ششمین کنفرانس بین‌المللی
«بازی‌های رایانه‌ای؛ فرصت‌ها و چالش‌ها»
۳۰ بهمن و ۱ اسفند ۱۳۹۹ – دانشگاه اصفهان

تولید محتوای جدید بازی‌های رایانه‌ای با استفاده از برنامه‌سازی احتمالاتی

زهرا امیرجان [۱] و مرتضی دُرّی‌گیو [۲،*]
۱. دانشجوی کارشناسی ارشد هوش مصنوعی، دانشگاه سمنان
zahra.amirjan@semnan.ac.ir
۲. استادیار دانشکده مهندسی برق و کامپیوتر، دانشگاه سمنان
dorrigiv@semnan.ac.ir

چکیده
بازی‌های ویدئویی یکی از بخش‌های مهم و سودآور در صنعت سرگرمی هستند. در عین حال، معمولاً هزینه تولید این دسته از بازی‌ها بسیار زیاد است. از این‌رو، در طول دو دهه اخیر، استفاده از روش تولید محتوای رویه‌ای به عنوان وسیله‌ای برای کاهش هزینه‌ها با استفاده از تکنیک‌های الگوریتمی برای تولید خودکار برخی از محتویات بازی، مورد توجه صنعت بازی‌سازی بوده است. از سوی دیگر، تولید محتوای رویه‌ای یک زمینه روبه‌رشد تحقیقاتی است که بر روی استفاده از هوش مصنوعی در طراحی و ایجاد محتوا (مثلاً مراحل بازی، محیط‌ها، داستان‌ها و غیره)، و اغلب برای بازی‌های ویدئویی متمرکز شده است. مدل‌های مارکوف به عنوان یک روش تولید محتوای رویه‌ای مبتنی بر احتمالات، برای مدل‌سازی و تولید محتوای بازی‌ها مورد استفاده قرار می‌گیرد. یکی از متداول‌ترین روش‌های مورد استفاده در تولید مرحله بازی، زنجیره‌های چندبعدی مارکوف است. شایان ذکر است که ساخت مدل‌های احتمالاتی به‌صورت مستقیم،کاری بسیار سنگین است و پیاده‌سازی آن غالباً مشکلات زیادی به همراه دارد. در سال‌های اخیر زبان‌های برنامه‌نویسی احتمالاتی زیادی معرفی شده‌اند، که یکی از اهداف اصلی آن‌ها، حل کردن این مشکلات با ترکیب احتمالات به‌همراه قدرت محاسباتی زبان‌های برنامه‌نویسی بوده است. نوآوری اصلی این مقاله، استفاده از یک زبان برنامه‌نویسی احتمالاتی برای تولید خودکار مراحل بازی است. در روش پیشنهادی این مقاله، تولید مراحل بازی Super Mario Bros مبتنی بر زنجیره‌های چندبعدی مارکوف، با استفاده از یکی از زبان‌های برنامه‌نویسی احتمالاتی به نام Figaro انجام گرفته است. مزیت اصلی روش پیشنهادی، امکان تولید مراحل به صورت برخط، در حین انجام بازی است. به علاوه، شواهد نشان می‌دهد که با حفظ کارایی برابر در تولید مراحل با روش‌های پیشین، امکان استنتاج بهتر و کاراتر با استفاده از الگوریتم‌های استنتاج متنوع موجود در Figaro، بر روی مدل احتمالاتی ایجاد خواهد شد.

کلمات کلیدی: تولید محتوای رویه‌ای، زنجیره‌های چندبعدی مارکوف، برنامه‌نویسی احتمالاتی، Figaro

۱–مقدمه
یکی از کارهای اصلی در توسعه یک بازی ویدئویی، تولید محتوا است. تولید محتوا بایستی برای تعریف نقشه‌ها، حوادث، شخصیت‌های غیر بازیکن و سایر اجزای گرافیکی و موسیقی انجام پذیرد. چنین فرآیند تولیدی، به دلیل پیچیدگی و نیاز به

نیروی انسانی متخصص، هزینه‌ی زیادی را به توسعه بازی ویدئویی تحمیل می‌نماید. از این‌رو، نیاز به ارائه روشی برای بهینه‌سازی این فرآیند و کاهش هزینه‌ها ضروری است. به همین دلیل، تولید محتوای رویه‌ای (PCG[1]) یک تکنیک مناسب برای تولید محتوا از جمله مراحل یک بازی ویدئویی می‌باشد. در این مقاله، تکنیک‌های تولید مراحل برای بازی‌های دوبعدی و پلتفرم (به عنوان مطالعه موردی Super Mario Bros) مورد بحث قرار گرفته است. استفاده از مدل‌های احتمالاتی، راهی برای متخصصین است تا اطلاعات را برای سیستم‌های مبتنی بر هوش مصنوعی فراهم کرده تا شرایط پیش‌بینی صحیح را برای آن‌ها فراهم سازند. یکی از روش‌های تولید مراحل بازی مبتنی بر یادگیری ماشین که بررسی شده است و از احتمالات برای یادگیری استفاده می‌کند، زنجیره‌های چندبعدی مارکوف (MdMCs[2]) است [۱]. در این روش، برای ضبط احتمال انتقال کاشی[3] به کاشی از مراحل آموزش بهره گرفته و سپس از آن احتمالات آموزش یافته برای نمونه‌برداری تولید مرحله جدید استفاده می‌کند.

یک برنامه‌ی احتمالاتی، ترکیبی از محاسبات قطعی به‌همراه ورودی‌های تصادفی است که این محاسبات باعث ایجاد یک شمای کلی درمورد داده‌ها می‌گردند. احتمالات به صورت ضمنی در این نتایج دخیل شده‌اند و نیازی به پیاده‌سازی فرمول‌های مختلف نیست. زبان‌های برنامه‌نویسی احتمالاتی (PPLs[4])، یکسری الگوریتم استنتاجی عمومی ارائه می‌دهند که می‌توانند با دخالت بسیار کم توسعه‌دهنده نتیجه‌ی مورد نظر را بازگردانند. به عبارت دیگر، این ویژگی اجازه می‌دهد تا تیم‌های مدل‌سازی و متخصصین بررسی داده‌ها را از هم مجزا نمود. یک زبان برنامه‌نویسی احتمالی برای توصیف مدل‌های احتمالی، طراحی شده است و سپس در این مدل‌ها نتیجه‌گیری و استنتاج می‌کند. این زبان‌های برنامه‌نویسی به‌طور ویژه به مدل‌های گرافی (شبکه‌های بیزین و شبکه‌های مارکوف) وابسته هستند، اما در مقابل از انعطاف‌پذیری بیشتری برخوردارند. زبان‌های برنامه‌نویسی احتمالی اغلب از یک زبان پایه گسترش می‌یابند. انتخاب زبان پایه بستگی به شباهت مدل به آنتولوژی (هستی‌شناسی) آن زبان، ملاحظات تجاری و هم‌چنین اولویت‌های خاص دامنه هدف دارد. به‌عنوان مثال، زبان‌های Dimple و Chimple براساس جاوا[5] هستند و زبان برنامه‌نویسی فیگارو[6] از اسکالا[7] گسترش می‌یابد. اگرچه برنامه‌نویسی احتمالاتی برای کار کردن نیازی به یادگیری ماشینی ندارد، اما می‌تواند راه را برای ساده کردن یادگیری ماشینی فراهم کند. در یادگیری ماشینی سنتی، برای افزایش کارایی، اقدام به جمع‌آوری داده‌های بیشتر و بیشتر می‌شود و به ماشین اجازه داده می‌شود تا یاد بگیرد و کار کند. در زبان‌های برنامه‌نویسی احتمالاتی اصول و زیربنای سیستم‌ها بر دانش بیشتر استوار است.

همان‌طور که اشاره گردید، یکی از اولین روش‌های تولید مرحله مبتنی بر یادگیری ماشین که بررسی شده است، زنجیره‌های چندبعدی مارکوف MdMCs است که برای ضبط احتمال انتقال کاشی به کاشی از مراحل آموزش استفاده می‌کند. در ادامه این روش، از آن احتمالات آموزش یافته برای نمونه‌برداری تولید مرحله جدید استفاده می‌شود. روش بعدی، استفاده از رویکرد مارکوف سلسله‌مراتبی HMdMC است [۲] که از خوشه‌بندی به منظور شناسایی ساختارهای سطح بالا در داده‌های آموزش، و سپس نمونه‌برداری نقشه‌های جدید، در دو سطح انتزاع استفاده می‌کند. این رویکرد برای مدل‌سازی وابستگی‌های

[1] Procedural Content Generation
[2] Multi-dimensional Markov Chains
[3] Tile
[4] Probabilistic Programming Languages
[5] Java
[6] Figaro
[7] Scala

طولانی‌تر بین کاشی‌ها مناسب هست [۱]. در نهایت، یک روش تصادفی مارکوف (MRF[1]) معرفی شده است که احتمال کاشی‌های درون بازی را با توجه به همسایگان اطراف آن که در آن موقعیت قرار گرفته‌اند، بدون جهت‌گیری ضبط می‌کند [۱]. تنها الزام این تکنیک‌ها دسترسی به مجموعه‌ای از نقشه‌های آموزشی برای دامنه هدف است.

از دیگر تکنیک‌های تولید مرحله برای بازی‌ها که بررسی شده است، استفاده از n-gram [۳] برای مدل‌سازی و تولید مراحل بازی Super Mario Bros می‌باشد. در این روش، هر یک از مراحل به عنوان یک‌سری از ستون‌ها نشان داده می‌شوند و با آموزش مدل‌های آماری با مقادیر مختلف n (به عنوان مثال، ستون بعدی وابسته به تعداد مختلف ستون‌های قبلی) آزمایش می‌شوند. از تکنیک‌های دیگر استفاده از شبکه‌های عصبی LSTM[2] [۴] برای مدل‌سازی و تولید مراحل Super Mario Bros می‌باشد. از جمله روش‌های دیگر که برای تولید مراحل بازی‌های دوبعدی پلتفرم مورد استفاده قرار گرفته است، یک رویکرد rhythm-based [۵] است.

ساختار مقاله به این شرح است: در بخش دوم، به مروری بر روش‌شناسی مقاله شامل بیان مسئله پرداخته شده است. در بخش سوم برنامه‌نویسی با فیگارو و استنتاج روی مدل احتمالاتی در فیگارو اختصاص یافته است. جزئیات روش پیشنهادی در بخش چهارم توضیح داده شده است. در نهایت در بخش پنجم به نتیجه‌گیری این روش و کارهای پیشنهادی اختصاص یافته است.

۲– روش‌شناسی

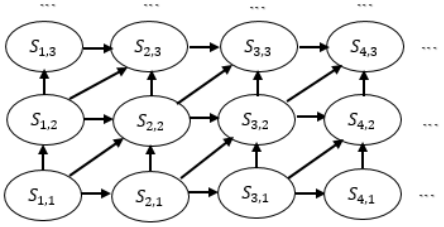
حجم وسیعی از کار بر روی نحوه تولید مراحل بازی به صورت رویه‌ای انجام گرفته است. به تازگی، تکنیک‌های یادگیری ماشین برای تولید مراحل بازی‌های ویدئویی به طور خودکار ایجاد شده‌اند. در تکنیک‌های یادگیری ماشین نیاز به جمع‌آوری داده‌های آموزشی می‌باشد. داده‌های آموزشی برای تولید مراحل جدید بازی‌ها با استفاده از روش‌های یادگیری ماشین شامل تصاویری از مراحل انواع بازی‌ها می‌باشد که این داده‌های آموزشی در مجموعه Video Game Level Corpus (VGLC) [۸] در دسترس می‌باشند. این مجموعه شامل ۴۲۸ مرحله از ۱۲ بازی می‌باشد که تصاویر مراحل بازی‌ها به همراه فایل متنی متناظر با آن‌ها در این مجموعه قرار دارد. داده‌های مورد نیاز برای آزمایش، شامل ۱۶ فایل متنی وگرافیکی از بازی Super Mario Bros می‌باشد.

۲–۱– بیان مسأله

مدل‌های مارکوف روابط احتمالاتی بین متغیرها (مانند متغیرهای قبلی در زنجیره‌های مارکوف یا متغیرهای اطراف آن در مارکوف با فیلدهای تصادفی) را در بر می‌گیرند. یک زنجیره مارکوف به عنوان یک مجموعه‌ای از حالت[3]ها به صورت $S = \{s_1, s_2, \dots, s_n\}$ و توزیع احتمال شرطی (CPD[4]) به صورت $p(s_t|s_{t-1})$ که نشان‌دهنده‌ی احتمال انتقال به حالت $s_t \in S$ از حالت قبلی آن یعنی $s_{t-1} \in S$ می‌باشد، تعریف می‌شود. یک مثال از زنجیره مارکوف را می‌توان به مدل‌سازی جملات اشاره کرد که احتمال یک کلمه خاص با توجه به کلمه قبلی، رخ می‌دهد. زنجیره‌ی مارکوف استاندارد به این صورت است که احتمال هر حالت، فقط به یک حالت قبلی آن وابسته بوده است. ولی زنجیره‌ی مارکوف مرتبه بالاتر هم وجود دارد، مانند زنجیره مارکوف مرتبه d، که در آن هر حالت به d حالت قبلی آن وابسته می‌باشد (d یک عدد طبیعی می‌باشد). در این حالت، توزیع

[1] Markov Random Field
[2] Long Short-Term Memory
[3] State
[4] Conditional Probability Distribution

احتمال شرطی آن به صورت $p(s_t|s_{t-1},\dots,s_{t-d})$ می‌باشد. زنجیره‌های مارکوف چندبعدی یک تعمیم از زنجیره‌های مارکوف مرتبه بالا است که ساختار شبکه آن به صورت یک گراف چندبعدی از حالت‌ها می‌باشد. به عنوان مثال، توزیع احتمال شرطی زنجیره مارکوف چندبعدی موجود در شکل ۱، می‌تواند به صورت $p(s_{t,r}|s_{t-1,r},s_{t,r-1},s_{t-1,r-1})$ نوشته شود (t و r نشان‌دهنده‌ی ابعاد شبکه می‌باشند به این صورت که t بیانگر شماره ستون و r بیانگر شماره ردیف می‌باشد).

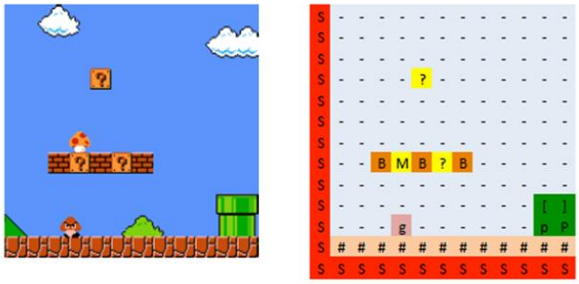


شکل ۱: ساختار شبکه زنجیره مارکوف چندبعدی مرتبه ۳ [۱]

شکل ۱، یک ساختار شبکه برای زنجیره‌ی مارکوف چندبعدی مرتبه ۳ می‌باشد. استفاده از زنجیره چندبعدی به جای یک‌بعدی، مدل را قادر می‌سازد تا به راحتی روابط را از داده‌های آموزشی دوبعدی از قبیل بافت‌های گرافیکی یا مراحل بازی‌های ویدئویی دریافت کند.

۲-۱-۱- نمایش داده‌های آموزشی

در این بخش، نحوه نمایش داده‌های آموزشی (یعنی نقشه‌های بازی)، بیان می‌شود. یک مرحله از بازی به صورت یک آرایه دوبعدی $w \times h$ نشان داده می‌شود که در آن هر سلول $m[x][y]$ یک مقدار از مجموعه‌ی کاشی‌ها که با T نمایش داده می‌شود را در بر دارد (به عنوان مثال یک کاشی می‌تواند نشان‌دهنده‌ی فضای خالی در آن مرحله بازی باشد). شکل ۲ یک بخش از مرحله بازی Super Mario Bros را که برای آموزش استفاده می‌شود و نحوه‌ی نمایش آن بخش از مرحله، به صورت یک آرایه دوبعدی را نشان می‌دهد. کاشی‌های s نشان‌دهنده‌ی مرز مراحل بازی می‌باشند.



شکل ۱: نمایش یک بخش از بازی Super Mario Bros [۱]

۲-۱-۲- آموزش مدل MdMC

برای آموزش مدل MdMC ابتدا باید ساختار شبکه، یا مجموعه‌ای از حالت‌های قبلی را که حالت فعلی به آن‌ها وابسته هست را مشخص نمود. در این آزمایشات از ساختار شبکه شکل ۱ استفاده شده است، اما می‌توان ساختار شبکه‌های دیگر (حتی پیچیده‌تر) را هم در نظر گرفت. در این صورت، هرچقدر ساختار شبکه پیچیده‌تر شود، تعداد پیکربندی کاشی‌های ممکن بیشتر خواهد شد. این بدین معنی خواهد بود که اندازه‌ی جدول احتمالاتی که از داده‌ها برآورد می‌شود، نیز بزرگ‌تر می‌شود. در نتیجه، نیاز به داده‌های آموزشی بیشتر برای برآورد پارامترهای مدل MdMC می‌باشد. الگوریتم ۱ نشان می‌دهد که چگونه بایستی

یک جدول احتمال (پارامتر P_{ns} در خروجی الگوریتم) برای یک ساختار شبکه خاص برای مدل MdMC را از فرکانس رخدادها در مراحل آموزش ایجاد نمود.

این الگوریتم دارای سه پارامتر است: ۱- پارامتر ns: ساختار شبکه زنجیره مارکوف است، ۲- پارامتر M: مجموعه‌ای از مراحل آموزش است (هر $m \in M$ مربوط به یک مرحله از مجموعه است) و ۳- پارامتر T: مجموعه‌ای از کاشی‌های موجود در نقشه‌های مراحل بازی است. شرح قدم‌های این الگوریتم در ادامه بیان شده است.

Algorithm 1 train-MdMC (ns,M,T)

1: **for** $m \in M$ **do**
2: **for** $pos \in position(m)$ **do**
3: $t = m[pos]$ //Current tile
4: $c = config(m, pos, ns)$
5: $S[c][t] = S[c][t] + 1$
6: **end for**
7: **end for**
8: **for each** $c \in C$ **do**
9: **for each** $t \in T$ **do**
10: $P_{ns}(t|c) = S[c][t]/\sum_{v \in T} S[c][v]$
11: **end for**
12: **end for**
13: **return** P_{ns}

در الگوریتم ۱، زنجیره در دو مرحله آموزش داده می‌شود: مرحله اول شمارش[1] و مرحله دوم تخمین احتمال[2]. در مرحله اول، الگوریتم برای هر موقعیت موجود در نقشه‌ی هر مرحله آموزش (m)، اجرا می‌شود (خطوط ۱-۲). دقت شود که $positions(m)$ دنباله‌ای از تمام موقعیت‌های m است. در هر موقعیت، الگوریتم نوع کاشی فعلی (t در خط ۳) و پیکربندی کاشی‌های قبلی (c در خط ۴، مربوط به ساختار شبکه (ns) در موقعیت فعلی (pos)) را تعیین می‌کند. سپس تعداد دفعاتی که t بعد از c رخ می‌دهد، شمارش می‌شود (خط ۵). شمارش کاشی در آرایه S ذخیره می‌شود که ابعاد آن، تعداد پیکربندی‌های کاشی ممکن (یعنی c) در تعداد کاشی‌های ممکن (یعنی t) می‌باشد. در مرحله دوم، احتمال هر نوع کاشی (t)، بعد از یک پیکربندی داده شده از کاشی‌های قبلی (c)، با تقسیم تعداد دفعات مشاهده شده توسط t به دنبال c به تعداد کل دفعات مشاهده شده c تعیین می‌شود (خطوط ۸-۱۲) و نتیجه در P_{ns} ذخیره می‌شود. در این آزمایش برای تخمین احتمالات از تقسیم کردن ساده استفاده شده است، اما می‌توان از روش‌های دیگری مانند smoothing نیز استفاده نمود. در نهایت، جدول احتمالات برآورد شده P_{ns} به عنوان خروجی الگوریتم بازگردانده می‌شود (خط ۱۳).

۲-۱-۳- نمونه‌برداری از مدل آموزش دیده

در مدل MdMC، نمونه‌برداری از یک مرحله جدید به معنی پرکردن یک ماتریس با کاشی‌های یک موقعیت در یک زمان با نمونه‌برداری از مدل آموزش دیده می‌باشد. گاهی اوقات نمونه‌برداری یک مرحله جدید به این شیوه، منجر به پیکربندی کاشی‌ها خواهد شد که داده‌ای برای آن وجود ندارد. در نتیجه، به این وضعیت یک حالت نادیده[3] گفته می‌شود. به عبارت دیگر، یک حالت نادیده برای مدل MdMC به عنوان یک پیکربندی از کاشی‌ها (c)، تعریف می‌شود به این صورت که $\forall_{t \in T} S[c][t] =$

[1] Absolute counts
[2] Probability estimation
[3] Unseen state

0 باشد. حالت‌های نادیده، نامطلوب هستند، زیرا هیچ داده‌ی آموزشی برای آن‌ها وجود نداشته و در نتیجه در مواجهه با این حالت‌ها مجبور به انتخاب یک کاشی به صورت تصادفی خواهیم بود. برای جلوگیری از این مسئله، الگوریتم از روش look-ahead استفاده می‌کند که به نمونه‌برداری چندین کاشی جلوتر منجر می‌شود تا جایی که تضمین کند که به هیچ حالت نادیده نرسد. اگر یک حالت نادیده با ساختار شبکه فعلی اجتناب ناپذیر باشد، الگوریتم به ساختار ساده‌تر برمی‌گردد. وقتی از یک ساختار شبکه ساده‌تر استفاده می‌شود، احتمال رخ دادن یک حالت نادیده کاهش می‌یابد. دلیل این موضوع این است که در وضعیت جدید، هر حالت به تعداد حالت قبلی کمتری وابسته هست. الگوریتم ۲ نشان می‌دهد که چگونه یک مرحله جدید نمونه‌برداری می‌شود. الگوریتم نمونه‌برداری شامل دو تابع است که یکی از آن‌ها موقعیت فعلی را در مرحله تولید شده و ساختار شبکه فعلی را مورد استفاده قرار می‌دهد، و تابع دیگر سعی می‌کند یک نوع کاشی را برای آن موقعیت انتخاب کند. اولین تابع sampleLevel (خطوط ۱–۱۱) نام دارد که دارای چهار پارامتر است:

۱–پارامتر l: که طول پیشروی (length of the look-ahead) یا تعداد کاشی پس از کاشی فعلی که الگوریتم برای حالت‌های نادیده بررسی می‌کند را نشان می‌دهد. ۲–پارامتر NS: یک فهرست مرتب شده از ساختارهای شبکه‌ای است که برای نمونه‌گیری استفاده می‌شود، که اولین ساختار، پیچیده‌ترین است، ساختارهایی که برای برگشت به ساختار ساده‌تر استفاده می‌شوند. ۳–پارامتر P: مجموعه‌ای از توزیع های احتمال شرطی است که برای هر ساختار شبکه آموزش دیده‌اند که عناصر P به صورت $P[ns] = P_{ns}$ قابل دسترسی می‌باشند. ۴–پارامتر T: مجموعه‌ای از کاشی‌های ممکن است. الگوریتم ۱، یک عنصر واحد مجموعه P را تولید می‌کند، بنابراین الگوریتم ۱ باید چند بار فراخوانی شود و خروجی‌ها در P، قبل از انتقال به الگوریتم۲ ترکیب شوند. تابع sampleLevel، با مقداردهی اولیه یک مرحله خالی شروع می‌شود (خط ۲). آرایه positions(m)، به عنوان دنباله‌ای از موقعیت‌ها در مرحله‌ی تولید شده تعریف می‌شوند به طوری که عنصر اول، موقعیت اول برای نمونه‌برداری، عنصر دوم، موقعیت دوم برای نمونه‌برداری و غیره می‌باشد. این اطلاعات به تابع دوم، sampleTile (خطوط ۱۲–۳۴) منتقل می‌شود که یک کاشی برای موقعیت داده شده را نمونه‌برداری می‌کند. تابع sampleTile دارای شش پارامتر است: ۱–پارامتر m: که مرحله جاری برای نمونه‌برداری می‌باشد، ۲–پارامتر pos: نشان‌دهنده‌ی یک موقعیت در m برای نمونه‌برداری می‌باشد، ۳–پارامتر l: طول پیشروی را در متغیر look-ahead نشان می‌دهد، ۴–پارامتر ns: ساختار شبکه فعلی است، ۵–پارامتر P_{ns} : جدول احتمالاتی مورد استفاده می‌باشد و ۶–پارامتر T: مجموعه‌ای از کاشی‌ها می‌باشد.

نمونه برداری در دو مرحله انجام می‌شود:

۱–تعیین کاشی فعلی (۱۳–۲۳) : در این مرحله یک کاشی در موقعیت فعلی نمونه‌برداری می‌شود. تابع sampleTile، پیکربندی کاشی‌های اطراف (خط ۱۷) را تعیین می‌کند و بررسی می‌کند که پیکربندی فعلی یک حالت نادیده هست یا نه (خطوط ۱۸–۲۰). اگر پیکربندی، یک حالت نادیده نیست ، پس تابع، یک کاشی را با توجه به توزیع احتمال شرطی مدل (خطوط ۲۱–۲۳) نمونه‌برداری می‌کند. اگر پیکربندی، یک حالت نادیده می‌باشد، سپس تابع با شکست مواجه می‌شود (خط ۱۹).

۲–پیش‌بینی (خطوط ۲۴–۳۴) : این مرحله، قسمت پیش‌بینی کاشی برای یک موقعیت از الگوریتم نمونه‌برداری را پیاده‌سازی می‌کند. ابتدا تابع sampleTile، در موقعیت بعدی در نقشه فراخوانی می‌شود (خط ۲۴). در هر بار فراخوانی از تابع sampleTile، اگر نتواند نمونه‌ای از یک کاشی را نمونه‌برداری کند، کاشی که در این موقعیت انتخاب شده است از مجموعه کاشی‌های ممکن حذف می‌شود (خط ۲۵) ، و یک کاشی جدید از مجموعه کاشی جدید، در جای خود نمونه‌برداری می‌شود (خطوط ۲۹–۳۱). اگر این فراخوانی به تابع sampleTile موفقیت‌آمیز باشد ، حلقه به پایان می‌رسد و sampleTile مقدار

true برمی‌گرداند(خط ۳۳). اگر در طی هر فراخوانی بازگشتی، مجموعه کاشی‌های ممکن خالی باشد، آن فراخوانی مقدار false باز می‌گرداند (خطوط ۲۶–۲۷). اگر تابع sampleTile، در سطح بالا مقدار false بازگرداند، یک ساختار شبکه ساده‌تر برای نمونه‌برداری از کاشی انتخاب می‌شود (زیرا در ساختار شبکه ساده‌تر احتمال رخداد حالت نادیده پایین می‌آید)(خط ۴).

Algorithm 2 sample-MdMC

```
1: function sampleLevel(l, NS, P, T)
2:     m = Empty map
3:     for pos ∈ positions(m) do
4:         for each ns ∈ NS do
5:             if sampleTile(m, pos, l, ns, P[ns], T) then
6:                 break
7:             end if
8:         end for
9:     end for
10:    return m
11: end function
12: function sampleTile(m, pos, l, ns, P_ns, T)
13:    if l < 0 ∨ outsideMap(pos, m) then
14:        return true
15:    end if
16:    T* = T
17:    c = config(m, pos, ns)
18:    if c is an unseen state then
19:        return false
20:    else
21:        t sampled according to P_ns(T*|c)
22:        m[pos] = t
23:    end if
24:    while ¬sampleTile(m, pos + 1, l − 1, ns, P_ns, T*) do
25:        T* = T*\t
26:        if T* = 0 then
27:            return false
28:        else
29:            t sampled according to P_ns(T*|c)
30:            m[pos] = t
31:        end if
32:    end while
33:    return true
34: end function
```

۳– برنامه‌نویسی احتمالاتی

یک رویکرد خوب برای استدلال تحت شرایط عدم قطعیت، استدلال احتمالاتی است. انواع زیادی از مدل‌های احتمالاتی وجود دارد و موارد جدید به‌طور مداوم درحال توسعه هستند. فیگارو برای کمک به ساختن و استنتاج طیف گسترده‌ای از مدل‌های احتمالی طراحی شده است. توسعه یک مدل احتمالی جدید به‌طور معمول نیازمند ایجاد یک نمایش برای مدل و یک الگوریتم استدلالی است که می‌تواند نتایج مفید را از شواهد، دریافت کند و در بسیاری از موارد، الگوریتمی برای یادگیری جنبه‌های مدل از داده‌ها فراهم آورد. این موارد، می‌توانند وظایف چالش برانگیز باشند. استدلال احتمالاتی نیازمند تلاش و تخصص قابل توجه است. علاوه براین، بیشتر ابزار استدلال احتمالاتی برای ادغام به برنامه‌های بزرگتر، مستقل و دشوار است. فیگارو، یک زبان برنامه‌نویسی احتمالی است که به هر دو این مسائل کمک می‌کند. فیگارو امکان بیان مدل‌های احتمالی را با استفاده از قدرت زبان‌های برنامه‌نویسی فراهم می‌کند و به مدل‌ساز، ابزار بیان برای ایجاد انواع مدل‌ها را می‌دهد. فیگارو با تعدادی از

الگوریتم‌های استدلال داخلی ساخته شده است که می‌توانند به‌طور خودکار به مدل‌های جدید اعمال شوند. علاوه براین، مدل‌های فیگارو، ساختار داده‌ها در زبان برنامه‌نویسی اسکالا هستند که با جاوا سازگار هستند و می‌توان آن‌ها را ساخت، مورد تغییر قرار داد و مستقیماً در هر برنامه اسکالا یا جاوا استفاده کرد. فیگارو می‌تواند طیف گسترده‌ای از مدل‌ها را شامل شود [۱۰].

فیگارو یک کتابخانه غنی از سازه‌ها برای ساخت این مدل‌ها ایجاد می‌کند و راه‌هایی برای گسترش این کتابخانه را برای ایجاد عناصر مدل خود فراهم می‌آورد. کتابخانه الگوریتم‌های استدلال فیگارو نیز قابل گسترش است. الگوریتم‌های استنتاج زیادی در فیگارو تعبیه شده‌اند [۹].

عناصر[1]، تمام ساختارهای داده‌ای یک مدل فیگارو هستند. برای تولید عناصر پیچیده‌تر می‌توان عناصر را به روش‌های مختلفی ترکیب کرد. ساده‌ترین عناصر، عناصر ساده[2] هستند که وابستگی به عناصر دیگر ندارند. عناصر مرکب[3]، عناصری هستند که از ترکیب عناصر ساده به دست می‌آیند به‌همین دلیل عناصر مرکب وابسته به عناصر دیگر می‌باشند.

۴– روش پیشنهادی

الگوریتم ۳، نشان می‌دهد که چگونه یک مرحله جدید با استفاده از برنامه‌نویسی فیگارو، نمونه‌برداری می‌شود. الگوریتم نمونه‌گیری شامل دو تابع است: ۱–تابع ProposedSampleLevel: که موقعیت فعلی را در مرحله تولید شده و ساختار شبکه فعلی، مورد استفاده قرار می‌دهد، ۲– تابع ProposedSampleTile: که سعی می‌کند یک نوع کاشی را برای آن موقعیت انتخاب کند. این دو تابع عملکردی مشابه با توابع sampleLevel و sampleTile در الگوریتم نمونه‌برداری روش مارکوف چندبعدی دارند، با این تفاوت که در این‌جا به جای متغیرها از عناصر استفاده می‌شود و عناصر، ساختار مدل احتمالاتی را می‌سازند. تابع ProposedsampleLevel که با مقداردهی اولیه یک مرحله خالی شروع می‌شود یعنی متغیر new_level_map یک آرایه‌ای از عناصر می‌باشد که در ابتدا این عناصر به طور قطعی خالی هستند (خط ۲). متغیرpositions(m) به عنوان دنباله‌ای از موقعیت‌ها در مرحله تولید شده تعریف می‌شوند. این اطلاعات به تابع دوم، ProposedSampleTile (خطوط ۸–۴۳) منتقل می‌شود که یک کاشی برای موقعیت داده شده را نمونه‌برداری می‌کند. در تابع ProposedSampleTile اگر طول پیشروی هنوز به صفر نرسیده باشد، ابتدا عنصر حاصل از ترکیب عناصر قبلی از آن موقعیت با استفاده از عنصر Apply در فیگارو مشخص می‌شود (خطوط ۱۳–۱۶) سپس بر اساس احتمالات این پیکربندی، احتمال اینکه این پیکربندی یک حالت نادیده باشد یا نه، با استفاده از عنصر Apply محاسبه می‌شود (خط ۱۷) و سپس با استفاده از الگوریتم mostLikelyValue بیشترین مقدار از آن حالت نادیده که مقدار true یا false هست مشخص می‌شود (خطوط ۱۸–۲۰) و بعد بر اساس اینکه این مقدار true یا false هست (مشابه با اینکه در تابع sampleTile در روش مارکوف چندبعدی)، پیکربندی موقعیت یک حالت نادیده باشد یا نه، رفتار می‌شود و کاشی‌ای که برای یک موقعیت انتخاب می‌شود با استفاده از عنصر Chain بر اساس مقدار عنصری که پیکربندی کاشی‌های قبلی را مشخص می‌کند، تعیین می‌شود (خطوط ۲۱–۴۳).

[1] Element
[2] Atomic
[3] Compound

Algorithm 3 ProposedSample-MdMC (Figaro)

function ProposedSampleLevel(l, NS, T)
  $new\_level\_map = Array[Element[String]]$
  for $pos \in positions(m)$ **do**
    for each $ns \in NS$ **do**
      if sampleTile2($new\_level\_map, pos, l, ns, P[ns], T$) **then**
        break
  return new_level_map
function ProposedSampleTile ($new\_level\_map, pos, l, ns, P_{ns}, T$)
  if $l < 0 \vee outsideMap(pos, new\_level\_map)$ **then**
    return true
  end if
  $T^* = T$
  $e1 = fun1(new\_level\_map, pos, ns).\_1$
  $e2 = fun1(new\_level\_map, pos, ns).\_2$
  $e3 = fun1(new\_level\_map, pos, ns).\_3$
  $str = Apply(e1, e2, e3)$
  $unseen = Apply(str)$
  $ve = MPEVariableElimination()$
  $ve.start()$
  $most\_likely\_state = ve.mostLikelyValue(unseen)$
  if most_likely_state is an unseen state **then**
    return false
  else
    $tile = Chain(str)$
    $new\_level\_map(pos) = tile$
    $ve = MPEVariableElimination()$
    $ve.start()$
    $most\_likely\_tile = ve.mostLikelyValue(tile)$
  end if
  while ¬ProposedSampleTile ($new\_level\_map, pos + 1, l - 1, ns, P_{ns}, T^*$) **do**
    $T^* = T^*$\most_likely_tile
    if $T^* = 0$ **then**
      return false
    else
      $e1 = fun1(new\_level\_map, pos, ns).\_1$
      $e2 = fun1(new\_level\_map, pos, ns).\_2$
      $e3 = fun1(new\_level\_map, pos, ns).\_3$
      $str = Apply(e1, e2, e3)$
      $tile = Chain(str)$
      $new\_level\_map(pos) = tile$
    end if
  end while
  return true

۵–نتیجه‌گیری

هدف استفاده از روش «تولید محتوای رویه‌ای» به‌کارگیری الگوریتم‌ها به منظور خودکار نمودن فرآیند تولید محتوای بازی، در جهت کاهش هزینه طراحی و توسعه بازی‌هاست. به طور معمول ساخت و تولید محتوا برای بازی‌های رایانه‌ای کاری سخت و هزینه‌بر است. همان‌طور که اهمیت تولید محتوای رویه‌ای برای توسعه بازی افزایش می‌یابد، محققان راه‌های جدیدی را برای تولید محتوای با کیفیت با یا بدون دخالت انسانی در نظر می‌گیرند؛ این مقاله به رهیافت نوین استفاده از برنامه‌سازی احتمالاتی در تولید محتوای رویه‌ای پرداخته است. در سال‌های اخیر زبان‌های برنامه‌نویسی احتمالاتی زیادی معرفی شده‌اند، که یکی از

اهداف اصلی آن‌ها، حل کردن این مشکلات با ترکیب احتمالات به‌همراه قدرت محاسباتی زبان‌های برنامه‌نویسی بوده است. نوآوری اصلی این مقاله، استفاده از یک زبان برنامه‌نویسی احتمالاتی برای تولید خودکار مراحل بازی است. در روش پیشنهادی این مقاله، تولید مراحل بازی Super Mario Bros مبتنی بر زنجیره‌های چندبعدی مارکوف، با استفاده از یکی از زبان‌های برنامه‌نویسی احتمالاتی به نام فیگارو انجام گرفته است. مزیت اصلی روش پیشنهادی، امکان تولید مراحل به صورت برخط، در حین انجام بازی است. تمرکز این پژوهش بر جنبه‌های الگوریتمی برنامه‌سازی احتمالاتی بوده، و از پرداختن به جنبه‌های کارایی اجتناب شده است.

در ادامه این پژوهش، بایستی مراحل تولید شده بازی‌ها در براساس معیارهای زیر ارزیابی شوند.

- قابلیت بازی‌پذیری (Playability (Play: بایستی تعدادی مرحله نمونه آزمایش شده است تا درصد قابلیت بازی‌پذیری برای یک پیکربندی خاص تعیین شود. برای بازی Super Mario Bros می‌توان از Adam Summerville's A* agent [7] برای تعیین Playability یک مرحله استفاده شود.
- حالت‌های نادیده (Unseen States (US: میانگین تعداد حالت‌های نادیده روی تعدادی مرحله نمونه‌برداری شده در هر دامنه و پیکربندی با استفاده از آموزش MdMC روی تعداد مراحل بازی انجام می‌شود.

شواهد نشان می‌دهد که استفاده از روش نوین برنامه‌سازی احتمالاتی، با حفظ کارایی برابر در تولید مراحل با روش‌های پیشین، امکان استنتاج بهتر و کاراتر با استفاده از الگوریتم‌های استنتاج متنوع موجود در فیگارو، بر روی مدل احتمالاتی ایجاد شود.

مراجع

[1] S .Snodgrass and S. Ontañón. "Learning to generate video game maps using markov models," *IEEE Transactions on Computational Intelligence and AI in Games*, pp. 410–422, 2017.

[2] S. Snodgrass and S. Ontañón, "A hierarchical approach to generating maps using markov chains," *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2014.

[3] S. Dahlskog, J.Togelius, and M.J. Nelson. "Linear levels through n-grams," *Proceedings of the 18th International Academic MindTrek Conference: Media Business, Management, Content & Services*, pp. 200–206. ACM, 2014.

[4] A. Summerville and M. Mateas, "Super mario as a string: Platformer level generation via lstms," arXiv preprint arXiv:1603.00930, 2016.

[5] G. Smith, M. Treanor, J. Whitehead, and M. Mateas, "Rhythm-based level generation for 2D platformers," *Proceedings of the 4th International Conference on Foundations of Digital Games*. ACM, 2009, pp. 175–182.

[6] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Searchbased procedural content generation: A taxonomy and survey," Computational Intelligence and AI in Games, *IEEE Transactions on, vol. 3, no. 3*, pp. 172–186, 2011.

[7] A. Summerville, S. Philip, and M. Mateas. Mcmcts pcg 4 smb: "Monte carlo tree search to guide platformer level generation," *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.

[8] A. J. Summerville, S. Snodgrass, M. Mateas, and Ontañón. "The VGLC: The video game level corpus," arXiv:1606.07487.

[9] https://en.wikipedia.org/wiki/Probabilistic_programming_language

[10] https://github.com/p2t2/figaro/tree/master/doc