تحلیل خودکار بازی رایانه‌ای با استفاده از شبکه پتری رنگی

سعید پاشازاده، استادیار

دانشکده مهندسی برق و کامپیوتر- دانشگاه تبریز- تبریز- ایران– pashazadeh@tabrizu.ac.ir

چکیده: وجود اشکال در طراحی از عوامل ایجاد نقص در سیستم است. تشخیص و رفع ایرادها در مرحله طراحی مانع از افزایش هزینه و زمان تولید سیستم می‌گردد. برای اثبات ویژگی‌های رفتاری سامانه‌ها از روش‌های رسمی استفاده می‌شود. برای اطمینان از عدم وجود بن‌بست، تضمین وجود جواب و تعیین میزان پیچیدگی معما، از روش‌های رسمی در طراحی معمای بازی‌های رایانه‌ای می‌توان استفاده کرد. شبکه پتری رنگی سلسله مراتبی[1] یک روش رسمی مدل‌سازی است که می‌تواند برای ارزیابی معماهای بازی‌های رایانه‌ای مانند جورچین‌های طراحی مسیر استفاده شود. بازی Unblock Me به‌عنوان یک مثال موردی در این مقاله مدل‌سازی شده و تحلیل خودکار دو معمای آن موردبررسی قرار گرفته است. مدل‌سازی بازی به‌صورت سلسله‌مراتبی انجام شده است. در این مقاله راهکارهای جدیدی برای حل مشکل انفجار حالت و کاهش زمان اجرای مدل ارائه گردیده است. مدل‌سازی باهدف امکان تحلیل خودکار فضای حالت سیستم انجام شده و توابع موردنیاز جهت اثبات ویژگی‌های رفتاری پیاده‌سازی شده است. این مقاله روشی برای مدل‌سازی و اثبات ویژگی‌های رفتاری بازی‌های رایانه‌ای از نوع جورچین را با استفاده از شبکه پتری رنگی سلسله مراتبی ارائه کرده و قابل‌تعمیم به بازی‌های مشابه است.

واژه‌های کلیدی: مدل‌سازی، ویژگی رفتاری، شبکه پتری رنگی، تحلیل فضای حالت، بازی رایانه‌ای.

Automatic Analysis of Computer Game Using Colored Petri Net

S. Pashazadeh, Assistant professor

Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran, Email: pashazadeh@tabrizu.ac.ir

Abstract: Existence of design faults is a source of failure in the system. Detection and removal of faults in design phase prohibits increase of cost and time of system development. Formal methods are used for proving behavioral properties of systems. Formal methods can be used in crux design of computer games to ensure absence of deadlock, guarantee of answer's existence and to determine complexity of crux. Hierarchical colored Petri net is a formal method that can be used to evaluate cruxes of computer games such as path planning puzzle games. Unblock me game is modeled as a case study and automatic analysis of its two cruxes are studied in this paper. Modeling of game is done hierarchically. New approaches for eliminating state space explosion problem and decreasing running time of model is presented in this paper. Modeling is done with aim of automatic state space analysis of the system and required functions for proving behavioral properties are implemented. This paper is presented a method for modeling and proving behavioral properties of puzzle type computer games using hierarchical colored Petri net that and can be generalized to similar games.

Keywords: Modeling, behavioral feature, colored Petri net, state-space analysis, computer game.

تاریخ ارسال مقاله: ۱۳۹۳/۱۱/۱
تاریخ اصلاح مقاله:۹۴/۳/۶ و ۱۳۹۴/۴/۲۸
تاریخ پذیرش مقاله: ۹۴/۷/۴
نام نویسنده مسئول: سعید پاشازاده
نشانی نویسنده مسئول: ایران - تبریز - بلوار ۲۹ بهمن - دانشگاه تبریز - دانشکده مهندسی برق و کامپیوتر

۱- مقدمه

روش‌های رسمی[۲] به علت داشتن پایه ریاضی برای اثبات ویژگی‌های رفتاری سامانه‌ها مورداستفاده قرار می‌گیرند. سامانه‌های نرم‌افزاری کاربردهای وسیعی دارند و اطمینان از صحت عملکرد آن‌ها و اثبات ویژگی‌های خاصی مانند عدم وجود بن‌بست[۳]، امن بودن، زنده بودن و عادل بودن باعث افزایش اتکاپذیری این سامانه‌ها می‌شود. بازی‌های رایانه‌ای ازجمله برنامه‌های کاربردی نرم‌افزاری هستند که می‌توان از روش‌های رسمی در طراحی آن‌ها استفاده کرد. بدین‌وسیله در مرحله طراحی می‌توان ثابت کرد که آیا سیستم مشکل رفتاری موردبررسی را خواهد داشت یا نه؟ هم‌چنین اطلاعات کامل، در رابطه با سناریویی که منجر به رخداد وضعیت ناخواسته در سیستم می‌شود، قابل‌استخراج است.

شبکه پتری رنگی سلسله‌مراتبی توسعه‌یافته شبکه پتری سنتی است که برای مدل‌سازی سامانه‌های همروند با ارتباطات سنکرون و آسنکرون طراحی شده است. این روش مدل‌سازی رسمی مبتنی بر تئوری کیسه[۴] و دارای واسط گرافیکی ساده است [۱]. اجرای مدل با ظاهری گرافیکی و ساده قابل‌مشاهده بوده و مبتنی بر قواعد جبری شبکه پتری رنگی است. شبکه پتری رنگی از زبان هوش مصنوعی ML که یک زبان رسمی مبتنی بر حساب لانداء[۵] است بهره می‌برد [۲]. زبان ML برای مدل‌سازی و همچنین تحلیل فضای حالت سیستم استفاده می‌شود. افزودن این زبان به شبکه پتری ضمن حفظ رسمی بودن آن، قدرت مدل‌سازی این نوع شبکه را به میزان چشم‌گیری افزایش داده است. مدل‌سازی سلسله‌مراتبی، امکان تجرید مدل در سطوح معنایی مختلف را مهیا کرده و باعث ساده‌تر شدن فرایند مدل‌سازی سیستم می‌شود. CPN tool ابزاری مناسب برای مدل‌سازی و تحلیل خودکار فضای حالت سیستم توسط شبکه پتری رنگی سلسله‌مراتبی است [۳].

شبکه پتری رنگی غیرزمان‌دار برای مدل‌سازی و اثبات صحت سامانه‌های مبتنی بر فضای حالت گسسته به کار می‌رود. اغلب بازی‌های رایانه‌ای مانند جورچین‌های از نوع پیدا کردن مسیر یا طراحی مسیر[۶] نرم‌افزارهایی هستند که دارای فضای حالت گسسته متناهی می‌باشند. در این مقاله طراحی معما (مسئله) برای چنین بازی‌هایی توسط شبکه پتری رنگی سلسله‌مراتبی موردبررسی قرار گرفته است. بازی معروف Unblock Me که در گوشی‌های تلفن همراه و رایانه‌های شخصی استفاده می‌شود به‌عنوان نمونه موردبررسی قرار گرفته است. اهداف مدل‌سازی این بازی به شرح زیر هستند:

۱) آیا ممکن است معما دارای بن‌بست باشد؟ این مشکل در برخی از بازی‌ها مشاهده می‌شود.
۲) آیا معمای طراحی‌شده قابل‌حل است؟ معمولاً معماها به‌طور تصادفی و خودکار تولید می‌شوند و ممکن است که برخی از آن‌ها دارای حل نباشند. نامطلوب است اگر کاربر با مسئله‌ای روبرو شود که حل ندارد.
۳) آیا میزان سختی معما قابل‌تعیین است؟ چه معیارهایی را می‌توان برای تعیین میزان سختی معما مطرح کرد که کمی باشند؟
۴) آیا می‌توان کاربر را در راستای حل معما راهنمایی کرد؟
۵) آیا می‌توان مانند بازی شطرنج به حرکت‌های کاربر امتیاز مثبت و منفی داد؟

۲- بررسی کارهای گذشته

استفاده از روش‌های رسمی در طراحی بازی‌های رایانه‌ای حوزه بکری است که به‌ندرت به آن پرداخته شده است. ایده استفاده از شبکه پتری برای مدل‌سازی بازی‌ها و بررسی ایرادهای احتمالی آن‌ها اولین بار توسط مقاله‌ای در یک کنفرانس سال ۲۰۰۹ مطرح شد. در این مقاله یک بازی نمونه با استفاده از شبکه پتری سنتی مدل‌سازی شده است و نشان داده شده که شبکه پتری برای طراحی بازی رایانه‌ای قابل‌استفاده و ارزشمند است [۴]. البته توانایی مدل‌سازی شبکه پتری سنتی در مقایسه با شبکه پتری رنگی سلسله‌مراتبی بسیار پایین است و به علت بزرگ شدن سریع مدل در هنگام تحلیل فضای حالت؛ با مشکل انفجار حالت مواجه می‌شود. این عوامل از مهم‌ترین دلایل فراگیر نشدن استفاده از شبکه پتری در طراحی بازی‌ها است.

در آخرین کاری که در این زمینه انجام شده است بررسی انگیزش بازیکنان توسط شبکه عصبی LVQ[۷] به کمک شبکه پتری سنتی مدل‌سازی و بررسی شده است. در این مقاله، خوشه‌بندی کاربران بر اساس رفتار انجام می‌شود. هدف، تنظیم بازی متناسب با انگیزش بازیکنان است؛ به‌طوری‌که میزان علاقه‌مندی آن‌ها برای ادامه بازی افزایش داده شود [۵].

با اینکه استفاده از شبکه پتری در مدل‌سازی بازی‌های رایانه‌ای متداول نیست ولی در زمینه‌های دیگر مانند مدل‌سازی فرایند گردشکار در سامانه‌های تولیدی پرکاربرد بوده و مقاله‌های بسیاری در این زمینه به چاپ رسیده است [۶]. همچنین استفاده از شبکه‌های پتری رنگی در بررسی زمان‌بندی دستگاه‌های تولیدی و خط تولید نیز مرسوم است [۷].

مدل‌سازی مدیریت تخصیص منابع برای اجتناب از بن‌بست در سامانه‌های رایانه‌ای توسط شبکه پتری رنگی ازجمله کاربردهای دیگر روش‌های رسمی و شبکه پتری رنگی است [۸]. با بررسی فضای حالت الگوریتم Banker، امن بودن وضعیت جاری سیستم قابل‌بررسی است. همچنین می‌توان اثبات کرد که آیا تخصیص یک منبع خاص به یک پردازنده باعث ناامن شدن وضعیت سیستم می‌شود یا نه؟

کاربرد دیگر شبکه پتری رنگی، مدل‌سازی قوانین دسترسی به منابع در سامانه‌های فراگیر است. می‌توان تضاد قوانین را با تحلیل فضای حالت سیستم کشف کرد [۹]. در تحقیق دیگری، راهکار کنترل دسترسی پردازنده‌ها به اطلاعات یا منابع و عملکرد قوانین اعطاء یا سلب اختیارات توسط شبکه پتری رنگی مدل‌سازی شده است. بدین ترتیب نشان داده‌شده است که حفره‌های امنیتی سامانه‌ها و سازمان‌ها به‌روش رسمی قابل‌استخراج است [۱۰].

مدل‌سازی و مطالعه ویژگی‌های رفتاری سامانه‌های هم‌روند مختلف با استفاده از شبکه پتری رنگی از طریق بررسی فضای حالت در اکثر مواقع موفقیت‌آمیز بوده و نتایج جالبی به دنبال دارد. استفاده از این روش در طراحی بازی‌های رایانه‌ای و معماهای آن‌ها در این مقاله موردمطالعه قرار گرفته است.

۳- مفاهیم اولیه در مدل‌سازی بازی **Unblock Me**

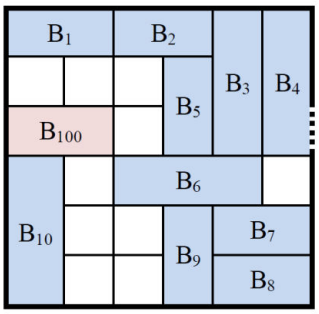
برای مدل‌سازی بازی فرض کنید که خانه‌های جدول ۶×۶ مربوطه و همچنین بلوک‌ها شماره‌گذاری شده‌اند. شکل ۱ خانه‌های فرضی جدول جورچین مربوط به بازی Unblock Me را نمایش می‌دهد.

۰	۱	۲	۳	۴	۵
۶	۷	۸	۹	۱۰	۱۱
۱۲	۱۳	۱۴	۱۵	۱۶	۱۷
۱۸	۱۹	۲۰	۲۱	۲۲	۲۳
۲۴	۲۵	۲۶	۲۷	۲۸	۲۹
۳۰	۳۱	۳۲	۳۳	۳۴	۳۵

شکل ۱: محیط بازی جورچین **Unblock Me** و موقعیت خانه‌های آن

خانه‌های این جدول به‌صورت سطری از چپ به راست با شروع از اندیس صفر شماره‌گذاری شده است. دیواره سمت راست خانه شماره ۱۷ به‌صورت نقطه‌چین ترسیم شده است. این دیواره محل خروج بلوک هدف[۸] است.

شکل ۲ پیکربندی اولیه معمای شماره ۳۲ سطح مقدماتی این بازی در حالت آسوده[۹] (بدون محدودیت زمانی) را نمایش می‌دهد. این معما به‌عنوان اولین مثال در مقاله موردبررسی قرار گرفته است. هدف، خارج ساختن بلوک شماره ۱۰۰ (بلوک هدف) از طریق حفره دیوار سمت راست خانه شماره ۱۷ نمایش داده‌شده در شکل ۱ است. همان‌طور که در شکل ۲ قابل‌مشاهده است بلوک‌های ۳ و ۴ و ۵ موانع خروج بلوک شماره ۱۰۰ هستند.



شکل ۲: وضعیت اولیه جورچین شماره ۳۲ بازی **Unblock Me**

تمام بلوک‌ها در این بازی به شکل مستطیلی بوده و فقط در راستای طولی قادر به حرکت می‌باشند. البته نسخه‌های مشابه این بازی بانام‌های دیگر نیز وجود دارند که در آن‌ها هر بلوک بیانگر یک خودرو است که فقط در راستای طولی به سمت جلو و عقب قادر به حرکت است. خارج کردن خودرو مشخص‌شده از پارکینگ، چالش مدنظر بازی است.

۴- مجموعه رنگ‌های تعریف‌شده برای مدل‌سازی سیستم

در این قسمت از مقاله مجموعه رنگ‌های تعریف‌شده برای مدل‌سازی سیستم ارائه شده است. این مجموعه رنگ‌ها به‌مثابه ساختمان‌داده نشانه‌ها[۱۰] بوده و به شرح زیر هستند:

```
colset PLACE = record loc : INT * blk : INT;
colset POSITION = list PLACE;
```

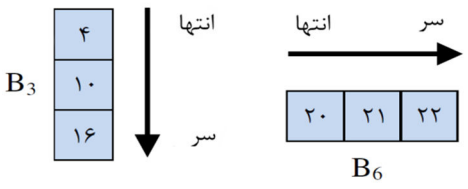
مجموعه رنگ PLACE رکوردی با دو فیلد است. فیلد اول بانام loc بیانگر یک خانه جدول جورچین، طبق شکل ۱ است و فیلد دوم بنام blk بیانگر این است که این خانه توسط چه بلوکی پر شده است. مجموعه رنگ POSITION فهرستی از مجموعه رنگ PLACE است. این مجموعه رنگ برای بیان وضعیت کلی جورچین استفاده خواهد شد. نحوه پرشدن خانه‌های جدول توسط بلوک‌های مختلف توسط یک لیست از این نوع بیان خواهد شد. ثابت InitialState از نوع رنگ PLACE، وضعیت اولیه جورچین نمونه موردبررسی در این مقاله را که در شکل ۲ نمایش داده شده است بیان می‌کند.

```
colset ORIENTATION= with Horizontal | Vertical |
                    Undefined;
```

مجموعه رنگ ORIENTATION از نوع شمارشی[۱۱] و برای بیان راستای قرار گرفتن یک بلوک تعریف شده است. مقدار Undefined برای مواقعی تعریف شده که بلوک به‌صورت مربعی بوده و راستای مشخصی نداشته باشد.

```
colset DIRECTION= with Forward | Backward;
colset POSLIST = list INT;
```

مجموعه رنگ DIRECTION از نوع شمارشی تعریف شده و بیانگر راستای حرکت یک بلوک است. شکل ۳ خانه‌های دو بلوک ۳ و ۶ از شکل ۲ را به‌عنوان نمونه نمایش می‌دهد. همان‌طور که در این شکل قابل‌مشاهده است؛ جهت افزایش شماره خانه‌های یک بلوک، به‌طور قراردادی بیانگر جهت حرکت روبه‌جلو در نظر گرفته شده است. جهت روبه‌جلو برای بلوکی که راستای آن عمودی باشد از بالا به پایین و برای بلوکی که راستای آن افقی باشد از چپ به راست در نظر گرفته شده است. خانه سروته یک بلوک طبق نمونه‌های شکل ۳ تعریف شده است. این اصطلاحات در بررسی نحوه حرکت بلوک‌ها در توابع تعریف شده برای این منظور استفاده خواهند شد.



شکل ۳: راستای قراردادی حرکت روبه‌جلو و موقعیت خانه سر و انتهای یک بلوک عمودی و افقی در مدل‌سازی سیستم

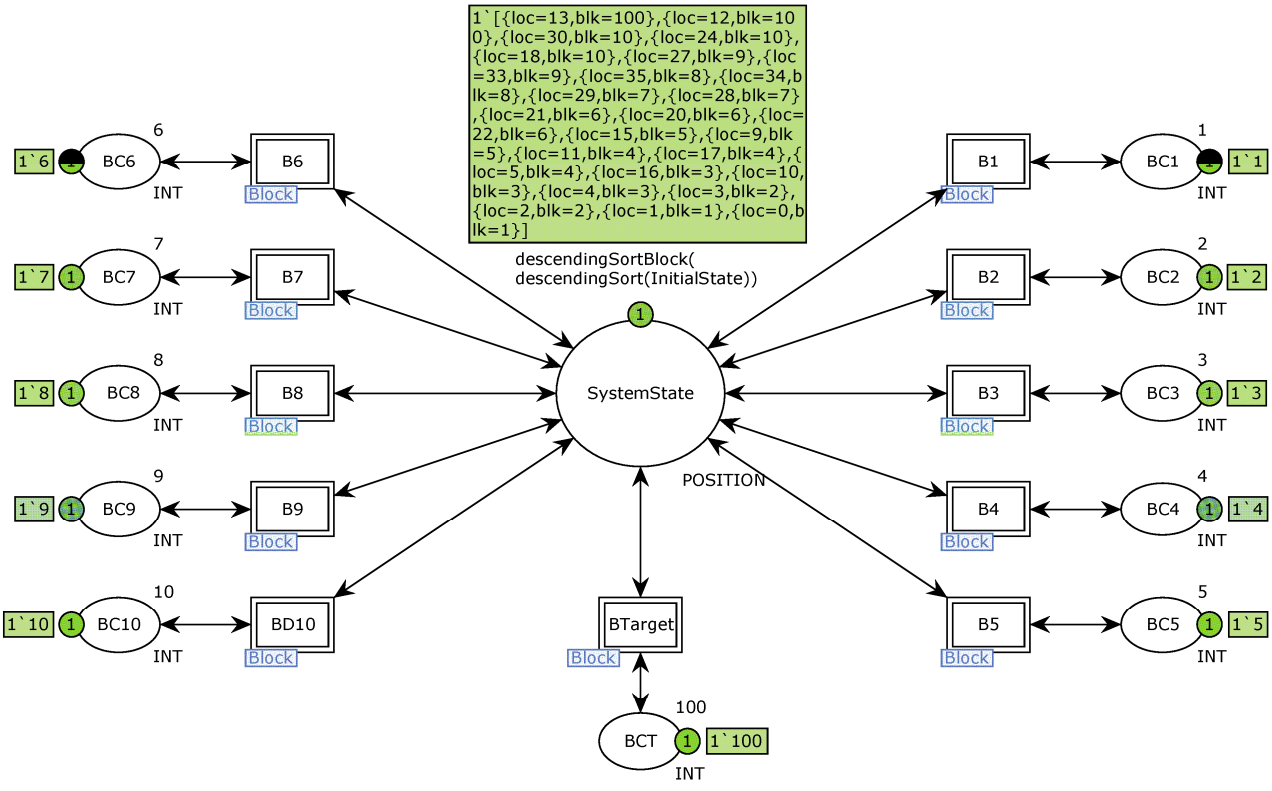
۵- مدل سیستم

در این قسمت از مقاله، مدل‌های طراحی‌شده ارائه می‌شوند. مدل‌سازی سیستم به‌صورت سلسله‌مراتبی انجام شده است که به‌طور چشم‌گیری پیچیدگی مدل را کاهش داده و اجازه بیان سیستم را در سطوح تجریدی مختلف مهیا می‌کند. شکل ۴ مدل سطح بالای بازی Unblock Me را نمایش می‌دهد.

مقدار ثابت InitialState در بخش ۲ مقاله توضیح داد شد. این مقدار ثابت، بیان‌گر وضعیت اولیه بلوک‌های مثال شکل ۲ است. اما سؤال مهم این است که چرا دو بار عمل مرتبط سازی نزولی روی وضعیت سیستم انجام شده است؟

در نرم‌افزار CPN tools ترتیب قرار گرفتن اعضای یک لیست مهم است. در مدل ارائه‌شده، ترتیب بیان اطلاعات بلوک‌ها در لیست برای ما مهم نیست ولی نرم‌افزار CPN tools به‌این‌ترتیب اهمیت می‌دهد. این امر باعث می‌شود که جایگشت‌های مختلفی که بیانگر یک اطلاعات هستند به‌عنوان وضعیت‌های مختلف سیستم در نظر گرفته شوند. این امر به‌طور نمایی تعداد وضعیت‌های سیستم را افزایش می‌دهد و باعث بروز مشکل انفجار حالت می‌گردد. اولین راهکار موفق پیشنهادشده در این مقاله برای اجتناب از مشکل انفجار فضای حالت، ذخیره وضعیت جدید سیستم در مکان SystemState و دو مرحله عمل مرتب‌سازی است. رکوردهای فهرستی که بیانگر وضعیت سیستم است، ابتدا به‌این‌ترتیب نزولی اندیس خانه‌های جدول و سپس به‌این‌ترتیب نزولی اندیس شماره بلوک‌های سیستم مرتب می‌شوند. توضیحات توابع مورداستفاده در مدل‌سازی سیستم در بخش ۶ ارائه شده است.

مکان SystemState وضعیت جاری جورچین را بیان می‌کند. وضعیت جورچین با مشخص کردن تک‌تک بلوک‌ها و موقعیت آن‌ها با توجه به خانه‌های جدول جورچین بیان می‌شود. نشانه‌گذاری اولیه[۱۲] این مکان توسط فراخوانی توابع به‌صورت descendingSortBlock(descendingSort(InitialState)) تعیین شده است.



شکل ۴: مدل سطح بالای بازی **Unblock Me**

در شکل ۴ مکان‌های BC1 تا BC10 هر یک بیانگر اندیس یک بلوک از مدل هستند. مکان BCT بیانگر اندیس بلوک هدف است که به‌طور قراردادی با اندیس شماره ۱۰۰ مشخص شده است. انتقال‌های B1 تا B9 و انتقال BD10 انتقال‌های جانشینی[۱۳] هستند که بیان‌گر مدل‌های حرکتی بلوک‌های ۱ تا ۱۰ می‌باشند. انتقال BTarget نیز انتقال جانشینی است که بیانگر حرکت‌های ممکن بلوک هدف است. همه این انتقال‌های جانشینی از نوع پیمانه Block هستند. مدل مربوط به این پیمانه که زیرپیمانه اصلی این سیستم است در شکل ۵ نمایش داده شده است.

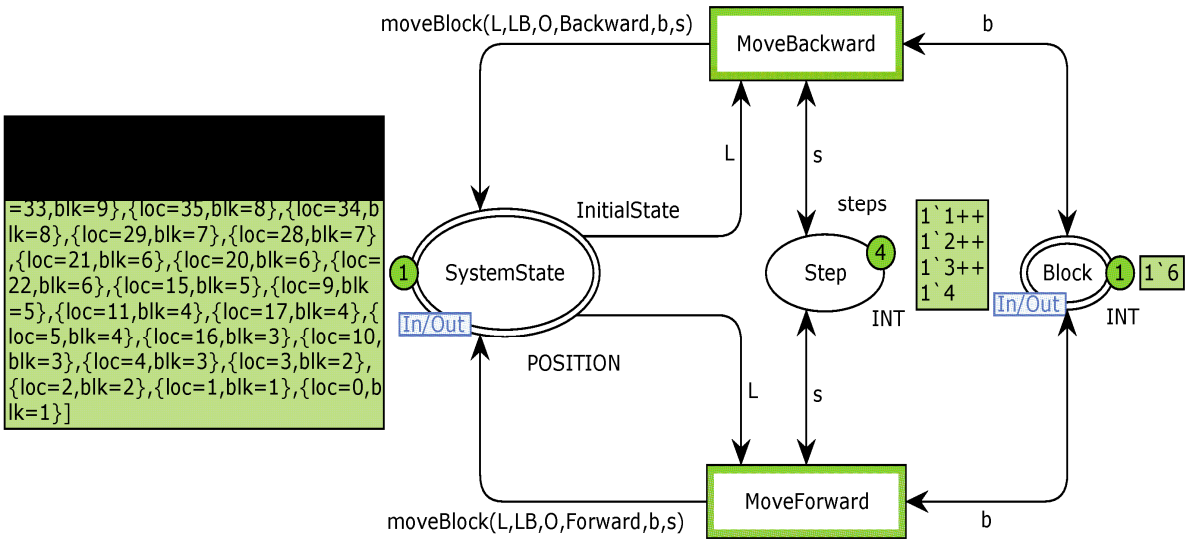
مدل پیمانه Block شکل ۵ مانند مدل سطح بالای شکل ۴ دارای مکانی بنام SystemState از نوع رنگ POSITION است. این مکان از نوع درگاه ورودی/خروجی است و با حفره[۱۴] SystemState در مدل

سطح بالای شکل ۴ مرتبط است. برای حرکت دادن یک بلوک لازم است که وضعیت سایر بلوک‌ها را برای تعیین امکان حرکت بلوک جاری داشته باشیم. مکان Block با نوع رنگ INT یک درگاه ورودی/خروجی است که اندیس بلوک را بیان می‌کند و با یکی از مکان‌های حفره BC1 تا BC10 یا مکان BCT مرتبط است. مکان Step یک مکان داخلی با نوع رنگ INT است. با توجه به اینکه هردو بعد جدول جورچین ۶ و حداقل طول بلوک‌ها ۲ است؛ بنابراین هر بلوک می‌تواند حرکتی به میزان ۱ تا ۴ قدم داشته باشد. به‌عنوان رنگ اولیه مکانStep، ثابت زیر تعریف شده است:

```
val steps = 1`1++1`2++1`3++1`4;
```

انتقال‌های MoveForward و MoveBackward مربوط به حرکت بلوک جاری به سمت جلو و عقب است. تشریح شرط نگهبان[۱۵] فعال شدن هر انتقال و همچنین برنوشته‌های[۱۶] هر کمان نیازمند معرفی توابع تعریف‌شده در مدل است. توضیحات این توابع در بخش بعد ارائه شده است. تعریف متغیرهایی که در نوشتن شرط نگهبان انتقال‌ها و برنوشته‌های کمان‌ها استفاده شده‌اند به شرح زیر می‌باشند:

```
var b: INT;     var L, LB:POSITION;
var s:INT;      var O:ORIENTATION;
```



شکل ۵: مدل پیمانه **Block**

۶- تابع‌های به‌کاررفته در مدل‌سازی سیستم

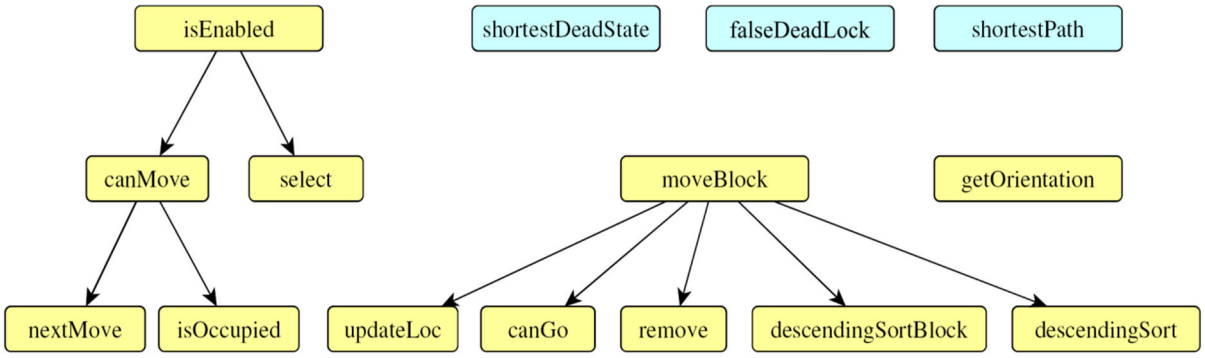
دومین راهکار موفق پیشنهادی این مقاله برای حل مشکل انفجار فضای حالت، کدنویسی به زبان ML است. این رویکرد باعث کاهش حجم مدل، افزایش سرعت اجرا و کاهش حافظه مصرفی از طریق کاهش مکان‌های به‌کاررفته در مدل می‌شود. به‌عنوان‌مثال، پردازشی مانند مرتب‌سازی نیاز به مدل پیچیده‌ای با چندین مکان و انتقال دارد. چنین مدلی فضای حالت بزرگی تولید می‌نماید. با استفاده از تعریف توابع به زبان ML می‌توان چنین پردازشی را بدون ایجاد فضای حالت اضافی در زمان کوتاه‌تری انجام داد.

شکل ۶ نمودار ساختاری[۱۷] توابع به‌کاررفته در مدل را نمایش می‌دهد. در این قسمت از مقاله، تمام توابع به‌کاررفته در مدل به‌طور مختصر توضیح داده‌شده‌اند و کد برخی از توابع سطح بالا ارائه شده است.

نمایه تابع select در زبان ML به شکل زیر است:

```
fun select(L : POSITION, index : INT): POSITION
```

پارامتر اول این تابع لیست اطلاعات تمام بلوک‌ها و پارامتر دوم آن شماره یک بلوک خاص است. تابع اطلاعات بلوک مربوطه را جستجو کرده و اطلاعات مکانی آن را به‌عنوان یک لیست جدید برمی‌گرداند.



شکل ۶: نمودار ساختاری تابع‌های به‌کاررفته در مدل‌سازی

نمایه تابع descendingSort به شکل زیر است:

```
fun descendingSort(L : POSITION) : POSITION
```

این تابع فهرستی از اطلاعات موقعیتی بلوک‌ها را دریافت کرده و این اطلاعات را به‌صورت نزولی مرتب می‌کند و تحت عنوان یک لیست جدید برمی‌گرداند. در شرط تواناسازی انتقال‌های حرکت روبه‌جلو و عقب تمام بلوک‌ها در شکل ۵، ابتدا با فراخوانی تابع select اطلاعات مربوط به موقعیت‌های مکانی یک بلوک خاص از میان اطلاعات کلیه بلوک‌ها که به‌عنوان وضعیت سیستم در مکان SystemState ذخیره شده است استخراج می‌شود. اطلاعات استخراج‌شده به‌صورت یک لیست از نوع رنگ POSITION خواهد بود. سپس این لیست به‌عنوان ورودی به تابع descendingSort داده خواهد شد تا اطلاعات مکانی آن بلوک به‌این‌ترتیب نزولی اندیس خانه‌هایی از جدول جورچین که توسط آن بلوک پرشده است مرتب شوند.

نمایه تابع getOrientation به شرح زیر است:

```
fun getOrientation( L : POSITION ): ORIENTATION
```

این تابع فهرستی از اطلاعات مکانی فقط یک بلوک را دریافت می‌کند. اگر بلوک فقط یک خانه از جدول را اشغال کرده باشد در این صورت بلوک مربعی بوده و افقی یا عمودی نیست. در این حالت تابع مقدار Undefined را برمی‌گرداند. در غیراین‌صورت اطلاعات موقعیتی دو خانه اول از لیست را برمی‌دارد. اگر اندیس مکانی این دو خانه طوری باشد که موقعیت آن‌ها یک واحد باهم اختلاف داشته باشد درنتیجه بلوک افقی است و در غیر این صورت بلوک عمودی است و به‌عنوان نتیجه، راستای قرارگیری بلوک را برمی‌گرداند. همیشه اطلاعات مکانی یک بلوک خاص، پس از مرتب شدن به‌صورت نزولی بر اساس اندیس خانه‌های پرشده؛ به‌عنوان ورودی به این تابع داده می‌شود تا به‌طور خودکار راستای قرارگرفتن بلوک در جورچین استخراج شود.

کد تابع isOccupied به شکل زیر است:

```
fun isOccupied(L: POSITION, Loc: INT, Block: INT) : BOOL
=
 let
     val  i = ref 0
     val  t = ref {loc = ~1, blk = ~1 }
     val  n = List.length(L)
     val  Res = ref false
 in
    i := 0;
    while  !i < n andalso !Res = false do(
       t := List.nth( L, !i );
       if #blk( !t ) <> Block andalso  #loc( !t )= Loc  then
                                  Res := true
       else  ();
       i := !i + 1
        );
     !Res
 end
| isOccupied( [] , _, _ )= true
```

پارامتر اول این تابع، لیست اطلاعات تمام بلوک‌ها و پارامتر دوم آن شماره یک مکان و پارامتر سوم آن شماره یک بلوک است. اگر موقعیت خواسته‌شده توسط بلوک دیگری غیر از بلوک مشخص‌شده در پارامتر سوم پرشده باشد مقدار صحیح و در غیراین‌صورت مقدار غلط برمی‌گرداند. اگر موقعیت خواسته‌شده توسط خود بلوک پرشده باشد مقدار غلط برمی‌گرداند.

نمایه تابع nextMove به شکل زیر است:

```
fun nextMove(L:POSITION, D:DIRECTION,
O:ORIENTATION, S:INT): POSLIST
```

این تابع به‌عنوان پارامتر اول، لیست اطلاعات یک بلوک را دریافت می‌کند و به‌عنوان پارامتر دوم، جهت حرکت بلوک و به‌عنوان پارامتر سوم، راستای قرارگرفتن آن بلوک را دریافت می‌کند و پارامتر آخر بیان‌گر میزان حرکت بلوک برحسب قدم (خانه) است.

این تابع با استفاده از یک حلقه، قدم‌به‌قدم خانه‌های بعدی جهت حرکت بلوک را محاسبه می‌کند. اگر خانه بعدی در داخل جدول جورچین باشد اندیس این خانه را در یک لیست از اعداد صحیح از نوع

رنگ POSLIST قرار می‌دهد. اگر موقعیت یکی از این خانه‌ها خارج از جدول جورچین باشد حلقه بررسی متوقف شده و اندیس ۱- در انتهای لیست اضافه می‌شود. در غیراین‌صورت فهرستی حاوی موقعیت‌های بعدی که قرار است توسط بلوک در اثر حرکت آن پر شوند به ترتیب از اول به آخر در یک لیست قرار گرفته و به‌عنوان حاصل برگردانده می‌شود.

کد تابع canMove به شکل زیر است:

```
fun  canMove(_,[],_,_,_) = false
| canMove([],_,_,_,_)= false
| canMove( L:POSITION, LB:POSITION , D:DIRECTION,
O:ORIENTATION, S: INT): BOOL =
let
    val   headP =  List.hd (LB)
    val   tailP = List.last(LB)
    val   nextLoc = ref []
    val   Block = #blk( List.hd(LB))
    val   LastLoc = ref ~1
    val   i = ref 0
    val   n = ref ~1
    val   res = ref  true
in
    nextLoc := nextMove(LB, D, O,S);
    if !nextLoc = [] then false
    else (
            LastLoc := List.last(!nextLoc);
            n := List.length(!nextLoc);
            if ( !LastLoc <> ~1) then (
                    while !i < !n andalso !res =  true do (
                       if isOccupied(L,List.nth(!nextLoc,
                                !i), Block) then   res := false
                        else   ();
                        i := !i + 1
                      );
                     !res
                    )
             else    false
  )
end
```

این تابع به‌عنوان پارامتر اول، لیست اطلاعات تمام بلوک‌ها و به‌عنوان پارامتر دوم، لیست اطلاعات یک بلوک خاص که حرکت دادن آن تحت بررسی است را دریافت می‌کند. پارامتر سوم، جهت حرکت بلوک و پارامتر چهارم، راستای قرار گرفتن آن بلوک را بیان می‌کند و پارامتر پنجم، میزان حرکت موردنظر برای بلوک را بر اساس قدم بیان می‌کند. این تابع ابتدا با فراخوانی تابع nextMove لیست خانه‌های بعدی از جدول را که در اثر حرکت بلوک پر خواهند شد دریافت می‌کند. اگر محتویات خانه آخر لیست برابر ۱- باشد، به این معنی است که بلوک به میزان خواسته‌شده قادر به حرکت نیست و این مقدار حرکت باعث خروج بلوک از جدول جورچین می‌شود. در این صورت تابع مقدار غلط را به‌عنوان نتیجه برمی‌گرداند. در غیراین‌صورت تک‌تک خانه‌هایی که توسط حرکت بلوک قرار است پر شوند، توسط فراخوانی تابع isOccupied بررسی می‌شوند. اگر هیچ بلوکی مانع حرکت این بلوک نباشد تابع canMove مقدار صحیح و در غیراین‌صورت مقدار غلط را به‌عنوان نتیجه برمی‌گرداند.

نمایه تابع isEnabled به شکل زیر است:

```
fun isEnabled( L:POSITION, LB:POSITION, D:DIRECTION,
O:ORIENTATION, S:INT): BOOL
```

این تابع لیست اطلاعات مکانی تمام بلوک‌ها را به‌عنوان پارامتر اول و لیست اطلاعات یک بلوک خاص را به‌عنوان پارامتر دوم دریافت می‌کند. پارامتر ورودی سوم این تابع، جهت حرکت و پارامتر چهارم، راستای قرارگرفتن بلوک خاصی که اطلاعات مکانی آن به‌عنوان پارامتر دوم مشخص‌شده، است. پارامتر آخر بیانگر میزان حرکت بلوک خاص موردنظر به میزان خواسته‌شده برحسب قدم است. این تابع با فراخوانی تابع canMove بررسی می‌کند که آیا بلوک موردنظر به میزان خواسته‌شده قادر به حرکت است یا نه؟ برای آن‌که پس از خروج بلوک هدف از جورچین، اجرای مدل معما متوقف شود، لازم است که انتقال‌های تمام بلوک‌ها ناتوان شوند. برای این منظور با فراخوانی تابع select جستجو می‌کنیم که آیا اطلاعات مکانی بلوک هدف (با اندیس ۱۰۰) هنوز در وضعیت سراسری سیستم موجود است یا خیر؟ اگر بلوک هدف در جورچین باشد و بلوک موردنظر توان حرکت داشته باشد این تابع مقدار صحیح را به‌عنوان نتیجه برمی‌گرداند. این تابع مهم‌ترین تابع در شرط محافظ انتقال‌های حرکت به جلو و عقب در مدل شکل ۵ است.

نمایه تابع remove به شکل زیر است:

```
fun remove(L:POSITION, index:INT): POSITION
```

پارامتر اول این تابع لیست اطلاعات تمام بلوک‌ها و پارامتر دوم آن شماره یک بلوک است. این تابع با حذف اطلاعات بلوک مشخص‌شده در پارامتر دوم، اطلاعات سایر بلوک‌ها را به‌عنوان یک لیست جدید برمی‌گرداند. این تابع برای خارج ساختن یک بلوک از جدول جورچین مورداستفاده قرار می‌گیرد.

نمایه تابع canGo به شکل زیر است:

```
canGo(L:POSITION): BOOL
```

این تابع لیست اطلاعات مکانی یک بلوک را که راستای آن افقی است دریافت می‌کند. فرض بر این است که اطلاعات مکانی آن بلوک به ترتیب نزولی خانه‌هایی از جورچین که توسط آن بلوک پرشده، مرتب است. اگر آخرین خانه پرشده از سمت راست بلوک افقی، خانه ۱۷ جدول جورچین باشد؛ مطابق شکل ۱ این بلوک قابل‌خروج از جورچین است و این تابع مقدار صحیح برمی‌گرداند. در بازی Unblock Me به‌محض اینکه یک بلوک افقی در خانه ۱۷ قرار می‌گیرد به‌طور خودکار از جدول جورچین خارج می‌شود.

کد تابع updateLoc به شرح زیر است:

```
fun updateLoc(L : POSITION, step: INT) : POSITION =
 let
    val  i = ref 0
    val  t1 = ref {loc = ~1, blk = ~1 }
    val  t2 = ref {loc = ~1, blk = ~1 }
    val  LR = ref []
    val  n = List.length(L)
 in
  i:= 0;
  while  !i < n do (
    t1 := List.nth(L, !i);
    if #loc(!t1) = 17 andalso step = 1 then
       t2 := {loc = #loc( !t1)+ step+100, blk =#blk(!t1)}
    else
       if #loc(!t1) = 118 andalso step = ~1 then
         t2 :={loc = #loc(!t1)+ step-100, blk=#blk(!t1) }
       else
         t2 := { loc = #loc( !t1)+ step, blk = #blk( !t1) };
       LR := !LR ^^ [!t2];
       i:= !i + 1
    );
  !LR
 end
| updateLoc([],_) = [];
```

پارامتر اول این تابع، لیست اطلاعات یک بلوک خاص و پارامتر دوم آن میزان تغییر موقعیت بلوک موردنظر است. درحرکت روبه‌جلو برای بلوک افقی، این میزان ۱+ و درحرکت افقی به عقب ۱- است. درحرکت روبه‌جلو این برای بلوک عمودی، میزان حرکت برابر ۶+ و درحرکت به عقب برابر ۶- است. این تابع بعد از تغییر موقعیت خانه‌های بلوک موردنظر، موقعیت جدید خانه‌ها را برمی‌گرداند.

تابع moveBlock یکی از توابع اصلی است که پس از شلیک کردن هر انتقال در شکل ۵، وضعیت بعدی سیستم را تعیین می‌کند. کد این تابع به شرح زیر است:

```
fun moveBlock( L: POSITION, LB:POSITION,
O:ORIENTATION, D: DIRECTION, blockIndex: INT, S: INT)
: POSITION=
  let
     val  step = ref 0
     val  nextPlace = ref ~1
     val  canOut = ref true
     val  LBU = ref []
     val  LR = ref []
  in
     if O=Horizontal  then
         if  D = Forward then  step := S else step := ~1*S
     else
          if D=Forward then step:= 6*S else step := ~6*S;
     LBU := updateLoc( LB , !step);
     if O = Horizontal then
              canOut := canGo(!LBU)
          else
              canOut := false;
     if !canOut = true then
              LR := remove(L, blockIndex)
          else
               LR := remove(L, blockIndex) ^^ !LBU;
     descendingSortBlock(descendingSort(!LR))
  end
| moveBlock( _,[],_,_,_,_) = []
| moveBlock( [],_,_,_,_,_) = []
```

این تابع به‌عنوان پارامتر اول، لیست اطلاعات تمام بلوک‌ها را دریافت می‌کند و به‌عنوان پارامتر دوم، لیست اطلاعات یک بلوک خاص را که قرار است حرکت کند دریافت می‌کند. پارامتر سوم، جهت حرکت بلوک و پارامتر چهارم، راستای قرار گرفتن آن بلوک را بیان می‌کند و پارامتر پنجم، اندیس بلوک و پارامتر آخر، میزان حرکت بلوک را بر اساس قدم بیان می‌کند.

هنگامی‌که این تابع فراخوانی می‌شود، مطمئن هستیم که شرط توانا شدن انتقال مربوطه، که توسط تابع isEnabled در شکل ۵ بررسی می‌شود، حتماً برقرار است. بنابراین، بلوک در راستا و جهت موردنظر به میزان خواسته‌شده قادر به حرکت است. این تابع با فراخوانی تابع updateLoc موقعیت جدید بلوک موردنظر را پس از حرکت محاسبه می‌کند. سپس با فراخوانی تابع canGo، درصورتی‌که راستای بلوک افقی باشد؛ امکان خروج بلوک از جورچین را بررسی می‌کند. اگر بلوک قابل‌خروج باشد اطلاعات مکانی آن توسط فراخوانی تابع remove از وضعیت سراسری سیستم خارج می‌شود. سپس برای کاهش فضای حالت، وضعیت جدید سیستم ابتدا به ترتیب نزولی خانه‌های پرشده و سپس به ترتیب نزولی بلوک‌ها مرتب می‌شود و حاصل به‌عنوان وضعیت بعدی سیستم نگهداری می‌شود.

در نمودار شکل ۶ سه تابع در گوشه راست بالا نمایش داده شده‌اند. این سه تابع برای بررسی فضای حالت مدل استفاده شده‌اند و در بخش بعد توضیح داده خواهند شد.

۷- تحلیل فضای حالت سیستم

گزارش گراف فضای حالت معمای ۳۲ که در شکل ۲ نمایش داده شد، در جدول ۱ ارائه شده است. هر وضعیت سیستم به‌صورت یک گره (نشانه‌گذاری) در گراف فضای حالت نمایش داده می‌شود. گره پایانی (مرده)، وضعیتی است که سیستم قادر به خروج از آن وضعیت نیست.

جدول ۱: گزارش گراف فضای حالت اجرای مدل برای معمای شماره ۳۲

تعداد گره‌ها	۲۶۱۵
تعداد یال‌ها	۲۴۱۴۲
زمان لازم برای تولید گراف	۲۷ ثانیه
وضعیت گراف	کامل
تعداد نشانه‌گذاری‌های مرده	۲۸ عدد
شماره گره‌های نشانه‌گذاری‌های مرده	۹۸۴ و ۹۷۸ و ۹۷۶ و ۹۷۳ و ۲۳۹۲ و ...

اولین مدل ایجادشده برای این بازی با مشکل انفجار فضای حالت مواجه شد؛ به‌طوری‌که تولید فضای حالت بعد از یک ساعت محاسبه به‌صورت جزئی (کامل نشده) گزارش شد. برای حل این مشکل، لیست وضعیت سیستم همان‌طور که در بخش‌های قبل توضیح داده شد، دو بار بر اساس دو معیار مختلف به‌صورت نزولی مرتب شد. این امر مشکل انفجار فضای حالت را حل کرد و زمان اجرای مدل و تولید فضای حالت را به ۲۷ ثانیه کاهش داد.

هدف از مدل‌سازی این بازی استخراج فضای حالت و تحلیل آن است. در این قسمت پاسخ سؤال‌های تحقیق که در بخش اول مقاله مطرح شدند ارائه شده است. دو نوع بن‌بست در این مقاله مطرح است که عبارت‌اند از: ۱) بن‌بست کاذب[۱۸] یا بن‌بست مدل و ۲) بن‌بست حقیقی[۱۹] یا بن‌بست معما. در مدل ارائه‌شده، درصورتی‌که بلوک هدف از مدل خارج شود تمام انتقال‌ها ناتوان می‌شوند و مدل دچار بن‌بست می‌شود. در این حالت، بن‌بست مدل یا بن‌بست کاذب رخ داده و به معنی حل شدن معما بوده و از دیدگاه تحلیل رفتاری اشکالی ندارد. بن‌بست حقیقی یا بن‌بست معما بدین معنی است که بازی قبل از پیدا کردن راه‌حل متوقف شود، که امری نامطلوب است. بررسی بن‌بست معما در سؤال اول تحقیق در بخش ۱ مطرح شده است. طبق گزارش فضای حالت جدول ۱، مدل دارای ۲۸ حالت پایانی است. با بررسی فضای حالت می‌توان نشان داد که حالت‌های پایانی (مرده) حالت‌هایی هستند که گره هدف از جورچین خارج شده است و بنابراین تمام این حالت‌های پایانی بن‌بست کاذب می‌باشند. اگر در یکی از حالت‌های پایانی، گره هدف در داخل جورچین قرار داشته باشد آنگاه بیان‌گر وجود بن‌بست در معمای مطرح شده است. فراخوانی تابع آماده ListDeadMarkings() بر روی فضای حالت تولیدشده از اجرای مدل، لیست حالت‌های پایانی را نمایش می‌دهد. حاصل اجرای این تابع بر روی معمای شماره ۳۲، فهرستی به شکل زیر است:

```
val it = [983, 978, 976, 973, 2392, 2389, 2347, 2343, 2340,
2185, 2171, 2094, 2090, 2038, 2034, 2031, 1851, 1829, 1729,
1675, 1463, 1435, 1262, 1224, 1218, 1216, 1213, 1011]: Node
list
```

برای بررسی کاذب بودن تمام حالت‌های پایانی مدل، تابع زیر برای بررسی فضای حالت تعریف شده است:

```
fun falseDeadLock(L) : BOOL =
  let
     val  n = List.length( L )
     val i = ref 0
     val  result = ref true
     val  mk = ref  [{ no = ~1 , loc = [] }]
     val  node = ref 1
  in
     i := 0;
     while  !i < n andalso !result = false do (
          node := List.nth( L , !i );
          let
              val     mrk =  List.hd(
               Mark.UnBlockMe'SystemState 1 (!node ))
            in
               if  select ( mrk , 100 ) =[]  then  ()
               else  result := false
            end;
            i := !i + 1
     );
     !result
  end
| falseDeadLock ( [] ) = false
```

این تابع لیست تمام حالت‌های پایانی فضای حالت سیستم را دریافت می‌کند. اگر تمام حالت‌های پایانی بن‌بست کاذب باشند مقدار صحیح و در غیراین‌صورت مقدار غلط را به‌عنوان حاصل برمی‌گرداند.

نتیجه فراخوانی این تابع روی گراف فضای حالت ایجادشده مربوط به معمای شماره ۳۲ نشان می‌دهد که تمام بن‌بست‌های مدل کاذب بوده و مدل دارای بن‌بست حقیقی نیست.

پاسخ سؤال دوم تحقیق این است که؛ اگر گراف فضای حالت وضعیت بن‌بست کاذب نداشته باشد بدین معنی است که، معما دارای جواب نیست. نتایج تحلیل فضای حالت نشان می‌دهد که مدل سیستم با معماهای شماره ۳۲ و ۱ دارای چندین حالت بن‌بست کاذب است و این معماها دارای راه‌حل‌های قابل‌قبول متعددی هستند.

برای تعیین میزان سختی معما که سؤال سوم تحقیق است، می‌توان سه معیار کمی را مطرح کرد. معیار اول که مهم‌ترین و بهترین معیار می‌تواند باشد تعداد حرکت‌های بلوک‌ها برای حل معما است. برای این منظور دو تابع برای تحلیل فضای حالت تعریف شده است. تابع shortestDeadState با جستجو در فضای حالت به دنبال وضعیت‌های بن‌بست کاذبی می‌گردد که طول مسیر از حالت اولیه به آن حالت کمینه باشد؛ سپس این تابع اندیس این وضعیت را برمی‌گرداند. کد این تابع به زبان ML به شرح زیر است:

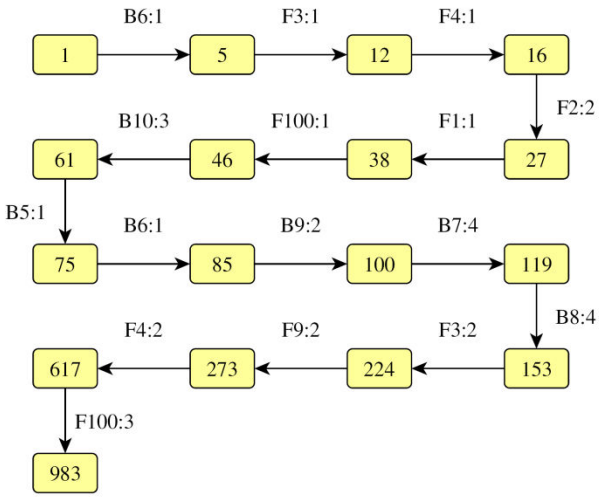
```
fun shortestDeadState(): INT =
  let
      val   pLen = ref ~1
      val   n = ref ~1
      val   L = ListDeadMarkings()
      val   len = List.length( L )
      val   i = ref  0
      val   shortPLen = ref 100000
      val   shortNIndex = ref ~1
  in
     if L = [] then 1
     else (
            while !i < len do (
                n := List.nth( L, !i);
                pLen :=  List.length( NodesInPath(1,!n));
                if  !pLen < !shortPLen then (
                                    shortPLen := !pLen;
                                    shortNIndex := !n
                                   )
                 else ();
                i := !i + 1
             );
            !shortNIndex
      )
  end
```

در یک اجرای مثال اول، عدد ۹۸۳ به‌عنوان اندیس نزدیک‌ترین حالت بن‌بست کاذب برگردانده شده است. البته در اجراهای مختلف این اندیس تغییر خواهد کرد ولی طول مسیر کمینه همیشه ثابت است. علت تغییر اندیس‌ها این است که نرم‌افزار هنگام اجرای مدل شکل ۵ به‌طور تصادفی الحاق‌ها[۲۰] را برای شلیک انتقال‌ها انتخاب می‌کند. تابع shortestPath طول مسیر بهینه را برمی‌گرداند. کد این تابع به شکل زیر است:

```
fun shortestPath() : INT = List.length(  NodesInPath ( 1,
shortestDeadState()))
```

طول این مسیر در معمای شماره ۳۲ برحسب گره، عدد ۱۷ است. این تابع اسامی گره‌های وضعیتی که منجر به حل مسئله با کوتاه‌ترین

تعداد حرکت می‌شود را ارائه می‌کند. گراف فضای حالت مسئله مطرح‌شده دارای ۲۶۱۵ گره است. بخشی از گراف فضای حالت سیستم که مربوط به مسیر بهینه است در شکل ۷ نمایش داده شده است. در این گراف برچسب یال‌ها بیان‌گر حرکت بلوک‌ها است. حرکت روبه‌جلو با کلمه F و روبه عقب با کلمه B نمایش داده شده و عدد بعدازاین حروف بیانگر شماره بلوک است. عددی که بعد از دونقطه نمایش داده شده است بیانگر میزان حرکت بلوک در راستای بیان برحسب قدم شده است.



شکل ۷: بخشی از فضای حالت مربوط به کوتاه‌ترین مسیر حل معمای شماره ۳۲

معیار کمی دوم پیشنهادی در این مقاله برای تعیین سختی یک معما، تعداد حالت‌های پایانی است که منجر به حل معما می‌شود. این کمیت معیاری برای بیان تعداد راه‌حل‌های مختلف برای حل یک معما است. تحلیل فضای حالت معمای شماره ۳۲ نشان می‌دهد که ۲۸ حالت بن‌بست کاذب وجود دارد. یعنی ۲۸ حالت پایانی مختلف منجر به حل معما می‌شود. برخلاف معیار اول که بزرگ‌تر بودن آن نشانه پیچیدگی معما است، هرقدر معیار دوم بزرگ‌تر باشد، نشانه ساده‌تر بودن معما است. به‌عنوان معیار کمی سوم می‌توان حاصل تقسیم معیار اول بر معیار دوم را به‌عنوان یک معیار ترکیبی پیشنهادی برای تعیین میزان پیچیدگی یک معما استفاده کرد.

در پاسخ به سؤال چهارم تحقیق می‌توان جواب داد که ارائه راهنمایی برای حل مسئله در هر مرحله از حل معما به‌راحتی امکان‌پذیر است. فرض کنید که بازی در وضعیت i قرار دارد. فراخوانی تابع shortestDeadState با ورودی وضعیت جاری، اندیس نزدیک‌ترین وضعیت پایانی که منجر به حل مسئله می‌شود را به ما می‌دهد. فراخوانی NodesInPath(i,shortestDeadState()) فهرستی از اندیس گره‌های بعدی برای رسیدن به نزدیک‌ترین راه‌حل را به ما می‌دهد. می‌توان با تعیین عنصر ابتدای لیست قدم بعدی از کوتاه‌ترین مسیر را برای حل معما به دست آورد. در هر حرکت، با فراخوانی تابع shortestPath می‌توان فهمید که چند قدم تا حل شدن معما باقی‌مانده است. حرکت خوب باعث خواهد شد که طول مسیر تا جواب بهینه کوتاه‌تر شود و حرکت غلط موجب افزایش طول مسیر خواهد شد. این راه‌کار پیشنهادی نشان می‌دهد که نیازهای مطرح‌شده در سؤال پنجم تحقیق به‌راحتی قابل تأمین است.

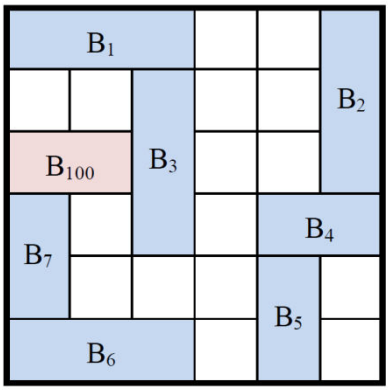
مدل پیشنهادی را با نشانه‌گذاری اولیه معمای شماره ۳۲ در نظر بگیرید. می‌توانیم با کمی تغییر بر روی این مدل، حرکت بلوک‌ها را در هر مرحله محدود به یک‌قدم نماییم. اثر این تغییر در مدل‌سازی، در گزارش فضای حالت حاصل از اجرای مدل در جدول ۲ قابل‌مشاهده است. گراف فضای حالت مدل دارای ۱۸۰۳۲ یال است که کم‌تر از تعداد یال‌های گراف فضای حالت مدل اصلی (با امکان حرکت‌های چند قدمی) که در جدول ۱ نمایش داده شده است. ولی طول کوتاه‌ترین مسیر تا حل معما برحسب تعداد گره به ۲۸ افزایش می‌یابد که باعث طولانی‌تر شدن مراحل حل معما می‌شود. به این دلیل، امکان حرکت بلوک‌ها به میزان بیشینه قدم ممکن به ازای هر حرکت، در مدل پیشنهادی شکل ۵ تعبیه شده است.

جدول ۲: گزارش فضای حالت معمای شماره ۳۲ با محدودیت حرکت یک‌قدمی در هر مرحله

تعداد گره‌ها	۲۶۱۵
تعداد یال‌ها	۱۸۰۳۲
زمان لازم برای تولید گراف	٤ ثانیه
وضعیت گراف	کامل

شکل ۸ معمای شماره ۱ سطح مبتدی در حالت آسوده را نمایش می‌دهد. تعداد بلوک‌های این معما ۳ واحد کم‌تر از تعداد بلوک‌های معمای شماره ۳۲ که در شکل ۲ نمایش داده شده است، می‌باشد و فضای حرکتی بیش‌تری برای بلوک‌ها دارد. ولی کاربر در شروع بازی احساس می‌کند که معمای اولین بازی سخت است.

نتیجه فراخوانی تابع falseDeadLock روی گراف فضای حالت ایجادشده مربوط به معمای شماره ۱ نشان می‌دهد که تمام بن‌بست‌های مدل کاذب بوده و مدل دارای بن‌بست حقیقی نیست.



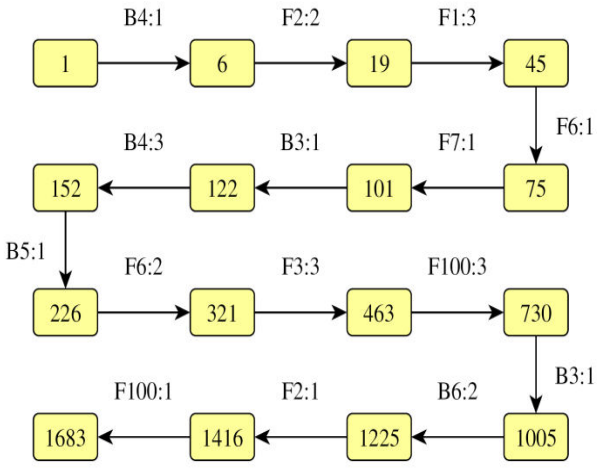
شکل ۸: وضعیت اولیه جورچین شماره ۱ بازی **Unblock Me**

به دلیل کاهش تعداد بلوک‌ها در معمای شماره ۱، لازم است که تعداد انتقال‌های جایگزینی در مدل شکل ۴ به ۸ (تعداد بلوک‌های معمای جدید) کاهش داده شود. نتایج تحلیل فضای حالت معمای شماره ۱، توسط مدل سیستم در جدول ۳ نمایش داده شده است. با توجه به آزادی حرکت بلوک‌ها، منطقی است که تعداد حالت‌های سیستم که به شکل یال‌های گراف فضای حالت ظاهر می‌شود و همچنین تعداد حالت‌های پایانی سیستم زیاد باشد. ولی انتظار داریم که تعداد حرکت‌ها برای حل معما کاهش یابد.

جدول ۳: گزارش فضای حالت معمای شماره ۱ بازی

تعداد گره‌ها	۳۹۵۰
تعداد یال‌ها	۳۶۶۰۶
زمان لازم برای تولید گراف	۳۵ ثانیه
وضعیت گراف	کامل
تعداد نشانه‌گذاری‌های مرده	۳۰۶ عدد
شماره گره‌های نشانه‌گذاری‌های مرده	۳۹۵۰ و ۳۹۴۹ و ۳۹۴۷ و ۳۹۴۶ و ...

شکل ۹ نتایج جستجو در فضای حالت، برای پیدا کردن کوتاه‌ترین راه‌حل برای معمای شماره ۱ را که از ۱۶ گره تشکیل شده نمایش می‌دهد. تحلیل فضای حالت معماهای شماره ۱ تا ۳۲ بازی Unblock Me نشان می‌دهد که پیچیدگی معماها به‌طور تناوبی تغییر می‌کند.



شکل ۹: بخشی از فضای حالت مربوط به کوتاه‌ترین مسیر حل معمای شماره ۱

۸- نتیجه‌گیری

شبکه پتری رنگی به دلیل داشتن توانایی مدل‌سازی سلسله مراتبی، قابلیت مدل‌سازی طیف وسیعی از سامانه‌ها و امکان استفاده از زبان هوش مصنوعی سطح بالا یک روش رسمی بسیار مناسب برای تحلیل رفتاری بازی‌های رایانه‌ای با فضای حالت گسسته است. در این مقاله، مدل‌سازی بازی‌های رایانه‌ای و تحلیل رفتاری آن‌ها به کمک شبکه پتری رنگی سلسله مراتبی موردبررسی قرار گرفت. یک بازی جورچین معروف از نوع طراحی مسیر بنام Unblock Me به‌عنوان نمونه مدل‌سازی شده و ارائه شد. مدل پیشنهادی پس از اجرای خودکار بازی، از طریق تولید فضای حالت سیستم، معمای مطرح‌شده را حل می‌کند و قادر به اثبات ویژگی‌های رفتاری مدل است. در این مقاله دو راهکار موفق عملی برای مقابله با مشکل انفجار فضای حالت مدل و افزایش سرعت اجرای آن پیشنهاد شد. مدل ارائه‌شده با توجه به نیازهای تحلیل رفتاری موردنظر تدوین شده و توابع مناسبی جهت تحلیل گراف فضای حالت سیستم ارائه شده است. مدل پیشنهادی قابل توسعه بوده و می‌تواند به‌عنوان نمونه، برای مدل‌سازی سامانه‌های مشابه به کار برده شود. با استفاده از مدل ارائه‌شده می‌توان میزان پیچیدگی معماهای جدید طراحی‌شده برای بازی Unblock Me را تحلیل کرده و به‌صورت کمی بیان کرد. رویکرد مدل‌سازی و تحلیل رفتاری این امکان را می‌دهد که مشکلات احتمالی بازی‌های جدید را در مرحله طراحی کشف و رفع نمود. مدل‌ها توانایی انجام تحلیل‌های متنوع و پیچیده‌ای را برای ما میسر می‌کنند که در این مقاله بررسی دو نمونه از آن‌ها ارائه شد.

مراجع

[1] K. Jensen and L.M. Kristensen, *Coloured Petri Nets Modelling and Validation of Concurrent Systems,* Springer, Berlin, Heidelberg, 2009.

[2] L.C. Paulson, *ML for the Working Programmer*, 2nd ed., NY: Cambridge university press, USA, 1996.

[3] M. Beaudouin-Lafon, W.E. Mackay, P. Andersen, P. Janecek, M. Jensen and et al., "CPN/Tools: A Post-WIMP Interface for Editing and Simulating Coloured Petri Nets," *Lecture Notes in Computer Science, Applications and Theory of Petri Nets*, Springer, Berlin Heidelberg, pp. 71-80, 2001.

[4] M. Araujo and L. roque, "Modeling Games with Petri Nets," *Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory*, September, Brunel University, vol. 5, 2009.

[5] M.A. Syufagi, M. Hariadi, and M.H. Purnomo, "Petri Net Model for Serious Games Based on Motivation Behavior Classification," *International Journal of Computer Games Technology*, vol. 2013, 2013.

[6] S. Ha and H.W. Suh, "A Timed Colored Petri Nets Modeling for Dynamic Workflow in Product Development Process," *Computers in Industry*, vol. 59, no. 2, pp. 193-209, 2008.

[7] A. Bożek, "Using Timed Coloured Petri Nets for Modelling, Simulation and Scheduling of Production Systems," *Production Scheduling, InTech, Rijeka (Croatia)*, pp. 207-230, 2012.

[8] S. Pashazadeh, "Modeling and Verification of Deadlock Potentials of a Concurrency Control Mechanism in Distributed Databases Using Hierarchical Colored Petri Net," *International Journal of Information and Education Technology (IJIET)*, vol. 2, no. 2, pp. 77-82, 2012.

[9] S. Tafkiki Alamdari, S. Pashazadeh and H. Mirzamohammadzadeh, "Security Analysis of CARBAC Model in Pervasive Computing Environment Using Colored Petri Net," *Journal of Computing*, vol. 4, no. 7, pp. 61-69, 2012.

[10] S. Pashazadeh, "Modeling and Verification of Access Rights in Take-Grant Protection Model Using Colored Petri Nets," *International Journal of Information & Network Security (IJINS)*, vol. 2, no. 1, pp. 413-425, 2013.

زیرنویس‌ها

۱ Hierarchical Colored Petri Net
۲ Formal Methods
۳ Deadlock
۴ Bag Theory
۵ Lambda-Calculus
۶ Path Planning
۷ Learning Vector Quantization (LVQ)
۸ Target Block
۹ Relax Mode
۱۰ Tokens
۱۱ Enumerated
۱۲ Initial Marking
۱۳ Substitution Transitions
۱۴ Socket
۱۵ Guard Condition
۱۶ Inscription
۱۷ Structure Chart (STC)
۱۸ False Deadlock
۱۹ True Deadlock