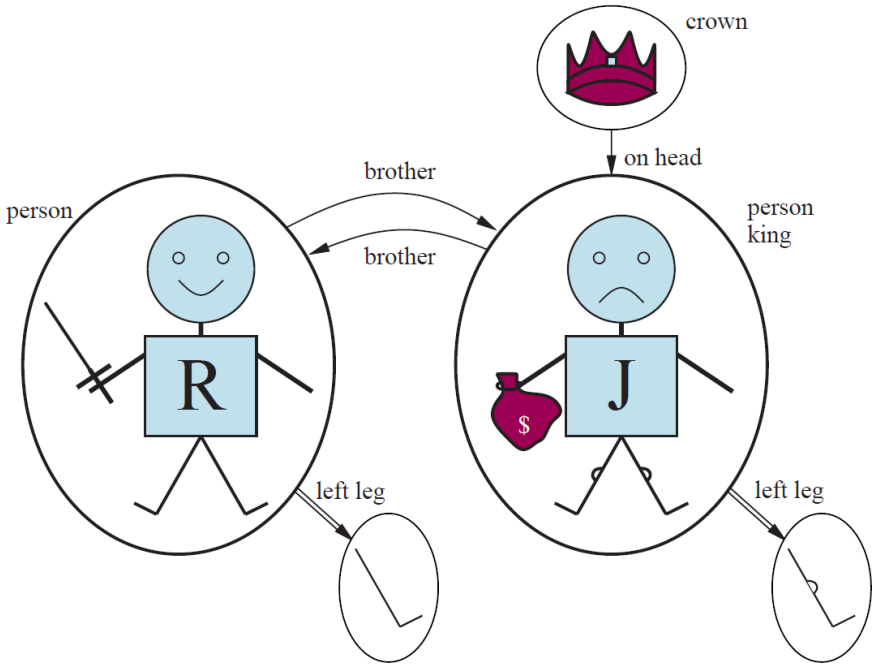
# بنام خدا

# Wumpus World

محمد معجونی – دانشگاه ارومیه



# فهرست

# چکیده

این گزارش به تفصیل به بررسی طراحی و پیاده‌سازی یک عامل هوشمند برای حل مسئله دنیای Wumpus می‌پردازد. عامل با استفاده از یک رابط کاربری گرافیکی (GUI) مبتنی بر tkinter در یک محیط x44 خانه، به کاوش، یافتن طلا و بازگشت ایمن به نقطه شروع (0,0) می‌پردازد. قابلیت‌های مهم این عامل شامل توانایی حرکت‌های قطری و استفاده استراتژیک از یک تیر برای مقابله با Wumpus است. منطق ضمنی مبتنی بر قواعد استنتاجی (برگرفته از منطق مرتبه اول) به عامل امکان می‌دهد تا با تحلیل ادراکات (بوی تعفن و نسیم) از محیط، خانه‌های امن را تشخیص داده و بهترین مسیر حرکت را انتخاب کند. این گزارش، ساختار عامل، استراتژی‌های کاوش و تصمیم‌گیری، نمایش دانش داخلی و جزئیات پیاده‌سازی را شرح می‌دهد و در نهایت به قابلیت‌های آتی و بهبودهای احتمالی اشاره می‌کند.

# مقدمه

Wumpus World یک مسئله کلاسیک و شناخته‌شده در حوزه هوش مصنوعی است که از دهه 1970 به‌عنوان یک محیط آزمایشی برای پژوهش در زمینه تصمیم‌گیری هوشمند، استنتاج منطقی، و حل مسئله مورد استفاده قرار گرفته است. این بازی ابتدا به‌عنوان یک نمونه ساده اما چالش‌برانگیز برای آزمایش الگوریتم‌های مبتنی بر منطق و دانش طراحی شد و امروزه به‌عنوان یک ابزار آموزشی برای یادگیری مفاهیم هوش مصنوعی و منطق مرتبه اول (First-Order Logic یا FOL) به کار می‌رود. هدف اصلی در Wumpus World، شبیه‌سازی یک محیط پویا و خطرناک

است که در آن یک عامل هوشمند (Agent) باید با استفاده از استنتاج‌های منطقی، مأموریت خود را به انجام برساند. در این پروژه، محیط بازی به‌صورت یک شبکه 4×4 طراحی شد که شامل عناصری مانند طلا (هدف عامل)، Wumpus (موجود خطرناک)، و گودال‌ها (Pits) است. عامل با استفاده از مشاهداتی مانند بوی بد (Stench) که نشان‌دهنده حضور Wumpus در نزدیکی است و نسیم (Breeze) که نشان‌دهنده وجود گودال است، باید تصمیمات خود را اتخاذ کند. برای این منظور، از منطق مرتبه اول استفاده شد تا دانش محیط به‌صورت قواعد منطقی مدل‌سازی شود؛ برای مثال، قاعده‌ای مانند «اگر در یک موقعیت بوی بد وجود داشته باشد، Wumpus در یکی از موقعیت‌های مجاور است» به عامل کمک می‌کند تا خطرات را شناسایی کند. این پروژه با هدف تقویت مهارت‌های عملی در زمینه برنامه‌نویسی، طراحی الگوریتم‌های هوش مصنوعی، و درک عمیق‌تر از کاربرد منطق مرتبه اول در مسائل تصمیم‌گیری اجرا شد. علاوه بر این، پروژه به دنبال ایجاد یک رابط کاربری گرافیکی تعاملی بود تا تجربه کاربری بهتری فراهم کند. برای پیاده‌سازی، از زبان برنامه‌نویسی پایتون و کتابخانه Tkinter استفاده شد که امکان نمایش بصری محیط بازی و اقدامات عامل را فراهم کرد. این پروژه نه‌تنها فرصتی برای یادگیری مفاهیم نظری هوش مصنوعی بود، بلکه به‌عنوان یک تمرین عملی برای طراحی سیستم‌های هوشمند و تعاملی نیز ارزشمند بود. در این گزارش، ابتدا شرح پروژه و وظایف تعریف‌شده ارائه می‌شود، سپس مراحل پیاده‌سازی، چالش‌های مواجه‌شده، نتایج به‌دست‌آمده، و پیشنهادهایی برای بهبود در آینده مورد بررسی قرار خواهد گرفت.

# تشریح مسئله دنیایWumpus

دنیای Wumpus یک مسئله کلاسیک در هوش مصنوعی است که در یک شبکه 4x4 خانه برگزار می‌شود. عامل بازی را از یک خانه مشخص به عنوان "خانه" (در این پیاده‌سازی، خانه (0,0)) آغاز می‌کند. این دنیا شامل عناصر زیر است:

- Wumpus (W): یک هیولای خطرناک است. ورود به خانه‌ای که Wumpus در آن قرار دارد، منجر به مرگ عامل می‌شود. وجود Wumpus با "بوی تعفن" (Stench) در خانه‌های مجاور (شامل خانه‌های قطری) قابل تشخیص است. عامل تنها یک تیر برای از بین بردن Wumpus در اختیار دارد.
- گودال‌ها (Pits): گودال‌های بی‌انتها هستند. ورود به خانه‌ای که گودال در آن قرار دارد، منجر به مرگ عامل می‌شود. وجود گودال‌ها با "نسیم" (Breeze) در خانه‌های مجاور (شامل خانه‌های قطری) قابل تشخیص است.
- طلا (Gold - G): هدف اصلی عامل است. یافتن طلا و بازگشت موفقیت‌آمیز به خانه، به معنای پیروزی در بازی است. وجود طلا با "درخشش" (Glitter) در خانه‌ای که طلا در آن قرار دارد، قابل تشخیص است (در کد فعلی، این مورد به صورت مستقیم در خانه طلا بررسی می‌شود و نیازی به ادراک Glitter جداگانه نیست).
- عامل (Agent): موجودیت هوشمندی است که وظیفه کاوش، یافتن طلا و بازگشت را بر عهده دارد.
- خانه (Home): نقطه شروع و پایان بازی برای عامل.

عامل می‌تواند به خانه‌های مجاور (افقی، عمودی و قطری) حرکت کند. دانش عامل در مورد محیط محدود به ادراکاتی (Stench، Breeze) است که از خانه فعلی خود دریافت می‌کند. این محدودیت در دانش، عامل را به استفاده از استنتاج منطقی برای درک محیط و تصمیم‌گیری سوق می‌دهد.

# طراحی و منطق عامل هوشمند

فرایند تصمیم‌گیری عامل بر اساس مجموعه‌ای از قواعد و یک استراتژی کاوش است که ایمنی و دستیابی به هدف را در اولویت قرار می‌دهد.

## ادراکات و نمایش داخلی دانش

عامل به طور ضمنی یک "حالت باور" (Belief State) در مورد دنیا نگهداری می‌کند:

- visited: مجموعه‌ای از خانه‌هایی که عامل قبلاً از آن‌ها بازدید کرده است. این مجموعه به عامل کمک می‌کند تا از کاوش مجدد خانه‌های شناخته‌شده خودداری کند و بر خانه‌های ناشناخته تمرکز کند.
- has_p: یک دیکشنری که خانه‌هایی را نشان می‌دهد که در آن‌ها "نسیم" (Breeze) ادراک شده است. این ادراک به عامل می‌گوید که احتمالاً یک گودال در یکی از خانه‌های همسایه (شامل قطری) آن خانه وجود دارد. هر کلید در این دیکشنری، مختصات (i,j) یک خانه است و مقدار True نشان‌دهنده ادراک نسیم در آن خانه است.

- has_w: یک دیکشنری مشابه has_p، که خانه‌هایی را نشان می‌دهد که در آن‌ها "بوی تعفن" (Stench) ادراک شده است. این ادراک نشان‌دهنده وجود Wumpus در یکی از خانه‌های همسایه است.

متد get_observations وظیفه جمع‌آوری ادراکات (Stench، Breeze) از خانه‌های همسایه خانه فعلی عامل را بر عهده دارد. این ادراکات مبنای استنتاج‌های بعدی عامل قرار می‌گیرند.

## استراتژی حرکت و کاوش

هدف اصلی عامل، یافتن طلا و بازگشت ایمن به خانه است. استراتژی حرکت عامل را می‌توان به فازهای زیر تقسیم کرد:

کاوش اولیه (یافتن خانه‌های امن): عامل در اولویت، به سمت خانه‌های "امن" و "بازدیدنشده" حرکت می‌کند. یک خانه زمانی امن در نظر گرفته می‌شود که:

- قبلاً از آن بازدید نشده باشد.
- هیچ یک از خانه‌های همسایه (بازدیدشده) آن دارای "بوی تعفن" (Stench) نباشد (به این معنی که Wumpus در نزدیکی نیست).
- هیچ یک از خانه‌های همسایه (بازدیدشده) آن دارای "نسیم" (Breeze) نباشد (به این معنی که گودال در نزدیکی نیست).

متد not_visited_safe() مسئول شناسایی چنین خانه‌های امنی است. اگر چندین خانه امن در دسترس باشند، عامل خانه‌ای را انتخاب می‌کند که نزدیک‌ترین فاصله را

به موقعیت فعلی او دارد (با استفاده از فاصله منهتن محاسبه می‌شود). این انتخاب هوشمندانه به عامل کمک می‌کند تا مسیرهای کاوش بهینه‌تری را طی کند.

جمع‌آوری طلا: اگر عامل به خانه‌ای برسد که حاوی طلا ("G") است، طلا را جمع‌آوری می‌کند (self.get_gold = True) و بلافاصله هدف خود را به بازگشت به خانه تغییر می‌دهد. خانه طلا پس از جمع‌آوری، خالی در نظر گرفته می‌شود. این تغییر هدف، رفتار عامل را از کاوش به بازگشت تغییر می‌دهد.

بازگشت به خانه: زمانی که طلا جمع‌آوری شد، یا اگر عامل در وضعیتی قرار گیرد که هیچ خانه امن بازدیدنشده‌ای برای کاوش وجود ندارد و تیر نیز در اختیار ندارد، به حالت "بازگشت به خانه" (self.back_home = True) می‌رود. در این حالت، عامل از یک رویکرد شبیه به جستجوی عمق اول (DFS) به نام best_next_move استفاده می‌کند تا کوتاه‌ترین مسیر بازگشت به نقطه شروع (0,0) را پیدا کند. این جستجو اولویت را به خانه‌هایی می‌دهد که قبلاً بازدید شده‌اند تا از ورود به مناطق ناشناخته و خطرناک جلوگیری شود.

استفاده از تیر: اگر عامل در شرایطی قرار گیرد که هیچ خانه امن بازدیدنشده‌ای برای کاوش وجود ندارد و هنوز یک تیر در اختیار دارد (self.arrow = True)، وارد فاز "استفاده از تیر" می‌شود.

- متد arrow_best_cell() تلاش می‌کند تا مکان احتمالی Wumpus را بر اساس Stenchهای مشاهده شده در خانه‌های بازدیدشده شناسایی کند. این

متد خانه‌ای را اولویت می‌دهد که در مجاورت بیشترین تعداد خانه‌های بازدیدشده‌ای که Stench را ثبت کرده‌اند، قرار دارد. این یک استدلال احتمالی برای مکان Wumpus است.

- اگر یک خانه امیدوارکننده برای شلیک تیر شناسایی شود، عامل سعی می‌کند به موقعیتی حرکت کند که بتواند به آن خانه شلیک کند.
- اگر عامل با موفقیت تیر را شلیک کند و Wumpus را از بین ببرد، Wumpus از شبکه حذف می‌شود و دانش has_w عامل نیز پاک می‌شود. این عمل ممکن است مسیرهای امن جدیدی را برای کاوش باز کند.

مدیریت موقعیت‌های ناامن:

- اگر عامل به خانه‌ای حرکت کند که حاوی Wumpus یا گودال باشد، بازی به پایان می‌رسد و نشان‌دهنده یک شکست است (کد ارائه شده به صراحت بازی را در این سناریوها پایان نمی‌دهد، بلکه با عدم وجود حرکات ایمن یا عواقب مستقیم، آن را ضمنی می‌کند).
- اگر هیچ خانه امن بازدیدنشده‌ای وجود نداشته باشد و عامل تیر دیگری نداشته باشد یا نتواند به طور موثر از آن استفاده کند، عامل به عنوان آخرین چاره تصمیم می‌گیرد به خانه بازگردد و اذعان می‌کند که کاوش بیشتر بسیار خطرناک است.

## نمایش دانش (منطق مرتبه اول ضمنی)

اگرچه قواعد به صراحت به عنوان گزاره‌های منطق مرتبه اول (FOL) اعلام نشده‌اند، اما منطق عامل برای تعیین خانه‌های امن و شناسایی مکان‌های Wumpus، به طور ضمنی از مفاهیم FOL استفاده می‌کند:

- ادراکات: Stench(x, y) و Breeze(x, y) گزاره‌های اساسی هستند که اطلاعات خام را به عامل می‌دهند.
- استنتاج برای گودال‌ها: Breeze(x, y) <=> (Pit(x',y') برای یک همسایه (x',y')) (اگر در خانه (x,y) نسیمی ادراک شود، به این معناست که گودالی در یکی از خانه‌های همسایه (x',y') وجود دارد، و بالعکس).
- اگر خانه‌ای (i,j) فاقد نسیم باشد، به این معنی است که تمام همسایگان (ii,jj) آن (NOT Pit(ii,jj هستند. این قاعده در not_visited_safe برای تعیین no_p استفاده می‌شود.
- استنتاج برای Wumpus: Stench(x, y) <=> (Wumpus(x',y') برای یک همسایه (x',y')) (اگر در خانه (x,y) بوی تعفنی ادراک شود، به این معناست که Wumpus در یکی از خانه‌های همسایه (x',y') وجود دارد، و بالعکس).
- اگر خانه‌ای (i,j) فاقد بوی تعفن باشد، به این معنی است که تمام همسایگان (ii,jj) آن (NOT Wumpus(ii,jj هستند. این قاعده در not_visited_safe() برای تعیین no_w استفاده می‌شود.
- تعریف خانه امن: Safe(x,y) := NOT Pit(x,y) AND NOT Wumpus(x,y) (خانه‌ای امن است اگر گودال و Wumpus در آن نباشد).

- مثال از قاعده تصمیم: IF EXISTS (Safe(x,y) AND NOT Visited(x,y)) THEN MoveTo(NearestSafe(x,y)) (اگر خانه‌ای امن و بازدیدنشده وجود دارد، به نزدیک‌ترین خانه امن حرکت کن).
- منطق استفاده از تیر: تابع arrow_best_cell سعی می‌کند خانه‌هایی را شناسایی کند که به احتمال زیاد حاوی Wumpus هستند، با یافتن خانه‌های بازدیدنشده‌ای که تمام همسایگان بازدیدشده آن‌ها بوی تعفن دارند و سپس یافتن خانه‌ای با بیشترین تعداد چنین همسایگانی. این یک استدلال اکتشافی برای ProbableWumpus(x,y) است.

## الگوریتم‌های جستجو

- best_next_move(pos, target): یک جستجوی عمق اول (DFS) ساده‌شده را پیاده‌سازی می‌کند تا کوتاه‌ترین مسیر را از pos به target در میان خانه‌های بازدیدشده (یا خود target) پیدا کند. این الگوریتم برای بازگشت به خانه یا رسیدن به موقعیت شلیک تیر بسیار حیاتی است.

# جزئیات پیاده‌سازی (پایتون با Tkinter)

شبیه‌سازی دنیای Wumpus با استفاده از کتابخانه tkinter پایتون برای رابط کاربری گرافیکی پیاده‌سازی شده است.

کلاس WumpusWorldGUI منطق بازی و عناصر GUI را در خود جای داده است.

- init: شبکه، وضعیت عامل، بوم GUI و دکمه‌های کنترلی را مقداردهی اولیه می‌کند.
- place_elements_randomly: شبکه را با Wumpus، طلا و گودال‌ها به صورت تصادفی پر می‌کند و اطمینان حاصل می‌کند که آن‌ها در خانه شروع قرار نمی‌گیرند. این متد همچنین اطلاعات has_p و has_w را برای خانه‌های مجاور گودال‌ها و Wumpusها به طور ضمنی تنظیم می‌کند.
- draw_grid: تخته بازی را رندر می‌کند و خانه‌ها را به صورت بازدیدشده (سبز روشن)، ناشناخته (خاکستری روشن) یا حاوی عناصر (قرمز برای Wumpus، آبی برای گودال، طلایی برای طلا) نمایش می‌دهد. عامل با یک دایره سبز نشان داده می‌شود.
- display_observations: ادراکات "Stench" یا "Breeze" را در خانه فعلی عامل بر اساس همسایگان آن نمایش می‌دهد.
- update_reason: دلیل حرکت اخیر عامل را در یک جعبه متنی اختصاصی نمایش می‌دهد.
- show_message: یک جعبه پیام پاپ‌آپ سفارشی برای پیام‌های مهم بازی (مانند "طلا پیدا شد!"، "خانه امنی وجود ندارد!").
- reset_game: تمام پارامترهای بازی را به حالت اولیه بازنشانی می‌کند تا بازی جدیدی آغاز شود.
- move_agent: منطق اصلی برای حرکت نوبتی عامل و تصمیم‌گیری آن است که فازهای مختلف (کاوش، جمع‌آوری طلا، استفاده از تیر، بازگشت به خانه) را هماهنگ می‌کند.

# نتیجه‌گیری

این شبیه‌سازی جامع از دنیای Wumpus، نه تنها یک راهکار عملی برای حل این مسئله کلاسیک هوش مصنوعی ارائه می‌دهد، بلکه به وضوح نشان‌دهنده پیچیدگی‌ها و ظرافت‌های رفتار یک عامل هوشمند در شرایط عدم قطعیت است. موفقیت عامل در کاوش محیط، یافتن گنج ارزشمند طلا، و سپس بازگشت ایمن به خانه، گواهی بر اثربخشی استراتژی‌های تصمیم‌گیری مبتنی بر منطق و جستجو است که در این پروژه پیاده‌سازی شده‌اند.

یکی از جنبه‌های کلیدی این پیاده‌سازی، توانایی عامل در انجام حرکات قطری است. این قابلیت، برخلاف نسخه‌های ساده‌تر دنیای Wumpus که تنها حرکات کاردینال را مجاز می‌دانند، فضای جستجو و گزینه‌های حرکتی عامل را به طرز چشمگیری گسترش می‌دهد. این آزادی عمل در حرکت، به عامل اجازه می‌دهد تا مسیرهای کوتاه‌تر و کارآمدتری را برای رسیدن به اهداف خود (چه کاوش مناطق جدید و چه بازگشت به خانه) پیدا کند، و به این ترتیب، انعطاف‌پذیری و هوشمندی بیشتری از خود نشان دهد.

علاوه بر این، مکانیزم استفاده از یک تیر تنها برای مقابله با Wumpus، لایه دیگری از پیچیدگی استراتژیک را به بازی اضافه می‌کند. این منبع محدود، عامل را وادار می‌کند تا در مورد زمان و مکان استفاده از تیر، با دقت فراوان تصمیم‌گیری کند. عامل تنها زمانی از تیر استفاده می‌کند که منطق داخلی آن به این نتیجه برسد که هیچ مسیر امن دیگری برای کاوش وجود ندارد و احتمال وجود Wumpus در یک خانه خاص به

اندازه‌ای بالاست که ارزش شلیک را داشته باشد. این تصمیم‌گیری مبتنی بر استنتاج و مدیریت ریسک، نمادی از هوش عملی عامل است که صرفاً به حرکت‌های کورکورانه اکتفا نمی‌کند.

رابط کاربری گرافیکی (GUI) پیاده‌سازی شده با tkinter نقش حیاتی در درک و تحلیل رفتار عامل ایفا می‌کند. این رابط نه تنها پیشرفت عامل را به صورت بصری و شهودی نمایش می‌دهد (از جمله خانه‌های بازدیدشده، موقعیت عناصر خطرناک و طلا)، بلکه از طریق جعبه "Movement Reasons"، دلایل منطقی پشت هر حرکت و تصمیم عامل را به وضوح توضیح می‌دهد. این شفافیت در تصمیم‌گیری، جنبه آموزشی پروژه را تقویت کرده و به کاربران کمک می‌کند تا فرایندهای فکری عامل را درک کنند، حتی اگر این فرایندها به صورت صریح با استفاده از فرمول‌های پیچیده منطق مرتبه اول کدنویسی نشده باشند. منطق کد، با قوانین ضمنی خود برای شناسایی خانه‌های امن (بر اساس عدم وجود بوی تعفن یا نسیم در همسایگی‌های بازدیدشده) و همچنین استنتاج مکان‌های احتمالی Wumpus، به طور مؤثر مفاهیم FOL را در خود جای داده است.

به طور خلاصه، این پروژه نشان می‌دهد که چگونه می‌توان با ترکیب الگوریتم‌های جستجو (مانند یک نوع DFS ساده‌شده برای یافتن مسیر)، نمایش دانش (از طریق مجموعه‌های visited و دیکشنری‌های has_p و has_w) و یک سیستم تصمیم‌گیری مبتنی بر قواعد، یک عامل هوشمند را برای پیمایش موفقیت‌آمیز در یک محیط پویا و پر از عدم قطعیت طراحی و پیاده‌سازی کرد. این شبیه‌سازی نه تنها یک

راه‌حل عملی ارائه می‌دهد، بلکه بستری عالی برای کاوش بیشتر در حوزه‌های پیشرفته‌تر هوش مصنوعی مانند استدلال احتمالی، برنامه‌ریزی و یادگیری تقویتی فراهم می‌آورد.