دانشگاه صنعتی امیرکبیر
دانشکده مهندسی کامپیوتر

پروژه پایانی
مبانی هوش محاسباتی
(Evolutionary Games)

استاد درس: دکتر عبادزاده

بهار ۱۴۰۰

فهرست

مقدمه

توی درس با الگوریتم‌های تکاملی آشنا شدیم. یکی از کاربردهای متداول این الگوریتم‌ها، استفاده ازشون برای ساختن هوش مصنوعی توی بازی‌هاست. نمونه‌ای از پیاده‌سازی الگوریتم‌های تکاملی در بازی‌ها رو می‌تونید در اینجا ببینید.

در این پروژه قصد داریم برای یک بازی دوبعدی‌ای که خودمون پیاده‌سازی‌اش کرده‌ایم، این الگوریتم‌ها رو نوشته و روند تکامل رو در اون تماشا کنیم.

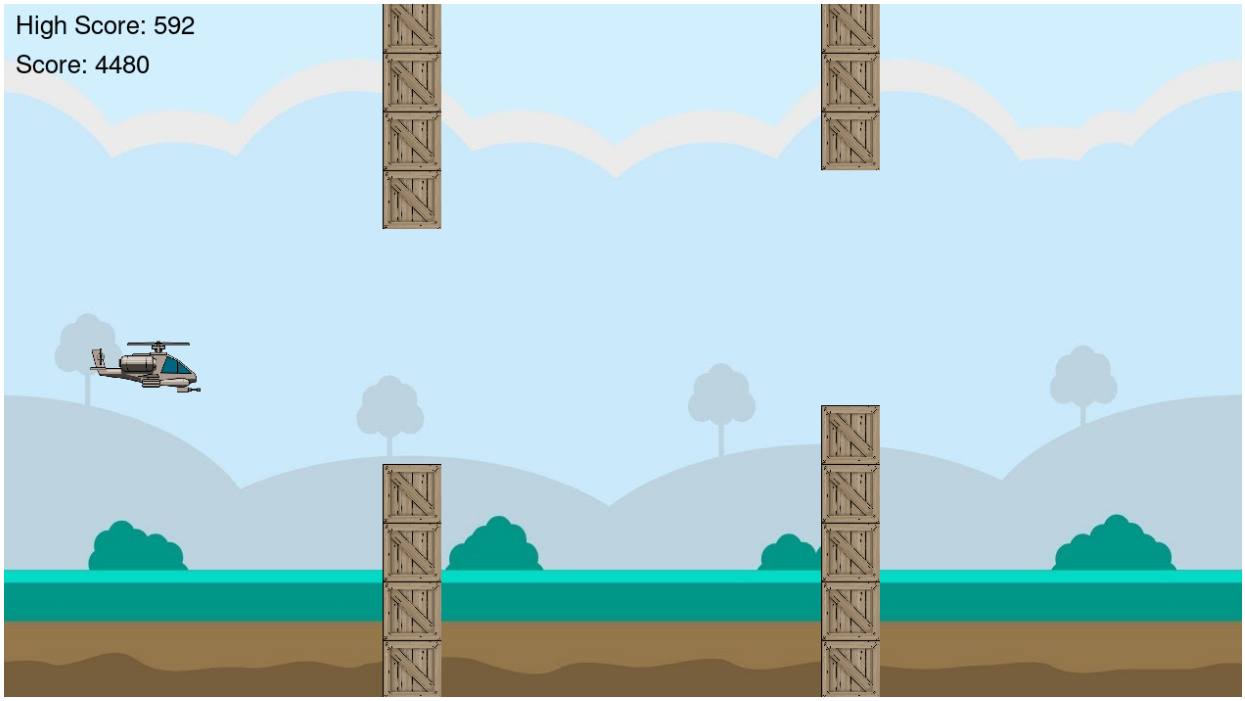
لتس پلی!

قبل از اینکه سراغ جزئیات پیاده‌سازی و روند کار بریم، بذارید با خود بازی آشنا شیم!

اول از همه، پروژه رو از این لینک دریافت کنید.
برای اجرای بازی، نیازه که کتابخونه‌ی pygame رو به کمک pip نصب کنید.
بعدش، به کمک دستورهای زیر، بازی رو توی سه‌تا mode مختلف می‌تونید اجرا کنید.

۱- مُد Helicopter:

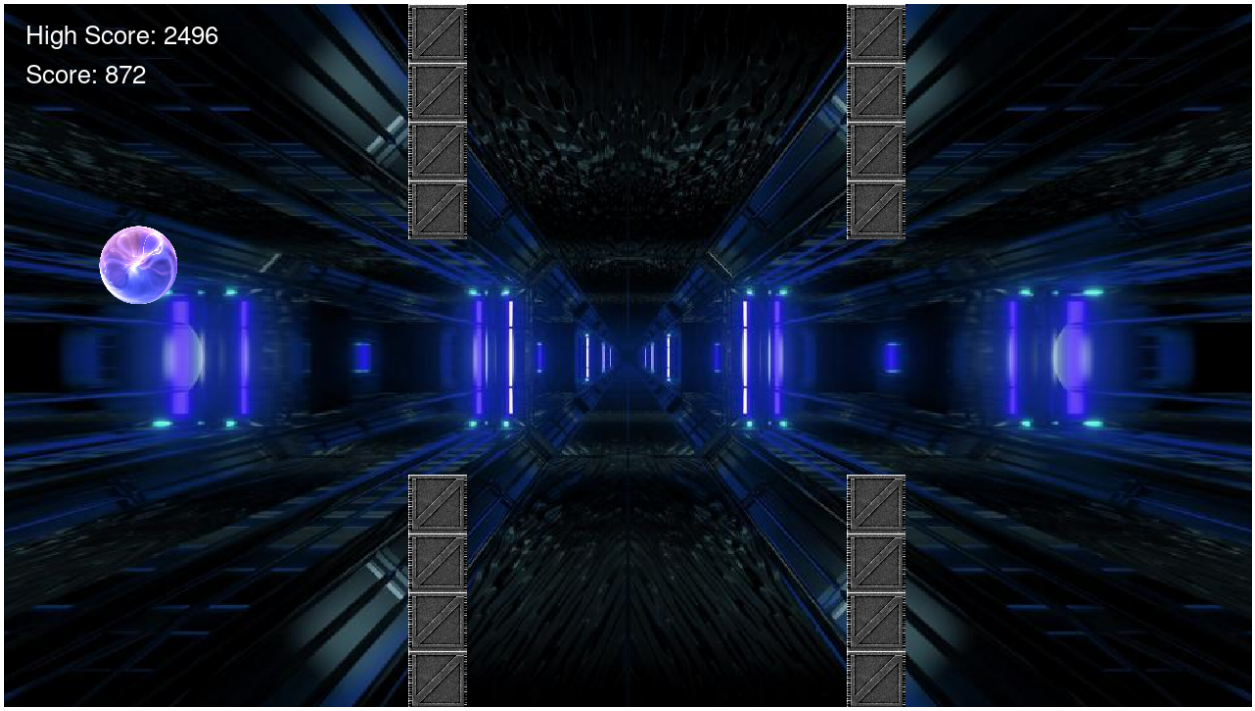
```
python game.py --mode helicopter --play True
```



با نگه‌داشتن کلید Space، هلیکوپتر به سمت بالا حرکت می‌کنه.

۲- مُد Gravity:

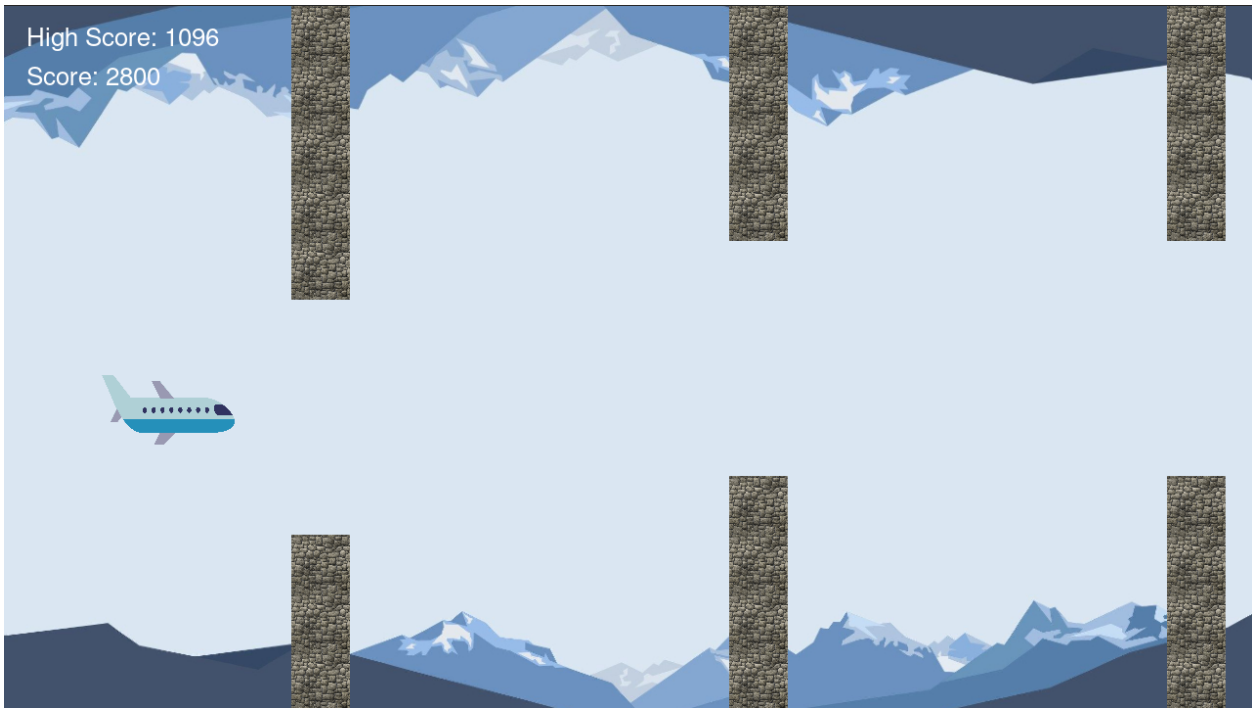
python game.py --mode gravity --play True



با کلیک کردن Space، جهت جاذبه عوض میشه.

۳- مُد Thrust:

python game.py --mode thrust --play True

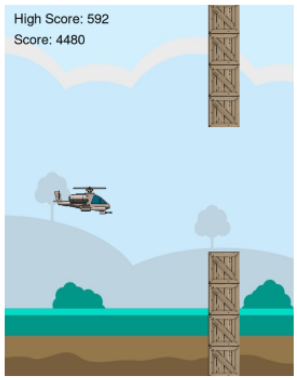


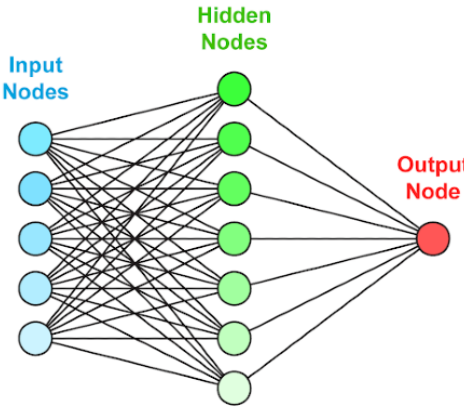
با کلیدهای Up و Down، هواپیما رو می‌تونید کنترل کنید.

شرح مسئله

حالا که یکمی با منطق بازی آشنا شدیم، وقتشه که بریم سراغ پیاده‌سازی هوش مصنوعی!

برای شروع کار، نیاز هست که یک بازنمایی مناسب از مسئله‌مون داشته باشیم. در اینجا ما یک شبکه عصبی در نظر می‌گیریم که به ما کمک میکنه بتونیم فرایند تصمیم‌گیری توی محیط رو مدل کنیم. شبکه با دریافت اطلاعات محیط، اون‌ها رو پردازش می‌کنه و بسته به اینکه چه خروجی‌ای بهمون میده، تصمیم می‌گیریم که چه کلیدی فشرده بشه.





اما سوالی که وجود داره، اینه که چجوری این شبکه عصبی‌مون رو آموزش بدیم؟

توی پروژه اول درس، شبکه عصبی‌ای زدیم که به کمکش بتونیم اعداد دست‌نوشته رو Classify کنیم. برای این کار، مجموعه‌ی بزرگی از داده‌های Label خورده داشتیم و روند کارمون بدین شکل بود که از یه شبکه‌ی رندوم شروع می‌کردیم، اجازه می‌دادیم که خروجی

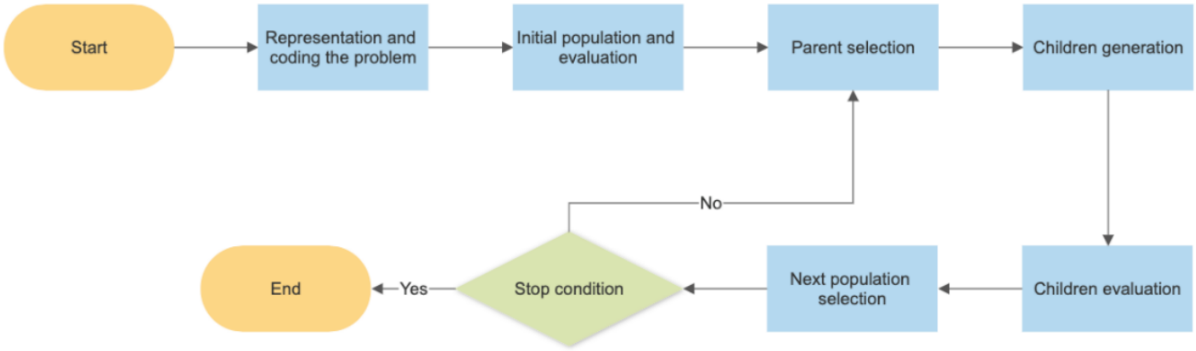
خودش رو تولید کنه، و بعد با تعریف کردن یک Cost Function، به کمک الگوریتم Backpropagation، وزن‌های شبکه رو آپدیت می‌کردیم.

اما در مسئله‌ی جدیدمون، مشکلی که وجود داره اینه که داشتن داده‌های Label خورده، معادل با اینه که ما توی مجموعه‌ی بزرگی از موقعیت‌های مختلف، به شبکه‌مون بگیم که چه کلیدی رو فشار بده! منطقا تولید کردن چنین دیتاستی عملی نیستش.
از طرف دیگه، اگر داده‌های Label خورده نداشته باشیم، دیگه نمی‌تونیم از Cost Function تعریف کردن و Backpropagation زدن استفاده کنیم.

حالا چی کار کنیم؟

میریم سراغ الگوریتم‌های تکاملی!
بدین شکل به مسئله نگاه می‌کنیم که هر شبکه عصبی رو معادل با یک موجود توی فرآیند تکامل می‌بینیم. پس برای این کار، میایم و موجودات مختلف (یعنی شبکه‌های عصبی مختلف) جنریت می‌کنیم و بهشون اجازه می‌دیم که بازی کنن. حالا با توجه به اینکه چقدر پیش‌روی داشتن توی بازی، شایستگی‌شون رو تعیین می‌کنیم. بعد باید موجودات نسل بعد رو با انتخاب یک‌سری از موجودات فعلی و تولید مثل، ایجاد کنیم. این روند رو تا جایی ادامه میدیم که موجودات‌مون بتونن خودشون رو با شرایط و فیزیک مسئله وفق بدن و در نتیجه، بتونن مسئله‌مون رو solve کنن.

شمای کلی چرخه تکامل رو در ادامه می‌بینین:



ما کد مربوط به ساختار چرخه تکامل رو برای این مسئله زدیم، و پیاده‌سازی مراحل رو به صورت یک‌سری TODO به عهده شما گذاشتیم.

در ادامه ساختار کدها و مواردی که باید انجام بدید رو شرح می‌دیم.

توجه: نیازی نیست که کدهایی که زده شده رو بخونید. هر آنچه که نیاز دارید بدونین رو توی این صورت پروژه و TODO هایی که گذاشتیم، توضیح دادیم.

توجه: مواردی که به **رنگ سبز** هستند، **امتیازی** می‌باشند.

ساختار پروژه

کد پروژه شامل ۷ تا فایل میشه:

- فایل game.py: پیاده‌سازی روند کلی اجرای بازی
- فایل player.py: آبجکت موجودات (شامل شبکه عصبی و شایستگی موجود)
- فایل evolution.py: روند تکامل موجودات هر نسل
- فایل nn.py: معماری شبکه عصبی، به همراه بخش feedforward
- فایل config.py: یک سری تنظیمات کلی بازی
- فایل util.py: برای ذخیره و لود کردن موجودات یک نسل در فایل
- فایل box_list.py: آبجکت موانع درون بازی

توجه: پیاده سازی تمام موارد زیر، برای مُد Helicopter اجباری و برای مُدهای Gravity و Thrust، امتیازی است.

موارد پروژه

مورد ۱) پیاده‌سازی شبکه عصبی (فایل nn.py):

توی init این کلاس، یک آرایه‌ی یک‌بُعدی با ۳ تا عنصر دریافت می‌شه. عنصر اول، تعداد نورون‌های لایه ورودی، عنصر دوم، تعداد نورون‌های Hidden Layer و عنصر سوم، تعداد نورون‌های لایه خروجی هستش. شما باید با توجه به این‌ها، ماتریس‌های مربوط به وزن‌ها و بایاس‌ها رو اینجا بسازید.

توی تابع activation، باید تابع سیگموید (یا غیره) رو بنویسید.
توی تابع forward، بردار ورودی رو دریافت می‌کنید و باید به کمک ضرب و جمع ماتریسی و اعمال activation در هر لایه، خروجی رو محاسبه کنید.

مورد ۲) پیاده‌سازی معماری و نحوه‌ی استفاده از شبکه عصبی (فایل player.py):

توی تابع init_network، معماری شبکه عصبی (یعنی همون ورودی‌های مربوط به مرحله قبل) مشخص شده. می‌تونید این معماری‌ها رو به طور دلخواه تغییر بدید.

نحوه تصمیم‌گیری هر موجود، توی تابع think باید پیاده‌سازی بشه. همونطور که گفتیم، این تصمیم‌گیری از طریق feedforward کردن اطلاعات محیط به شبکه عصبی انجام میشه. اطلاعاتی که به عنوان پارامتر در این تابع تعریف شده اند رو در ادامه لیست کرده‌ایم:

- mode: مُدهای سه‌گانه بازی
- box_lists: یک آرایه که هر عنصر اون، یک ستون از موانع هستش. هر ستون، یک آبجکت از کلاس BoxList هست که دارای دو فیلد x (موقعیت x ستون) و gap_mid (موقعیت y وسطِ gap ستون) می‌باشد.
- agent_position: مختصات x و y موجود
- velocity: سرعت موجود در راستای y

شما می‌تونین به هر نحو که دوست دارین، از این پارامترها برای مشخص کردن بردار ورودی شبکه‌عصبی‌تون استفاده کنین. همچنین، طول و عرض صفحه رو از CONFIG می‌تونید بگیرید. همچنین، حواستون باشه که اون اول بازی، ممکنه box_lists خالی باشه.

سپس تابع forward آبجکت self.nn رو با این بردار ورودی صدا بزنین. حالا با توجه به خروجی شبکه عصبی، باید خروجی رو بدین شکل return کنید:

- Helicopter: اگر 1 برگردونید، Space فشرده میشه و هلیکوپتر به سمت بالا میره و اگر 1- برگردونید، کلیدی فشرده نمیشه.

- Gravity: اگر 1 برگردونید، جاذبه به سمت بالا میشه و اگر 1- برگردونید، موجود به سمت پایین کشیده میشه.
- Thrust: اگر 1 برگردونید، موجود به سمت بالا سرعت میگیره، اگر 0 برگردونید، سرعتش تغییری نمیکنه و اگر 1- برگردونید، موجود به سمت پایین سرعت میگیره.

با توجه به خروجی‌ای که return کردید، موجود حرکت می‌کنه.

نکته مهم: یه نکته‌ای که حتما لازمه توجه کنید بهش، اینه که همیشه باید مقادیر ورودی شبکه‌های عصبی، نرمالایز بشن. یعنی، ورودی‌ها باید همشون توی یک range نسبتا یکسانی (مثلا بین ۰ و ۱) باشن. با تقسیم کردن هر کدام از ورودی‌ها بر ماکزیمم مقدار ممکن‌شون، می‌تونید نرمالایزیشن رو انجام بدید.

مورد ۳) پیاده‌سازی انتخاب بازماندگان (فایل evolution.py):

بعد از اینکه موجودات با توجه به شبکه‌ی عصبی مورد قبل، بازی رو انجام دادن، نگاه می‌کنیم به اینکه هر کدوم چقدر پیش‌روی داشتن، و از اون طریق، مقدار fitness شون رو حساب می‌کنیم. این کار رو ما توی تابع calculate_fitness انجام دادیم. مقدار fitness هر موجود برابر شده با delta_x ای که داشته و این مقدار توی فیلد fitness آبجکت اون player ریخته شده.

حالا شما باید تابع next_population_selection رو پیاده‌سازی کنید. در ورودی لیستی از موجودات رو دریافت می‌کنید (که هر کدوم آبجکت کلاس Player هستن)، و به کمک فیلد fitness شون، باید به تعداد num_players تا موجود از اون‌ها رو به عنوان بازماندگان توی یه آرایه برگردونید.

توجه: از اونجا که روش (μ, λ) فراموش‌کار هستش، تصمیم گرفتیم که از $(\mu + \lambda)$ استفاده کنیم. در نتیجه ورودی این تابع، اجتماع موجودات جدید و موجودات نسل قبل می‌باشند.

به عنوان روش اجباری، صرفا sort کنیدشون برحسب fitness و بهترین‌ها رو برگردونید. به عنوان امتیازی، از یک روش پیچیده‌تر که توی درس خوندید استفاده کنید.

مورد ۴) پیاده‌سازی انتخاب والدین و تولید موجودات نسل جدید (فایل evolution.py):

بعد از اینکه بازماندگان مشخص شدن، حالا وقتش هس که والدین رو از این بازماندگان انتخاب کرده و به کمکشون، نسل جدید رو بسازیم.
شما باید تابع generate_new_population رو پیاده‌سازی کنید. در ورودی، بازماندگان رو دریافت می‌کنید و به عنوان خروجی، یک آرایه به تعداد num_players از **موجودات جدید** برمی‌گردونید. به عنوان روش اجباری، برای تولید یک فرزند جدید، به صورت متناسب با

شایستگی یک والد انتخاب کنید و کپی‌ای از اون رو به عنوان فرزند در نظر گرفته (pass by reference نکنید!) و با صدا زدن تابع mutate، روی اون فرزند mutation انجام بدید.

به عنوان امتیازی، به جای انتخاب متناسب با شایستگی، از یک روش دیگر که توی درس خوندید استفاده کنید. همچنین برای تولید فرزند، می‌تونید به صورت جفت جفت والد انتخاب کرده و crossover رو نیز پیاده‌سازی کنید.

مورد ۵) پیاده‌سازی mutation (فایل evolution.py):

در این تابع، شما باید یک موجود رو بهش به عنوان ورودی بدین و پارامترهای شبکه عصبی مربوط به اون موجود رو جهش بدید. برای این کار، با یک احتمال اندک، به هر کدوم از وزن‌ها و بایاس‌های شبکه عصبی، یک نویز گاوسی اضافه کنید. میزان مناسب احتمال جهش و انحراف معیار نویز گاوسی رو خودتون باید با امتحان کردن مقادیر مختلف، tune اش کنید.

اجرای بازی:

بعد از پیاده‌سازی موارد بالا، بازی رو بدون play True-- اجرا کنید.

کول استاف!

شورت‌کات‌های کیبورد:

- با فشردن **کلید esc**، بازی بسته میشه.
- با فشردن **کلید f**، مقدار FPS بازی رو مشاهده می‌کنید.
- برای اینکه یکم سریع‌تر تکامل رو ببینید، می‌تونید در حین اجرای بازی، **کلید d** رو فشار بدید تا سرعت بازی دوبرابر بشه.
- اگر **کلید s** رو فشار بدید، به جای نشون دادن تمام موجودات نسل، تنها یکی از اون‌ها نمایش داده میشه.

اگر FPS بازی زیاد زیر ۵۰ میادش، بهتره بازی رو روی 1x ران کنید، با کلید s فقط یه موجود رو نمایش بدید یا اینکه تعداد موجودات هر نسل رو از فایل config.py کنترل کنید.

برای اینکه مجبور نشید هر بار از اول تکامل رو شروع کنید، به طور اتوماتیک هر ۵ نسل یک‌بار، موجودات اون نسل توی فایل ذخیره میشه (این فرکانس ذخیره‌سازی رو هم می‌تونید از config.py تغییرش بدید). برای اینکه تکامل از checkpoint آخر ادامه پیدا کنه، باید کامند زیر رو وارد کنید:

```
python game.py --mode $mode$ --checkpoint checkpoint/$mode$/$gen_num$
```

مثلا:

```
python game.py --mode helicopter --checkpoint checkpoint/helicopter/20
```

سایر موارد امتیازی

در کنار موارد امتیازی مطرح شده تا به اینجا، پیاده‌سازی موارد زیر نیز امتیازی هستش:

۱- Transfer Learning:

این مفهوم به این معنیه که ما یک مدل یادگیری ماشین رو روی یک تسک خاص آموزش بدیم و بعد، از اون مدل آموزش‌دیده، در جهت حل کردن یک تسک دیگه (یا همون تسک ولی در محیط دیگه‌ای) استفاده کنیم.[1]

در مسئله‌ای که ما حل کردیم، همیشه از یک نقشه‌ی یکسان استفاده می‌کردیم. حال می‌خواهیم مفهوم Transfer Learning رو روی موجودات آموزش‌دیده تست کنیم. برای این کار، به طور عادی مدل رو آموزش بدید و بعد از اینکه از خوب عمل کردن موجودات در نقشه فعلی اطمینان حاصل کردید، با تغییر نقشه و load کردن موجودات از روی checkpoint، عملکرد اون موجودات آموزش‌دیده رو روی نقشه جدید ارزیابی کنید. تغییر نقشه با تغییر پارامتر seed در فایل config.py حاصل میشه (هر عدد int ای می‌تونید قرار بدید بجاش).

[1] توی این لینک می‌تونید توضیحاتی بیشتری رو درباره این موضوع بخونید.

۲- Learning Curve:

برای تحلیل بهتر فرآیند تکامل، در هر نسل، بیشترین، کمترین و متوسط شایستگی موجودات رو محاسبه کرده و در نهایت پلات کنید. برای این کار، کدتون رو توی تابع next_population_selection در فایل evolution.py بنویسید (چون اونجا دسترسی به تمام موجودات و fitness شون داریم).

به ازای هر نسل، اطلاعات شایستگی اون نسل رو در یک فایل ذخیره کنید. سپس یک فایل پایتون جدید تعریف کنید که اطلاعات این فایل رو خونده و اون رو پلات کنه (اجرای این فایل پایتون، جدای از اجرای برنامه است).

نحوه تحویل و پل ارتباطی

پروژه رو به صورت فایل زیپ در کورسز بارگذاری کنید.

برای این پروژه، **گزارش نیازی نیست** بنویسید و بعد از ددلاین، **تحویل آنلاین** خواهیم داشت.

اگه سوالی داشتید یا به مشکلی برخوردید، توی تلگرام از $@Tavakoli\_Matin$، $@Hossein\_Zaredar$ و یا $@par\_raa$ بپرسید.