون پرداخت هزینه‌ای، ترمیم کد انجام شد. در صورتی که به برنامه‌نویسان، به عنوان نیروی کار، بایستی دستمزد پرداخت شود. همچنین طراحی بازی به‌گونه‌ای صورت گرفته است که به ازای هر کد واحد جدید، معما مربوطه در بازی به صورت خودکار ایجاد می‌شـــود؛ یعنی هزینه‌ای برای ایجاد معماهای جدید صرف نمی‌شود.
- در صورتـــی که طراحی بازی به خوبـــی صورت گرفته باشد (نتایج نشان از طراحی خوب این بازی دارد)، به مرور زمـــان تعداد بازیکنـــان افزایش می‌یابـــد، در حالی که تعداد برنامه‌نویسانی که یک شرکت می‌تواند استخدام کند همواره

جدول ۵: نرخ ترمیم گزارش شده از ابزارهای GenProg، بازی باراپاپا

نام برنامه	GenProg	بازی
Schedule	۴٪	۱۰۰٪
Schedule2	۱۲٪	۱۰۰٪
Tcas	۲۴٪	۱۰۰٪
Totinfo	۶٪	۸۰٪
Printtokens2	۲۴٪	۱۰۰٪

محدود است (به دلیل پرداخت دستمزد و مسائل مالی).

● از آنجا که ترمیم نرم‌افزار یک وظیفه فنی و کسل‌کننده است، برنامه‌نویسان در طولانی مدت، از فعالیت در این حوزه، خسته می‌شوند و علاقه‌مند هستند که در بخش‌های دیگر از فرآیند تولید نرم‌افزار، فعالیت کنند. در آزمایش‌های انجام شده در این تحقیق، مدت زمانی که برنامه‌نویسان درگیر ترمیم نرم‌افزار بودند، کوتاه بود و خستگی بر آنها چیره نگشت؛ با این‌حال ترمیم نرم‌افزار توسط آنها زمان‌گیر بود. اما از سوی دیگر بازی جنبه تفریح برای بازیکنان دارد و با ورود بازیکنان به حالت سیال ذهنی، ساعت‌ها بازی کردن، آنها را خسته نمی‌کند.

● مسئله دیگر این است که بازیکنان در هر زمان از شبانه‌روز به صورت مجانی، بازی را انجام می‌دهند. اما برنامه‌نویسان فقط در ساعات کاری، ترمیم نرم‌افزار را انجام می‌دهند.

در ادامه به مزیت‌های انسان و بازی در حل مسائل جدی که دلیل اصلی موفقیت این بازی است می‌پردازیم. بایستی در اینجا اشاره کرد که بازیکنان در طول بازی غیرتصادفی و هوشمندانه رفتار می‌کنند. هوش انسانی و توانایی یادگیری در انسان منجر به عملکرد موثر در بازی می‌شود. تعداد زیاد از GWAPهای موفق، شواهد محکمی دال بر این قضیه است و نشان می‌دهد مغز بازیکنان به طور تصادفی عمل نمی‌کند. اما یک مثال معروف در زمینه بازی‌های جدی، بازی فولدیت است [۲۹] که در اینجا به آن اشاره می‌کنیم. این بازی توسط دانشگاه واشنگتن ارائه شد و در آن تا کردن پروتئین‌ها به صورت یک معمای برخط طراحی شده است. این بازی قرار است به چالش دانشمندان در تا خوردگی پروتئین جهت یافتن راه‌حلی برای بیماری‌هایی مانند ایدز، آلزایمر و سرطان، کمک کند. در بازی برخط فولدیت، ۵۷۰۰۰ کاربر ثبت نام کردند و بازی را انجام دادند که در کمتر از ۱۰ روز راه‌حلی برای مشکل ارائه شده در درمان ایدز معرفی کردند. دلیل اصلی موفقیت این نوع از بازی‌ها توانایی بازیکنان در حل هوشمندانه معماهای بازی است. علاوه بر آنچه در گفته شد، بازی مطرح شده در این مقاله به عنوان یک GWAP، از هوشمندی بازیکنان به صورت زیر استفاده می‌کند:

۱- با وجود این که بازیکنان رمزهای عبور را از طریق آزمون و خطا پیدا می‌کنند، اما هوشمندانه رفتار می‌کنند. زیرا، از آنجایی که افراد متفاوت از یکدیگر می‌آموزند و فکر می‌کنند، رفتار بازیکنان از همدیگر متفاوت است. به عنوان مثال، یک بازیکن سعی می کند مقدار تغییر در مقادیر شخصیت‌ها دهد، در حالی که بازیکن دیگر سعی بر اضافه کرد یک روش جدید به برنامه دارد. علاوه بر این، برخی از بازیکنان ممکن است کمتر به الگوهای کشف شده قبلی متعهد باشند و دائماً در تلاش برای کشف الگوهای جدید باشند.

مزیت ویژگی های مختلف بازیکنان این است که به تعداد بازیکنان الگوریتم‌های هوشمند برای حل معما وجود دارد. به این ترتیب انواع معماها به سرعت توسط جمعیت بازیکنان حل می‌شود. اما روش‌های تصادفی فاقد این ویژگی هستند. علاوه بر این، از آنجایی که اکثر روش‌های هوشمند در ترمیم نرم‌افزارها از الگوریتم‌های خاصی پیروی می‌کنند، به عنوان مثال، الگوریتم‌های بازپخت شبیه‌سازی شده و الگوریتم‌های ژنتیک، این روش‌ها نیز به طور معمول قادر به حل سریع و دقیق تعداد قابل توجهی از معماهای مختلف نیستند.

۲- بازیکنان برتر در یک بازی، دارایی‌های ارزشمندی برای یک GWAP هستند. ممکن است فقط یک بازیکن برتر در بین ۱۰۰ بازیکن وجود داشته باشد. با این حال، آن بازیکن برتر به احتمال فراوان یک معمای پیچیده را به روشی بسیار کارآمد حل می‌کند، طوری که پاسخ او به معماها برای هدف ما بسیار ارزشمند است. به عنوان مثال، در آزمایش‌ها، بازیکنی وجود داشت که ۴۵ ترمیم را در

کمتر از ۲۰ دقیقه انجام داد. او به راحتی توانست رابطه بین عناصر بازی را پیدا کند و به سرعت معماها را حل کنند.

۶-۱. بازیکنـــان چگونه و کجا از هوش خود اســـتفاده می‌کنند

در ابتـــدای انجام بازی و در ســـطوح اولیـــه، بازیکنان کورکورانه تغییرات را جستجو می‌کنند. اما در طول انجام بازی، بازیکنان یاد می‌گیرند که چگونه رمزهای عبور را با هـــوش خود پیدا کنند. در ادامه به فرصت‌هایی که منجر به به تحلیل و یادگیری بازی توسط بازیکنان می‌شود، اشاره می‌کنیم، که منجر به بهبود بازی می‌شود:

۱- معمولاً یک ســـری دســـتورات در اکثر کدها تکرار می‌شوند و خطاهای مشابهی در آنها رخ می‌دهد. همچنین، عناصر بـــازی مربوطه در ســـطوح مختلف بـــازی تکرار می‌شـــوند. برای مثال، چندیـــن تا از کدهـــای محک ارائه شـــده در این مقاله با عبارت شرط if (input == 0) شروع می‌شوند که یک خطای معمول در برنامه‌ها حذف به اشتباه کد پیش‌شرط اســـت. به طور معمول، بازیکنان با چند بار بازی کردن، الگوهای حل معماها را کشف می‌کنند و نحوه اســـتفاده از آنها را در سایر سطوح بازی یاد می‌گیرند. به طور معمول، روش‌های ترمیم خـــودکار نرم‌افزار قادر به کشف این الگوها نیستند.

۲- رونـــد تغییر در بارپاپاها و ســـرزمین آنها، یک دید بصری برای بازیکنان ایجاد می‌کند. بازیکنان با این دیدگاه بصری بهتر می‌توانند از هوش خود اســـتفاده کنند. به این ترتیب بازیکنـــان می‌توانند تاثیرات تغییـــرات را در نتایج مشاهده کنند و به راحتی بتوانند آنها را تحلیل کنند.

۳- بازیکنان معمولاً به دنبال رابطه علت و معلولی بین تغییراتی که ایجاد می‌کنند و جریان بازی هستند. به همین دلیل، انسان در یک روالی که فقط یکسری تغییرات تکراری را انجام دهد، گیر نمی‌کند.

۷. نتیجه‌گیری و کارهای آتی

جهت خودکارســـازی ترمیم نرم‌افزار، تحقیقات وسیعی صـــورت گرفته اســـت، اما به دلیل مشـــکلاتی که هنوز این روش‌ها دارند، در صنعت انسان‌ها ترمیم کدها تولید می‌کنند. ترمیم نرم‌افزار توسط انســـان هزینه بالایی دارد. بنابراین در این مقاله راه‌کاری در زمینه بهبود مشـــکلات روش‌های ترمیم نرم‌افزار مبتنی بر انســـان، بیان شـــد. به دلیل این که بازی‌ها همواره سرگرم‌کننده و جذاب هستند، در این راهکار از مفهوم بازی‌های هدفمند استفاده شده است.

نتایج حاصل از ارزیابی نشـــان می‌دهد، بازی باربا‌پاپا به اندازه کافی جذاب اســـت که بازیکنان را به انجام بازی تشـــویق کند. در نتیجه، این بازی می‌تواند ترمیم نرم‌افزار را بـــه یک فرآیندی جذاب تبدیل کنـــد. جذابیت بازی منجر می‌شود که تعداد فراوانی بازیکن، بدون دریافت هزینه‌ای، در انجام این بازی مشارکت کنند. بنابراین برای حل مسئله ترمیم نرم‌افزار، نیروهای ارزان فراوانی در دسترس هستند، و می‌تواننـــد در مدت زمان کوتاهی وصله‌های نرم‌افزار را تولید کنند. بازی به گونه‌ای طراحی شده است که برای تمام افراد قابل درک باشـــد و نتایج، این ادعا را، که این نیروها نیـــاز به تخصص خاصی ندارند، اثبات کرد. همچنین نتایج نشـــان داد که بازی توانست برای ۹۲ درصد کدها، وصله صحیح را تولید کند.

بازی طراحی شـــده، اولین بازی برای ترمیم نرم‌افزار است، بنابراین واضح است که همچنان تلاش‌های فراوانی نیاز اســـت تا چالش‌های موجود رفع گردند. علاوه بر رفع چالش‌هایی که به آنها اشـــاره شد، چند دسته فعالیت دیگر نیز در نظر گرفته شده اســـت که در کارهای آتی بر روی آنها تمرکز خواهد شد، از جمله:

۱. این بازی برای ترمیم نرم‌افزار برای عبارت خطادار طراحی شـــده است، یعنی ابتدا توسط یک روش مکان‌یابی خطا، محل خطا یافت شود و سپس توسط بازی ترمیم آن صورت گیرد. هدف این است که بازی جهت مکان‌یابی خطا هم گسترش یابد.

۲. بـــه منظور ایجاد راهنمایی‌هـــای بهتر به بازیکنان و تولید داده برای مسیرهای پیچیده، می‌توان از راهکارهای تولید داده مبتنی بر جســـتجو استفاده کرد. با تقویت بازی با این راهکارها، محیط هوشـــمندتری طراحی می‌شود که

می‌توانـــد به بازیکنان در تولید داده، کمک شـــایان توجهی کند.

منابع

[1] G. Kirkpatrick, “Welcoming All Gods and Embracing All Places”: Computer Games As Constitutively Transcendent of the Local.”, Game History and the Local. Palgrave Macmillan, Cham, pp. 199-219, (2021).

[2] R. Abreu, P. Zoeteweij, R Golsteijn, and et al. “A practical evaluation of spectrum-based fault localization”, Journal of Systems and Software, 82(11),pp. 1780-1792, (2009).

[3] J.A. Jones, M.J. Harrold. “Empirical evaluation of the tarantula automatic faultlocalization technique”, In Proc. of Automated Software Engineering (ASE), Long Beach, CA, USA, pp. 273–282, (2005).

[4] X. Zhang, N. Gupta, R. Gupta. “Locating faulty code by multiple points slicing,” Softw. Pract. Exper., 37(9), pp. 935-961, (2007).

[5] J. H. Perkins, S. Kim, S. Larsen, S. Amarasinghe, J. Bachrach and M. Carbin, “Automatically patching errors in deployed software,” in ACM SIGOPS, (2009).

[6] A. Koyuncu, K. Liu, Tegawendé F. Bissyandé, Dongsun Kim, Jacques Klein, Martin Monperrus, and Yves Le Traon. “Fixminer: Mining relevant fix patterns for automated program repair.” Empirical Software Engineering 25, pp. 1980-2024, (2020).

[7] K. Liu, A. Koyuncu, D. Kim, and et al, “TBar: Revisiting template-based automated program repair.” In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 31-42. (2019).

[8] J. H. Perkins, S. Kim, S. Larsen, and et al “Automatically patching errors in deployed software,” in ACM SIGOPS, (2009).

[9] J. Sohn, S. Kang, and S. Yoo. “Arachne: Search-Based Repair of Deep Neural Networks.” ACM Transactions on Software Engineering and Methodology 32, no. 4, pp. 1-26, (2023).

[10] N. Jiang, L. Thibaud, and T. Lin. “Cure: Code-aware neural machine translation for automatic program repair.” In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 1161-1173. IEEE, 2021.

[11] F. Long and M. Rinard, “Staged program repair with condition synthesis,” in 10th Joint Meeting on Foundations of Software Engineering, (2015).

[12] B. Mehne, H. Yoshida, M.R Prasad, and et al. “Accelerating search-based program repair.” In 2018 IEEE 11th international conference on software testing, verification and validation (ICST), pp. 227-238. IEEE, 2018.

[13] C. L. Goues, M. Dewey-Vogt, S. Forrest and et al. “A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each,” in 34th International Conference on Software Engineering (ICSE), (2012).

[14] T. Durieux, and M. Monperrus. “Dynamoth: dynamic code synthesis for automatic program repair.” In Proceedings of the 11th International Workshop on Automation of Software Test, pp. 85-91. (2016).

[15] R. Gupta, S. Pal, A. Kanade and et al. “DeepFix: Fixing Common C Language Errors by Deep Learning,” in Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)1345, (2017).

[16] L. Von Ahn. “Games with a purpose”. In: Computer 39.6, pp. 92-94, (2006).

[17] L. Von Ahn, and L. Dabbish. “Designing games with a purpose”. In: Communications of the ACM 51.8, pp. 58-67, (2008).

[18] E. Oh Navarro and A. van der Hoek. “Simse: An interactive simulation game for software engineering education”. In: CATE., pp. 12-17, (2004).

[19] N. Tillmann, J. D. Halleux, and T. Xie. “Pex4fun: Teaching and learning computer science via social gaming”. In: 2012 IEEE 25th Conference on Software Engineering Education and Training., pp. 90-91, (2012).

[20] N. Tillmann, J. Bishop, N. Horspool, and et al. “Code hunt: Searching for secret code for fun”. In: 7th International Workshop on SearchBased Software Testing, pp. 23-26, (2014).

[21] H. Logas, J. Whitehead, M. Mateas, and et al. “Software verification games: Designing Xylem, The Code of Plants”. In: FDG. (2014).

[22] W. Dietl, S. Dietzel, M.D. Ernst, and et al. “Verification games: Making verification fun”. In: 14th Workshop on Formal Techniques for Java-like Programs, pp. 42-49, (2012).

[23] D. Fava, D. Shapiro, J. Osborn, and et al. “Crowdsourcing program preconditions via a classification game”. In: 38th International Conference on Software Engineering, pp. 1086-1096, (2016).

[24] J.M. Rojas and G. Fraser. “Code defenders: A mutation testing game”. In: 11th International Workshop on Mutation Analysis. IEEE. (2015).

[25] N. Chen, and K. Sunghun. “Puzzle-based automatic testing: Bringing humans into the loop by solving puzzles.” 2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. IEEE, (2012).

[26] S. AmiriChimeh, H. Haghighi, M. Vahidi-Asl et al. “Rings: A Game with a Purpose for Test Data Generation”. In: Interacting with Computers, pp. 1-30, (2017).

[27] Sh. Moosavi, H. Haghighi, H. Sahabi and et al. “Greenify: A Game with the Purpose of Test Data Generation for Unit Testing.” International Conference on Fundamentals of Software Engineering. Springer, Cham, (2019).

[28] G. Luca, D. Micucci, and L. Mariani. “Automatic software repair: A survey.” In Proceedings of the 40th International Conference on Software Engineering, pp. 1219-1219. (2018).

[29] F. Khatib, F. DiMaio, Foldit Contenders Group and et al. “Crystal structure of a monomeric retroviral protease solved by protein folding game players”. In: Nature structural molecular biology 18.10, pp. 1175-1177, (2011).

[30] Sh. Moosavi, M. Vahidi-Asl, and H. Haghighi. “Test Data Generation for Program Units Using a Game with a Purpose.” Scientia Iranica (2023).