مرحله دوم پروژه از دو بخش تشکیل شده است. برای هر بخش دارای یک محیط جداگانه هستیم که هر محیط ویژگی‌های خاص خود را دارد. برای هر محیط باید سیاست تصمیم‌گیری استخراج شود تا بر اساس آن عامل بتواند با محیط تعامل کرده و هدف‌های تعیین شده در هر بخش را کسب کند. بخش اول یک مسأله تصادفی در قالب یک MDP و بخش دوم در رابطه با یک محیط ناشناخته ارائه شده است. توجه کنید برای اجرای این فاز باید کتابخانه pygame را نصب کنید. فایل‌های مورد نیاز را از این لینک دریافت کنید.

۱– فرآیند تصمیم مارکوف

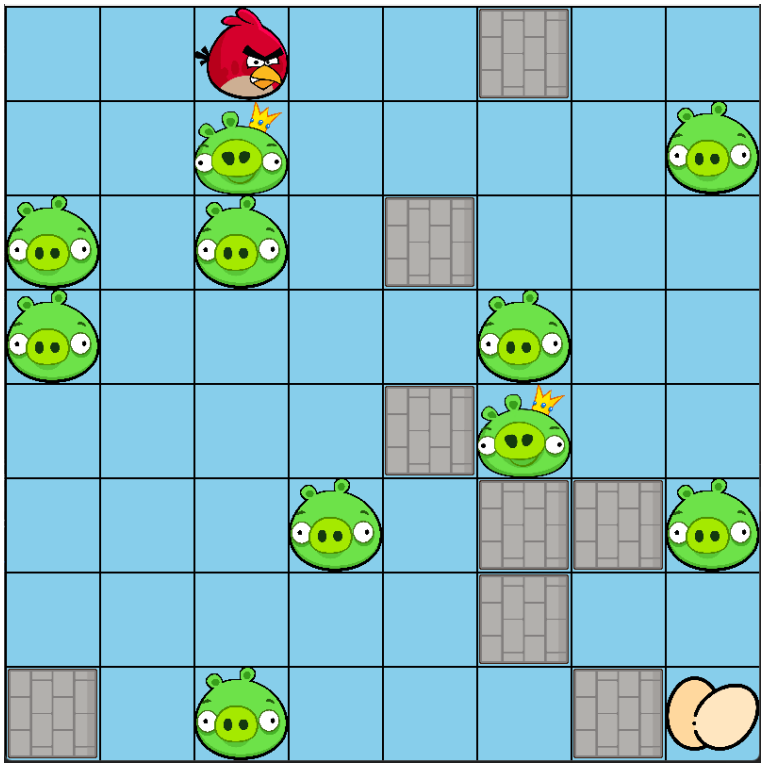
در این بخش هدف این است که مسئله را به صورت یک فرآیند تصمیم‌گیری مارکوف حل کنیم. احتمالاً با بازی معروف Angry Birds آشنایی دارید. شما باید برای پرنده عصبانی سیاستی استخراج کنید که با استفاده از آن بتواند خوک‌ها را نابود کرده و بدون برخورد با ملکه خوک‌ها به نقطه پایان بازی برسد. بازی با کسب امتیاز لازم (در نتیجه نابود کردن تعدادی از خوک‌ها) و رسیدن به تخم‌مرغ‌های موجود در انتهای صفحه بازی به پایان می‌رسد.

۱–۱–معرفی محیط

- یک محیط Grid به ابعاد 8x8 داریم. شروع فعالیت عامل (پرنده عصبانی) در نقطه (0, 0) واقع در موقعیت بالا–چپ است. تخم‌مرغ‌ها در نقطه (7, 7) در موقعیت پایین–راست قرار دارند. در صورت رسیدن عامل به تخم‌مرغ‌ها، عامل **۴۰۰ امتیاز مثبت** دریافت می‌کند.
- پرنده خشمگین دارای کنش‌های تصادفی است. کنش‌های پرنده شامل **بالا، پایین، چپ و راست** می‌باشند. عامل با هر کنشی که در محیط انجام دهد، **یک امتیاز منفی** دریافت خواهد کرد. از آنجا که محیط تصادفی است، کنش در نظر گرفته شده برای عامل با احتمال خاصی انجام می‌شود که به آن کنش اصلی می‌گوییم. در کنار کنش اصلی، عامل با احتمالات دیگری یکی از کنش‌های همسایه را ممکن است به جای کنش‌اصلی انجام دهد. برای مثال اگر کنش در نظر گرفته شده برای عامل "بالا" باشد، عامل به احتمال ۸۰ درصد به سمت بالا حرکت خواهد کرد و به احتمال ۱۰ درصد کنش "چپ" و ۱۰ درصد کنش "راست" را انجام خواهد داد. در جدول زیر کنش اصلی و کنش‌های همسایه آن مشخص شده اند. توجه کنید که احتمال کنش‌ها در بازی می‌تواند متفاوت باشد.

کنش اصلی	کنش همسایه	کنش همسایه
بالا	چپ	راست
پایین	چپ	راست
چپ	بالا	پایین
راست	بالا	پایین

- در بازی ۸ خوک وجود دارد. با هربار اجرای بازی، موقعیت خوک‌ها بطور تصادفی تعیین می‌شود. در نتیجه برخورد با خوک‌ها توسط پرنده، عامل **۲۵۰ امتیاز مثبت** کسب می‌کند.
- دو ملکه در صفحه بازی وجود دارد. با هربار اجرا شدن بازی موقعیت این دو ملکه بصورت تصادفی مشخص می‌شود. در صورت برخورد پرنده با هر ملکه، عامل **٤٠٠ امتیاز منفی** دریافت می‌کند.
- در محیط **۸ سنگ** قرار دارد که عامل نمی‌تواند از آنها عبور کند. موقعیت این سنگ‌ها با هربار اجرای بازی بصورت تصادفی مشخص خواهد شد.



شکل۱-۱محیط Angry Birds برای MDP

۱-۲-مراحل پیاده‌سازی

۱. در ابتدا باید در فایل environment.py تابع reward_function را پیاده‌سازی کنید. توجه کنید که این تابع را باید به شکلی پیاده‌سازی کنید که بر اساس امتیازات معرفی شده در زیربخش قبلی برای نزدیک شدن یا دوری از خوک‌ها، تخم مرغ و دو ملکه پاداش در نظر بگیرد. بدیهی ست که این امتیازات بنا بر نزدیکی خوکها به عامل یا نزدیکی آنها به تخم‌مرغ یا... باید تحت تاثیر قرار بگیرند و باید طوری به آنها امتیاز دهید که عامل شما در نهایت سیاستی را استخراج کند که بتواند امتیاز لازم از خوردن خوکها را بدست آورده و پس از آن به سمت تخم‌مرغ‌ها حرکت کند. شما باید این امتیازات را در قالب یک **مسئله جست‌وجو** با الگوریتم‌هایی که در بخش‌های قبلی درس آموخته اید، محاسبه کنید. توجه کنید که این پاداشی است که برای استخراج سیاست باید از آنها استفاده کنید و لزوماً برابر با امتیازاتی که عامل در حین بازی کسب می‌کند (که در بخش قبل معرفی شدند) نیست.

۲. حال باید الگوریتمی برای استخراج سیاست پیاده‌سازی کنید. برای پیاده‌سازی این الگوریتم از توابعی که برای تعامل با محیط در اختیار شما قرار داده شده استفاده کنید تا به اطلاعات مورد نیاز برای پیاده‌سازی الگوریتم دسترسی داشته باشید. توجه کنید که در این مرحله reward دریافت شده به ازای هر کنش، بر اساس همان تابع پاداشی است که در مرحله ۱ پیاده‌سازی کرده‌اید.
۳. حال پس از اینکه الگوریتم خود را پیاده‌سازی کردید، لازم است تا عامل با استفاده از سیاست استخراجی شما در محیط فعالیت کند. برای انجام اینکار نیز می‌توانید از توابعی که بصورت آماده در اختیار شما قرار داده شده است استفاده کنید.

۱-۳-ارزیابی

در ابتدا باید تابع reward_function شما و الگوریتمی که برای استخراج سیاست پیاده‌سازی کرده‌اید، بررسی شوند. سپس باید اطلاعات زیر را برای اجرای الگوریتم در اختیار ما قرار بدهید:

۱. یک Heat Map که نشان دهنده v_value به ازای هر خانه از محیط بازی می‌باشد. توجه کنید که در کنار سیاست استخراجی، الگوریتم شما باید حتما لیست *V را در خروجی خود داشته باشد. بدیهی است چون محیط دارای ۶۴ خانه خانه است، پس لیست *V نیز باید دارای ۶۴ عضو باشد . شما باید این heat map را از روی لیست *V بسازید. یک Heat Map دو بعدی که مانند صفحه بازی 8x8 باشد.
۲. حال باید در حین اجرای الگوریتم اطلاعاتی را استخراج کنید تا بتوانیم همگرایی الگوریتم شما را بررسی کنیم. بعد از هر iteration که جدول *V بروزرسانی شد، معیار زیر را محاسبه کنید. نام آن Value Difference می‌باشد.

$$Value\ Diffrence\ = \sum_{s\in S}\left|V^{(k+1)}(s) - V^{(k)}(s)\right|$$

توجه کنید که این فرمول بر روی*V قبل از بروزرسانی (اشاره به اندیس k) و پس از بروزرسانی در یک iteration (اشاره به اندیس k+1) را در نظر می‌گیرد. شما در هر iteration از اجرای الگوریتم باید این مقدار را محاسبه کرده و ذخیره کنید و در نهایت نمودار تغییرات آنرا رسم کنید. توجه کنید در صورت همگرایی الگوریتم شما، این معیار باید به مرور کاهش یابد. زمانیکه Value Difference کمتر از یک مقدار مثل ∈ بشود به معنای آن است که می‌توانیم اجرای الگوریتم را متوقف کنیم. . مقدار ∈ یک عدد بسیار کوچک (به عنوان مثال 0.01 یا 0.001) است.

۳. حال در مرحله بعد خروجی الگوریتم شما باید یک سیاست باشد که عامل بتواند با استفاده از آن در محیط فعالیت کند. به این منظور عملکرد عامل در محیط‌های مختلف (با نقشه‌های متفاوت) مورد ارزیابی قرار می‌گیرد. با توجه به تصادفی بودن محیط، برای هر محیط عملکرد عامل در ۵ دور مورد بررسی قرار می‌گیرد. در هر دور عامل با شروع از نقطه ابتدایی بازی باید سیاست بهینه را محاسبه کرده (یا با بکارگیری سیاست بهینه محاسبه شده در دورهای قبلی) و با استفاده از آن به کسب امتیاز در محیط پرداخته و در نهایت به تخم‌مرغ‌ها برسد تا آن دور از بازی به پایان برسد. میانگین امتیاز دریافت شده عامل در ۵ دور، عملکرد او را در این محیط نشان می‌دهد. در این ۵ دور ویژگی‌های محیط تغییری نخواهد کرد و بنابراین همانطور که گفته شد می‌توانید فقط در دور اول سیاست بهینه را استخراج کرده و در سایر دورها از همان سیاست استفاده کنید. این روند برای تعدادی محیط که برای ارزیابی در نظر گرفته شده است تکرار شده و بر اساس برآیند امتیازات بدست آمده مطابق جدول زیر نمره دریافتی شما مشخص می‌شود:

میانگین امتیاز دریافتی	نمره
1300 < mean score	100 %
1150 < mean score < 1300	80 %
900 < mean score < 1150	60 %
900 > mean score	50 %