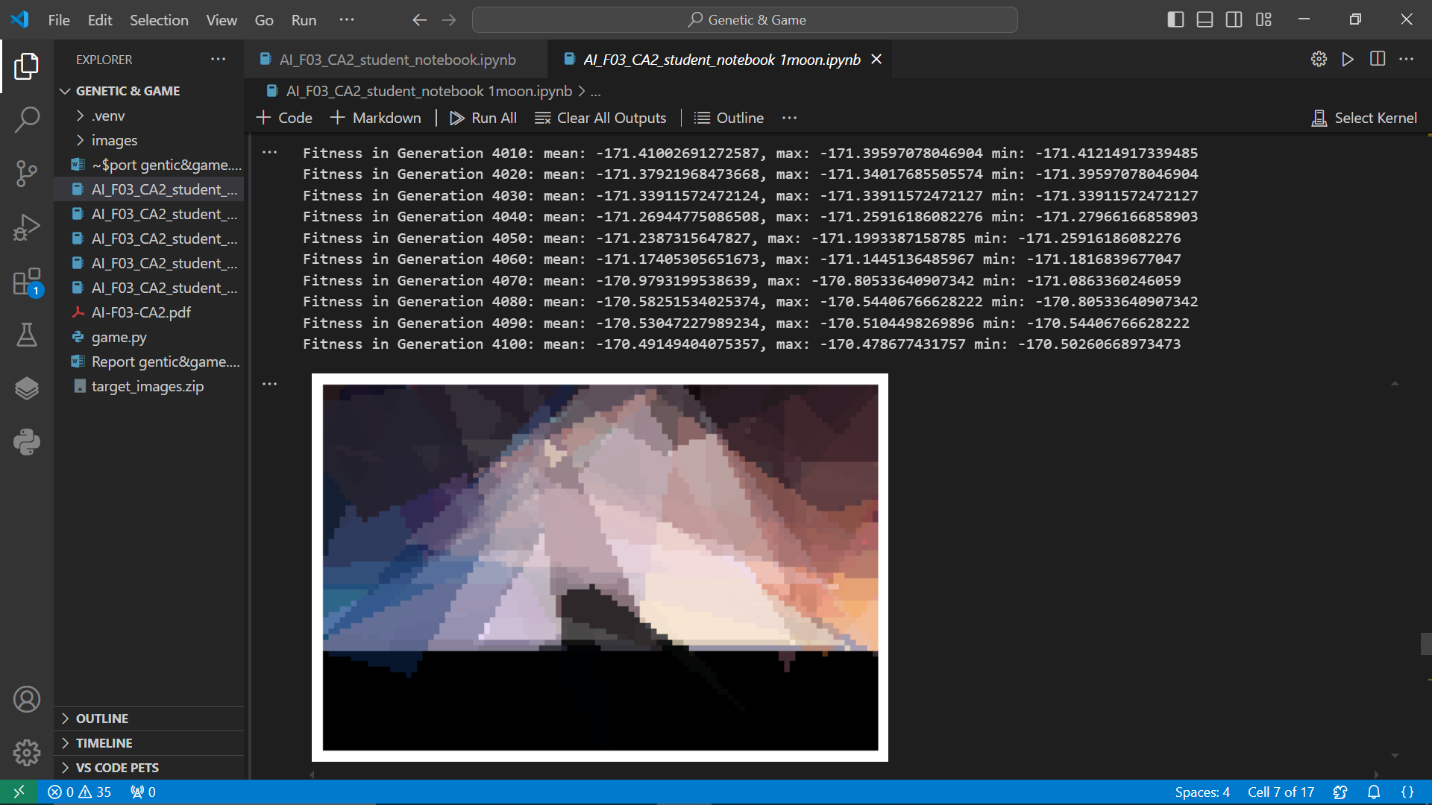
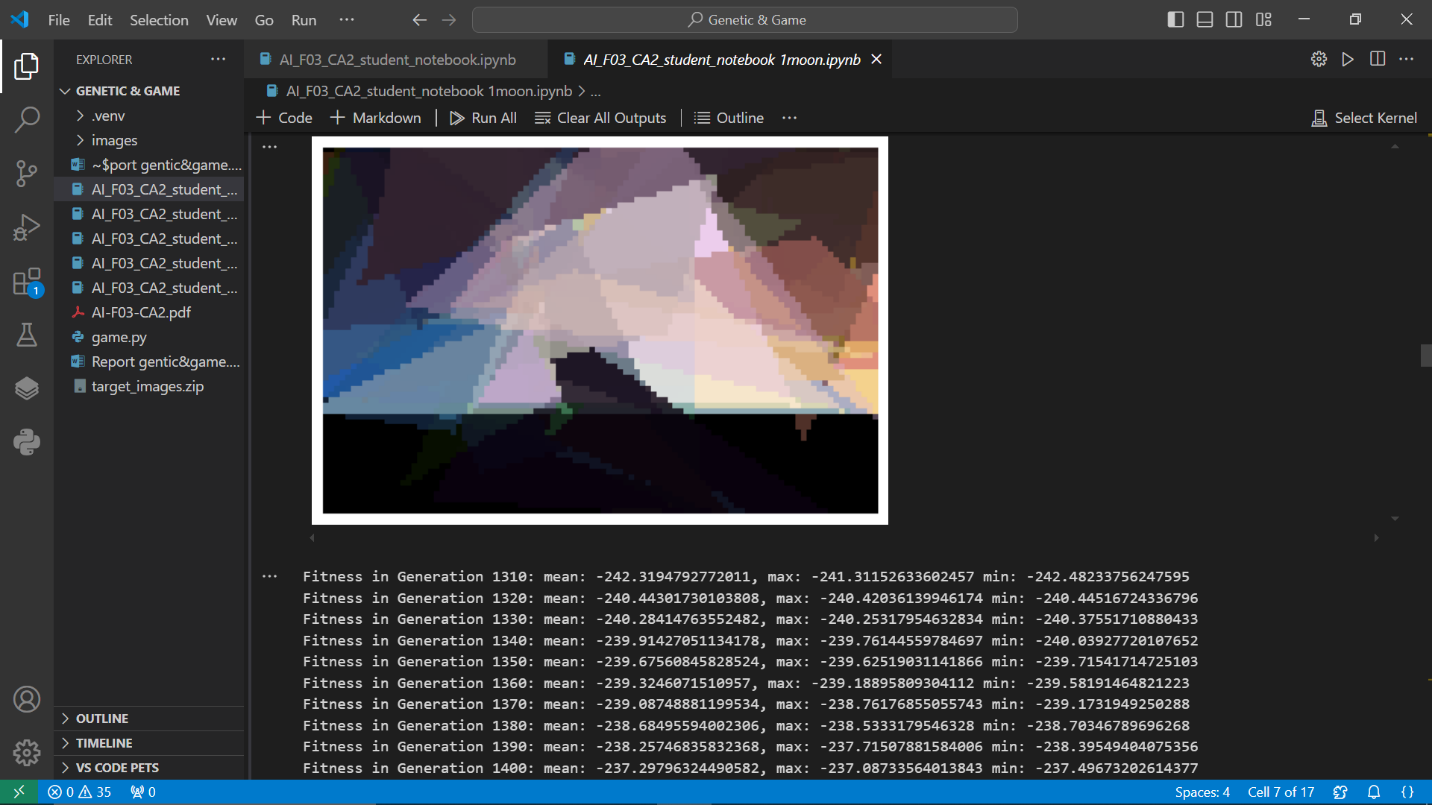
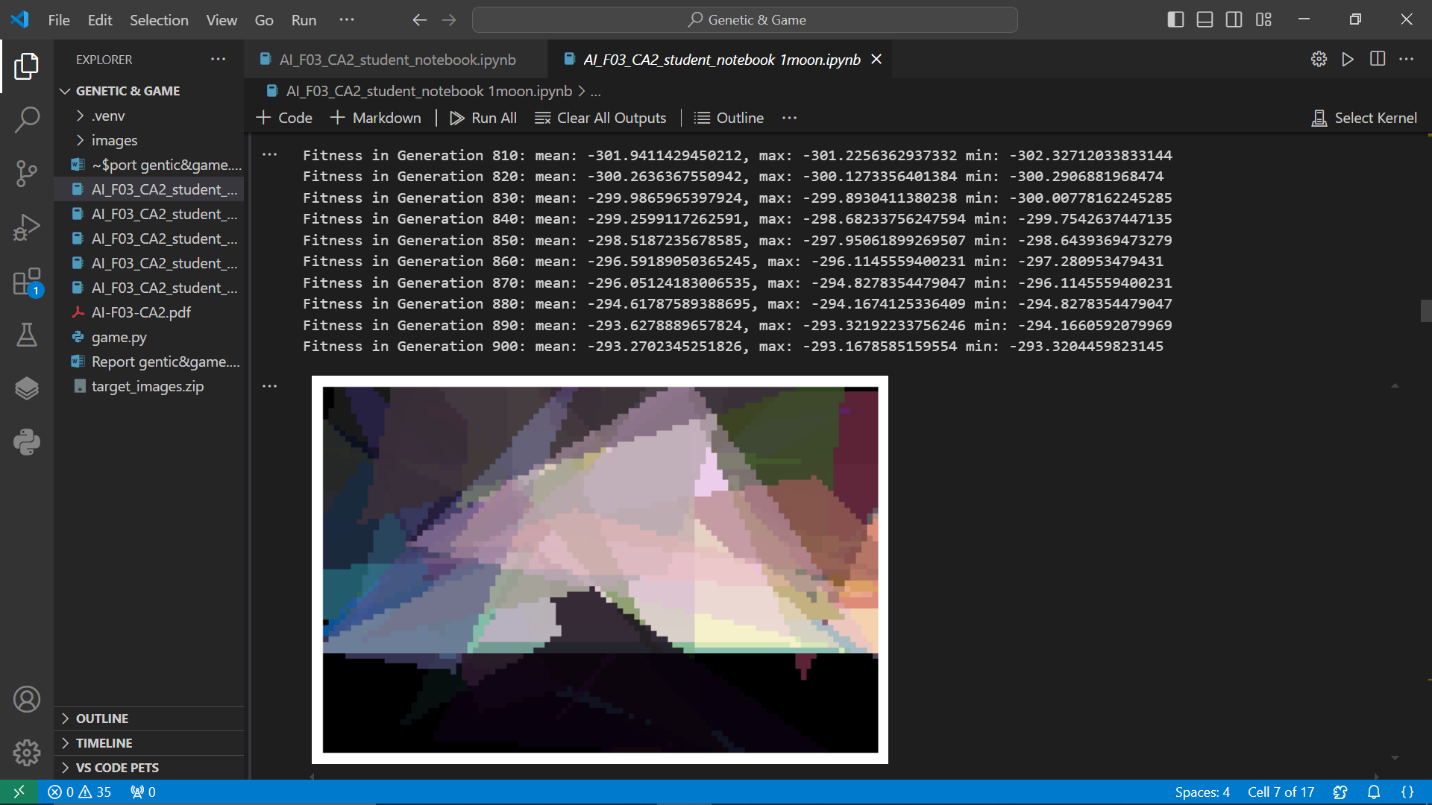
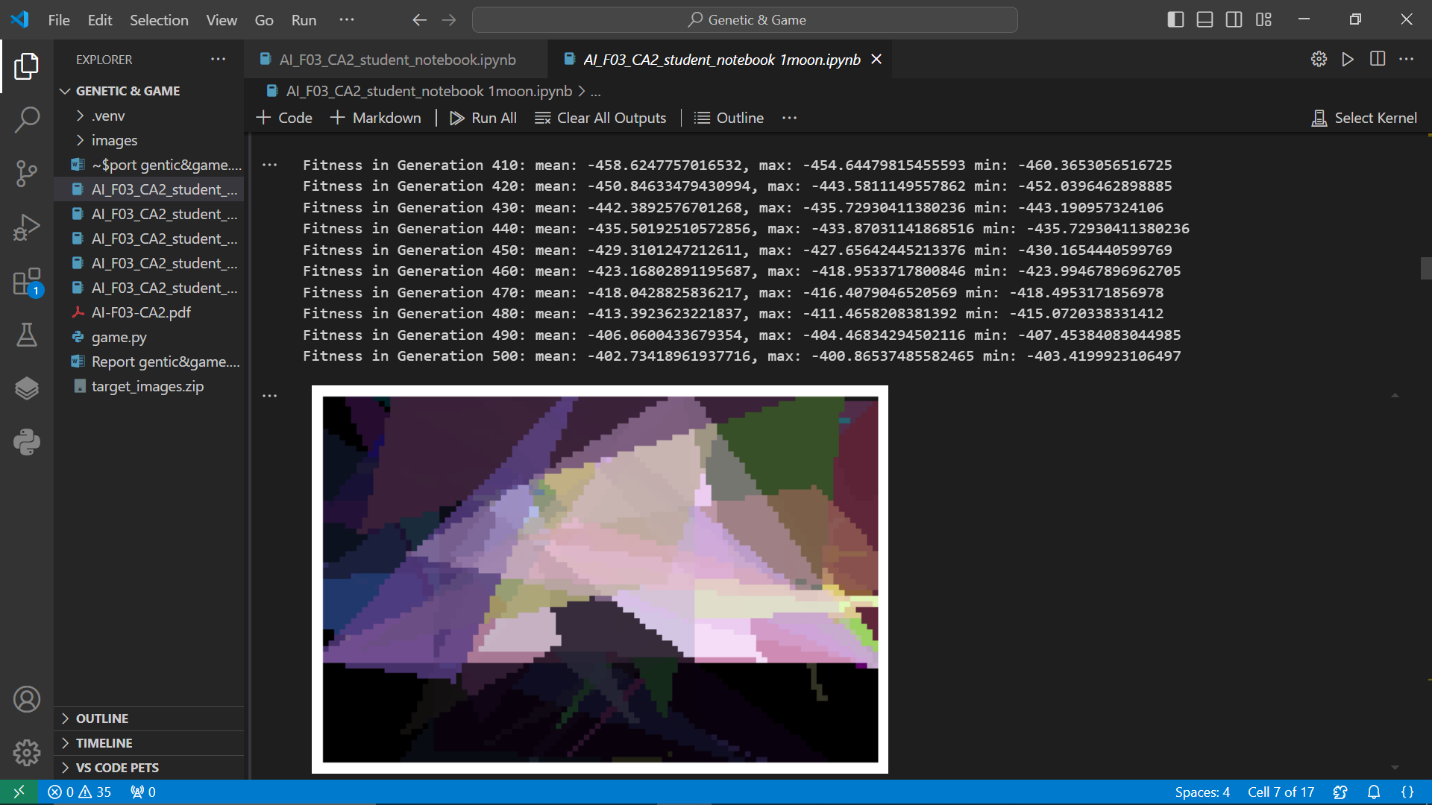
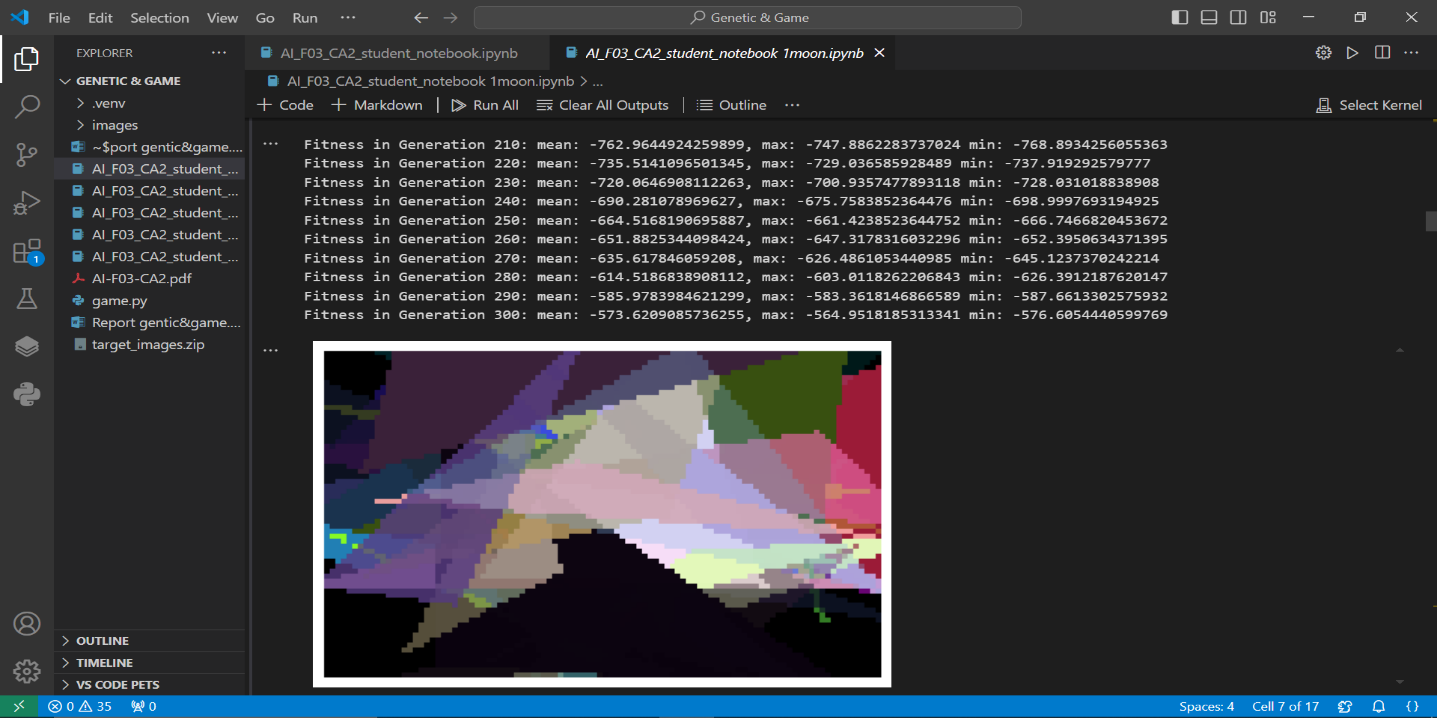
«گزارش کار پروژه دوم هوش مصنوعی»

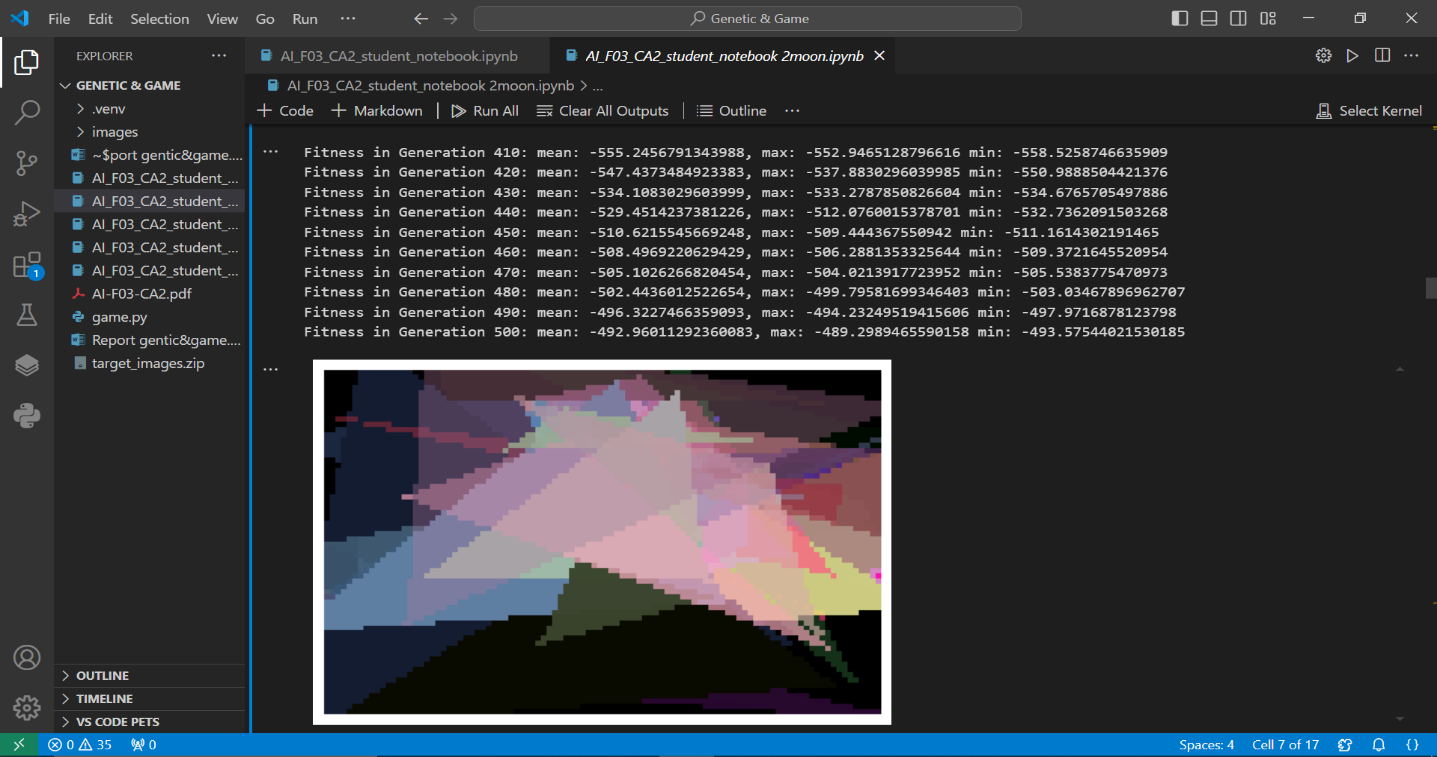
«برنا فروهری 810101480»

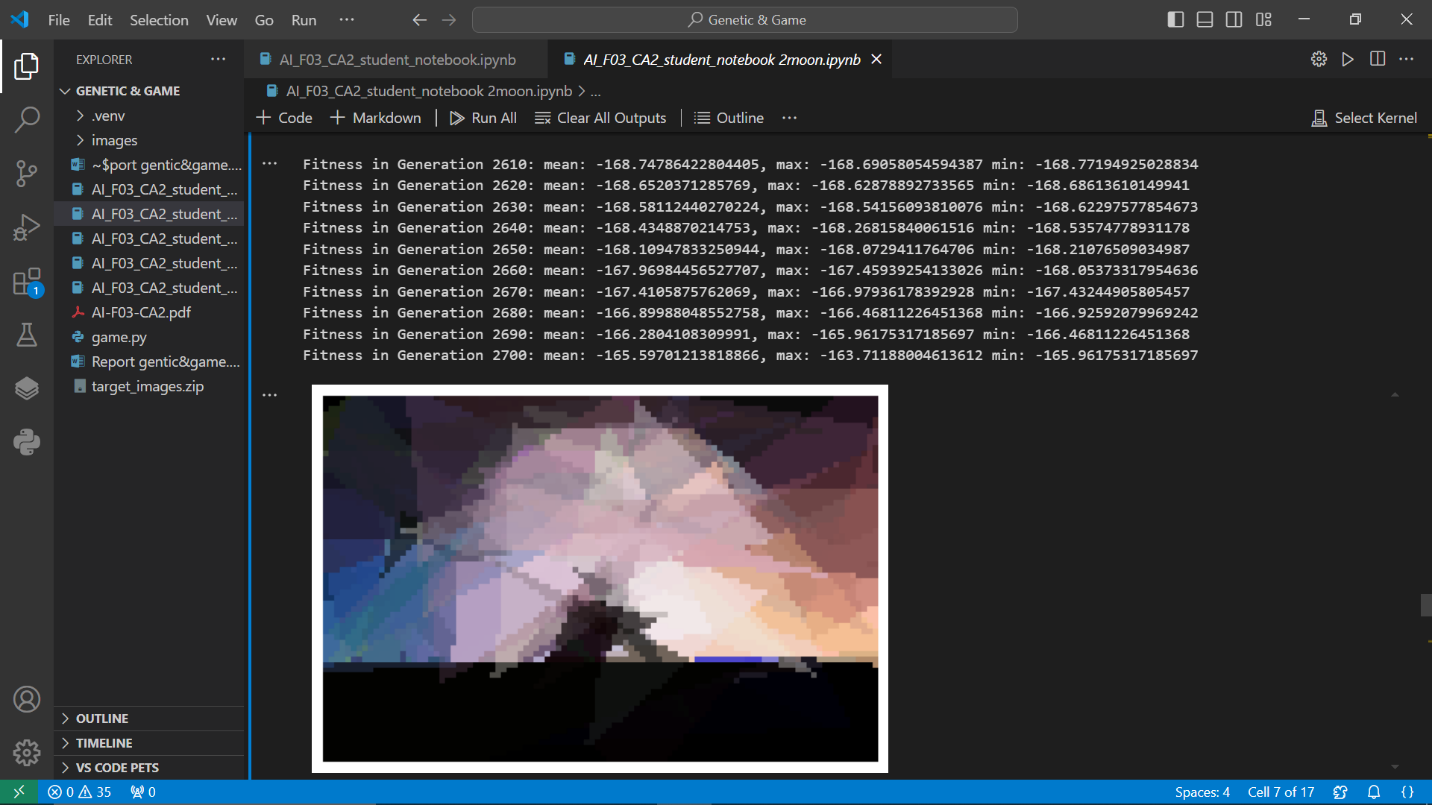
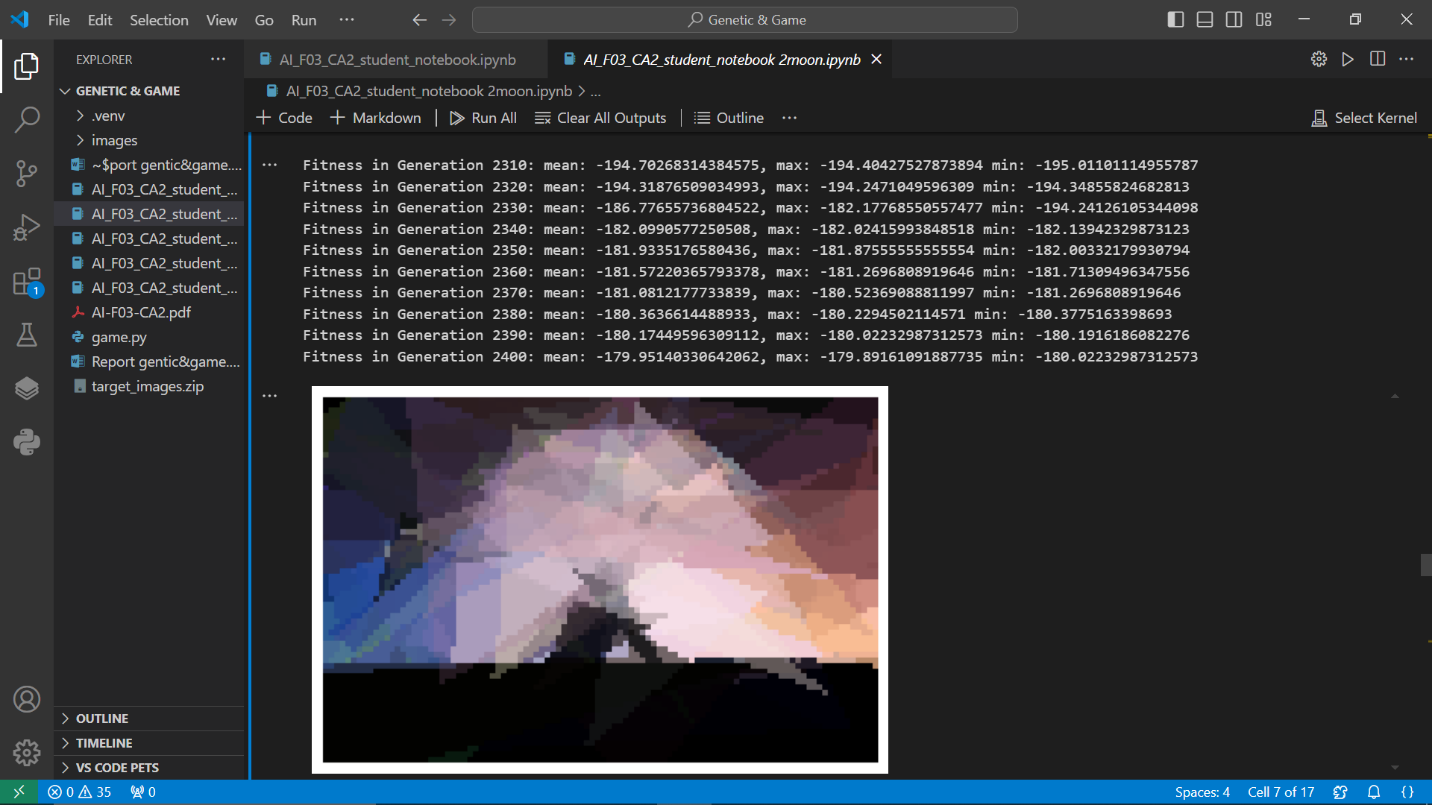
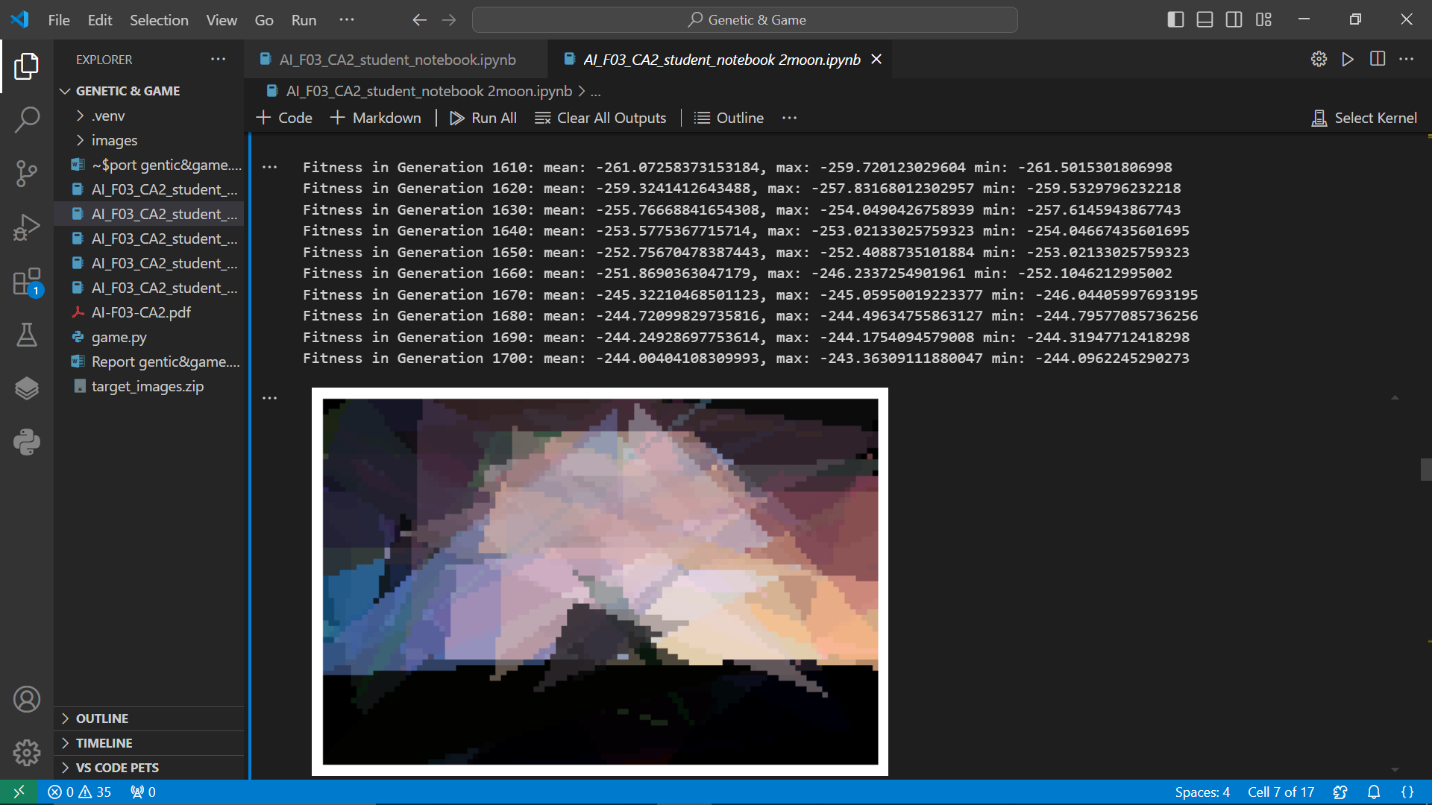
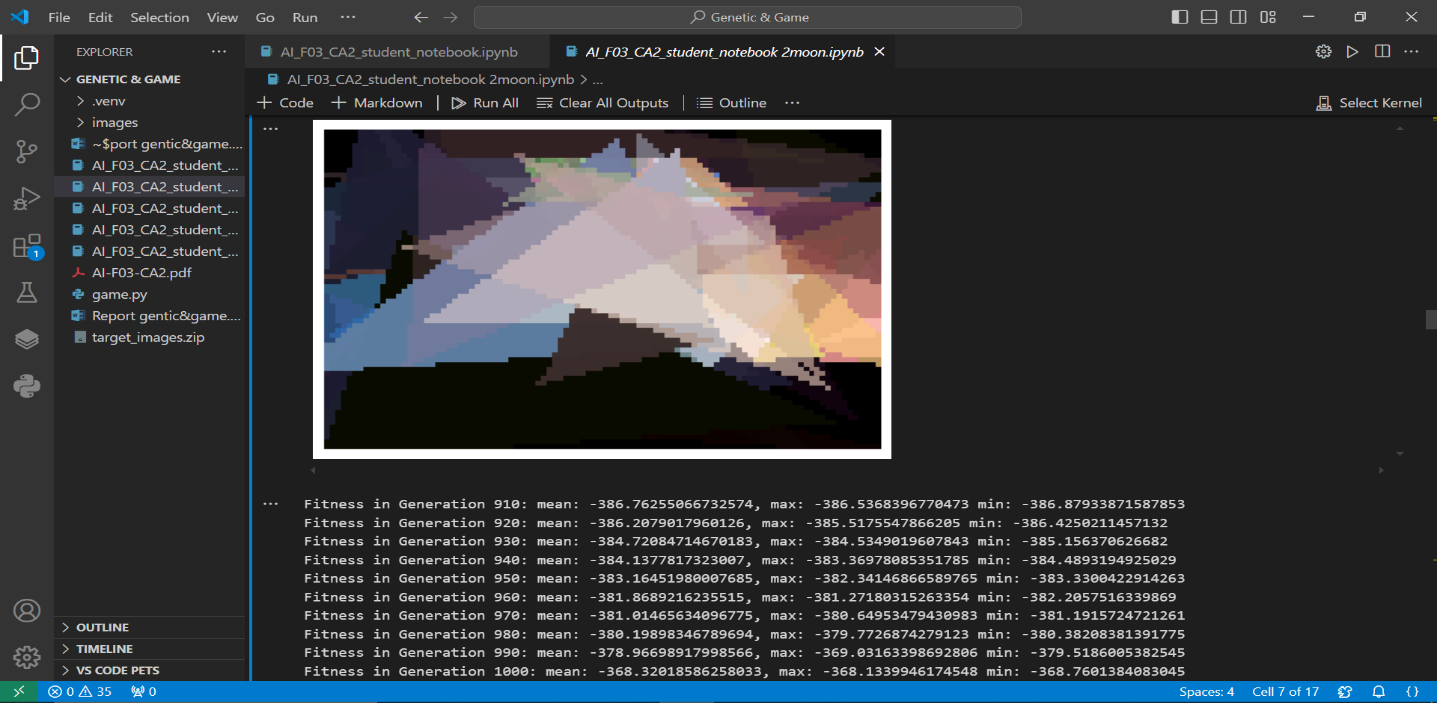
قسمت اول: Genetic

Population 50

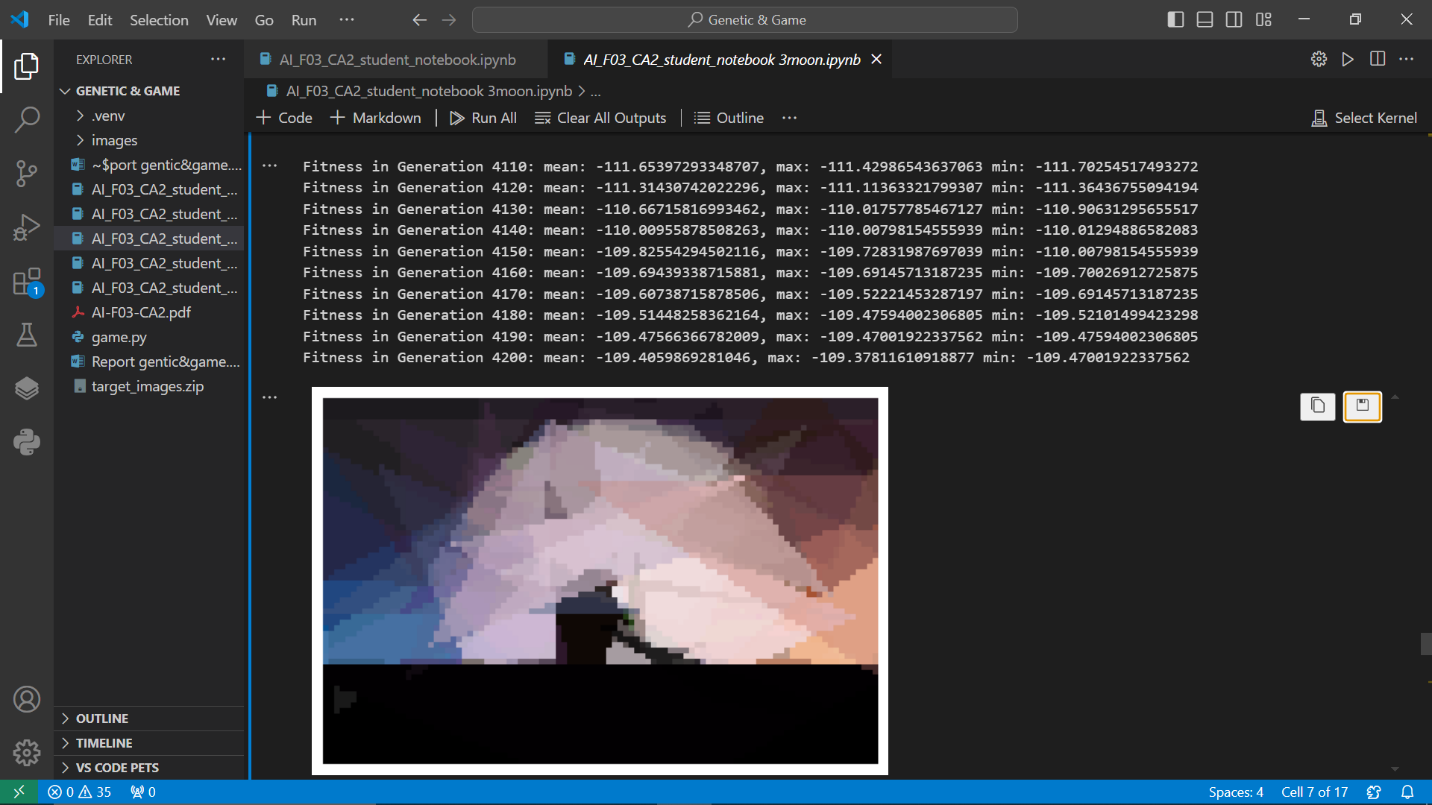
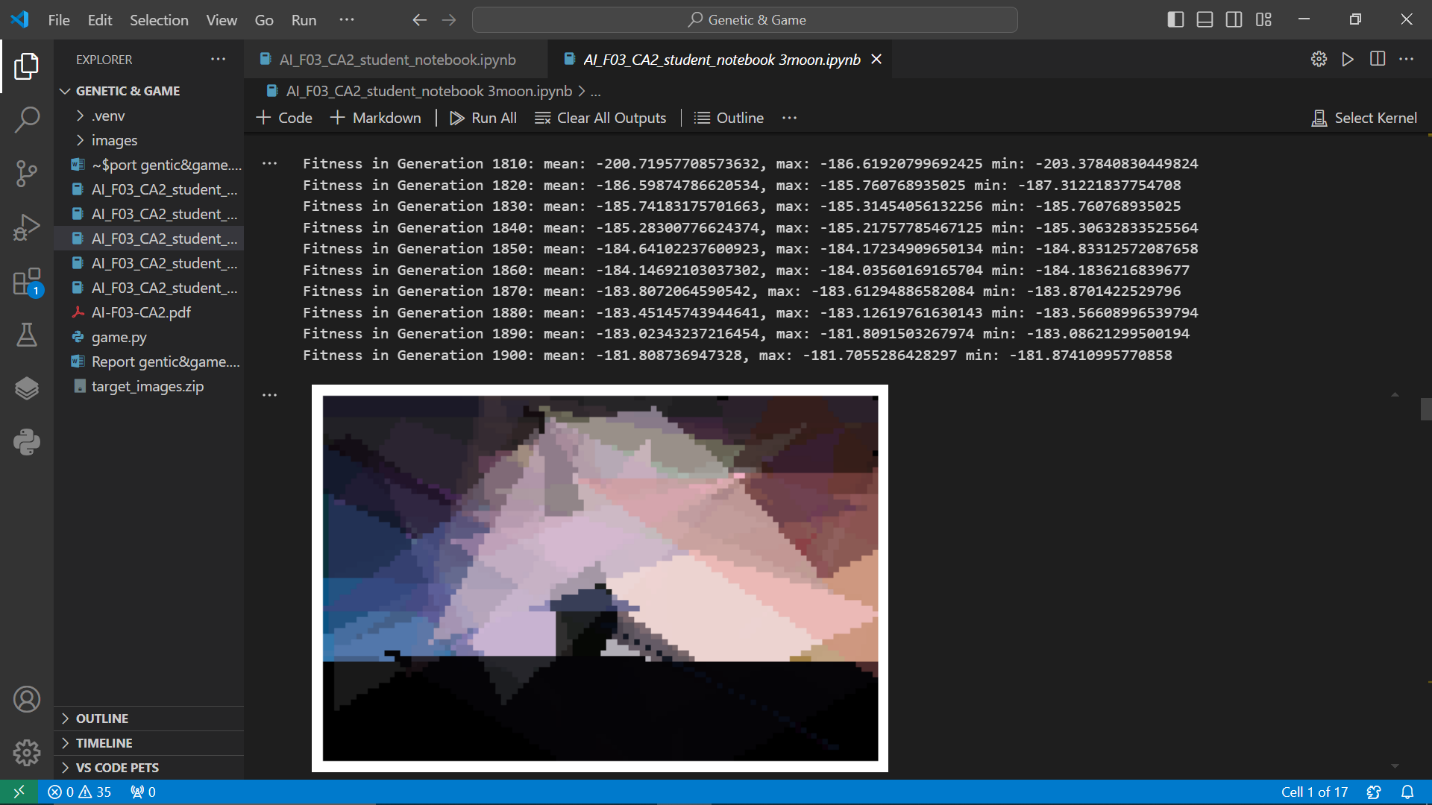
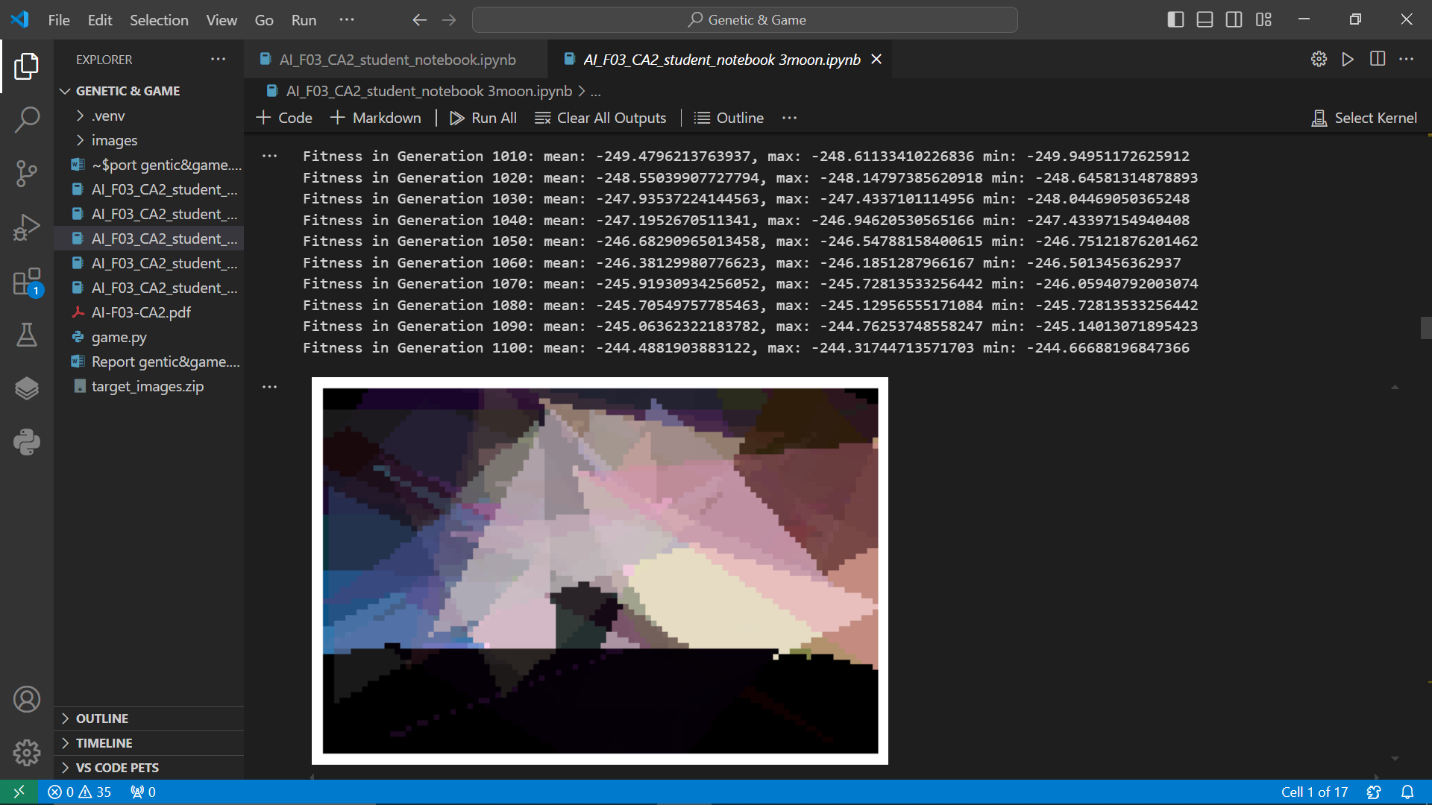
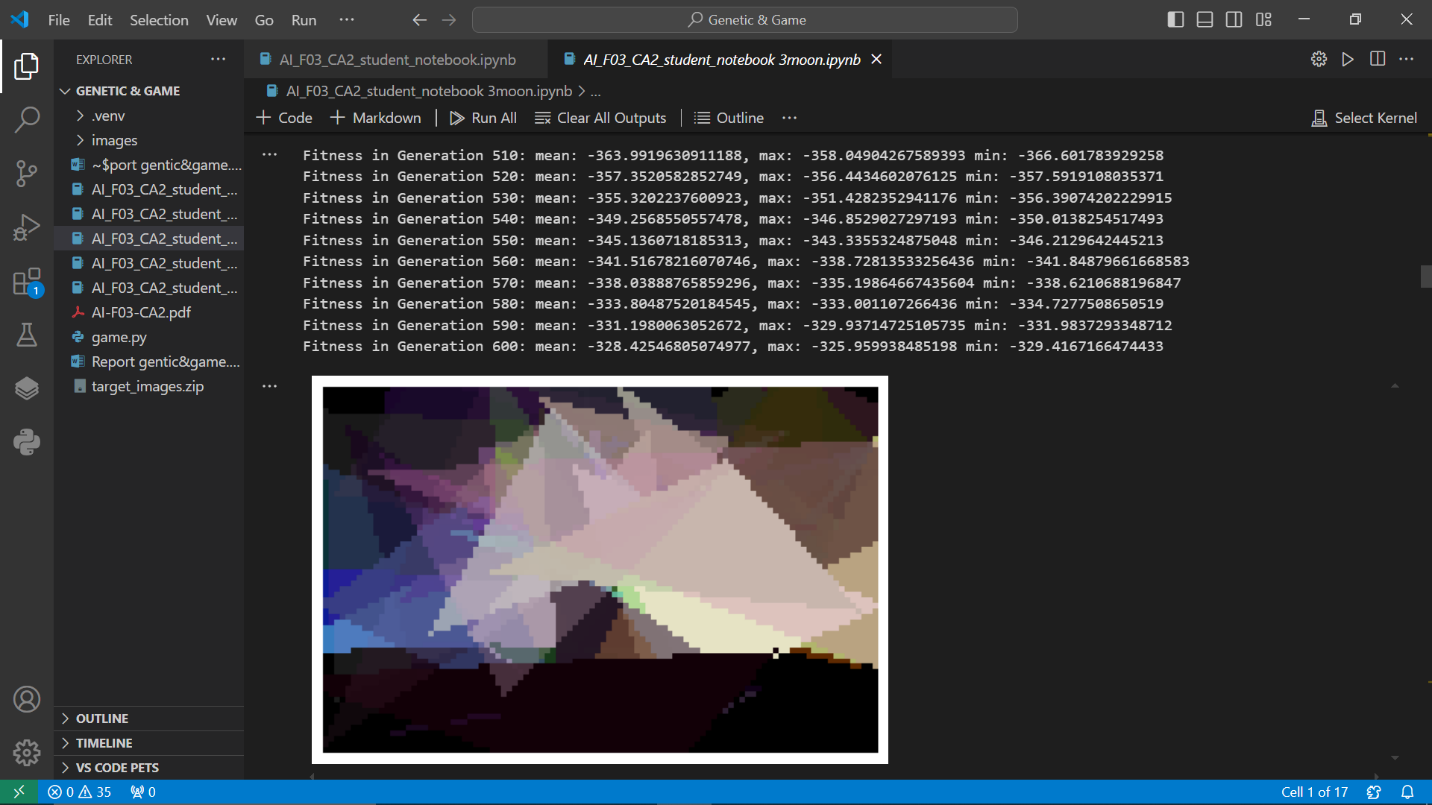
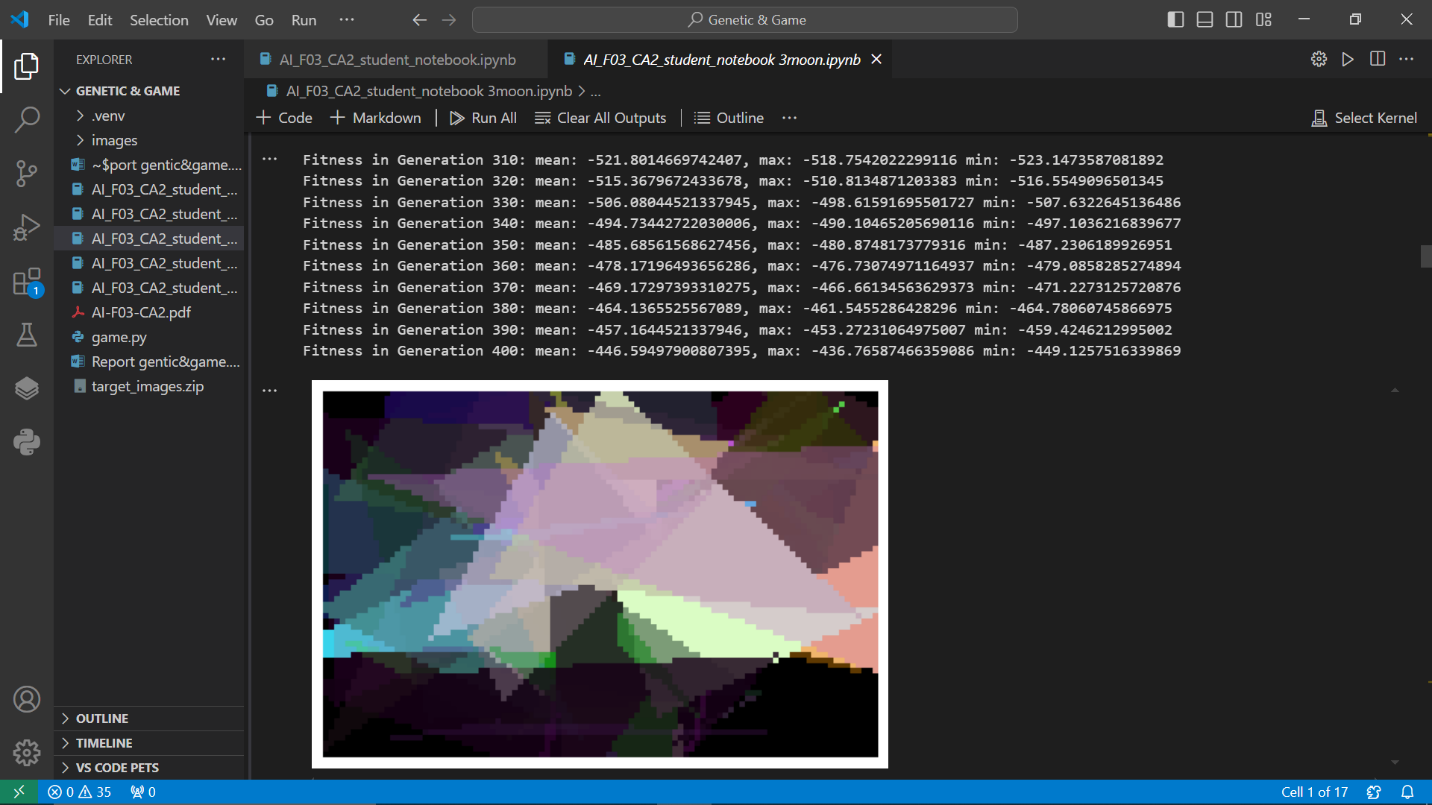


Population 70





Population 100



معیار سنجش نزدیکی عکس ساخته شده با عکس اصلی: معیار تفاوت تصاویر در این روش، از تفاوت در شدت و رنگ پیکسل‌ها بر اساس میانگین اختلافات، مجموع خطاها محاسبه شده است.

**معیارهای متداول برای سنجش تفاوت دو تصویر**

در مقایسه دو تصویر برای شبیه‌سازی یا بهبود شباهت آن‌ها به یکدیگر، می‌توان از معیارهای زیر استفاده کرد:

1. **میانگین خطای مطلق (MAE)**: محاسبه میانگین تفاوت مطلق پیکسل‌ها در تصویر هدف و تصویر تولید شده.
2. **مجموع مربع خطاها (MSE)**: محاسبه مجموع مربع تفاوت پیکسل‌ها در هر دو تصویر.
3. **PSNR (Peak Signal-to-Noise Ratio)**: مقایسه کیفیت و دقت تصویر بازسازی‌شده در مقابل تصویر اصلی. این معیار معمولاً برای ارزیابی نزدیکی در شدت پیکسل‌ها استفاده می‌شود.
4. **SSIM (Structural Similarity Index)**: مقایسه ساختارها و الگوهای موجود در دو تصویر، که به خصوص در حفظ ساختارهای محلی کاربرد دارد.

معیارهای ساده‌تر مانند MSE یا MAE که ما از MSE استفاده کردیم اغلب به دلیل سادگی و محاسبات سریع، در الگوریتم‌های ژنتیک استفاده می‌شوند، چرا که این الگوریتم‌ها نیاز به محاسبه‌های مکرر دارند. با استفاده ز این معیارها، الگوریتم ژنتیک می‌تواند با هر نسل، کروموزوم‌ها را سریع‌تر ارزیابی کند و به همین دلیل برای سرعت و کارایی بیشتر انتخاب می‌شوند.

پاسخ سوالات:

1. برای کروموزومی که در نظر گرفته اید فضای حالت آن را محاسبه کنید.

پاسخ:

برای محاسبه فضای حالت کروموزوم‌ها در یک الگوریتم ژنتیک، باید تمام ترکیبات ممکن برای کروموزوم‌ها را در نظر بگیریم. در این پروژه، کروموزوم‌ها به صورت مجموعه‌ای از مثلث‌ها تعریف شده‌اند که هر مثلث با پارامترهایی مانند نقاط (مختصات رأس‌ها)، رنگ و شفافیت مشخص می‌شود.

1. **مختصات رأس‌ها**: هر مثلث دارای ۳ رأس است و هر رأس مختصات (x, y) درون بورد تصویر دارد.
2. **رنگ و شفافیت**: رنگ هر مثلث معمولاً با سه مقدار RGB مشخص می‌شود و شفافیت معمولاً با یک مقدار بین ۰ (کاملاً شفاف) تا ۲۵۵ (کاملاً مات) مشخص می‌شود.

**محاسبه فضای حالت برای یک مثلث**

برای محاسبه فضای حالت هر مثلث، باید تعداد ترکیبات ممکن برای هر پارامتر را محاسبه کرده و آن‌ها را در هم ضرب کنیم.

**مختصات رأس‌ها**:

* + اگر عرض و ارتفاع تصویر به ترتیب width و height باشند، مختصات هر رأس می‌تواند هر مقدار صحیح از (0, 0) تا (width-1, height-1) باشد.
  + بنابراین برای هر رأس width×height ترکیب ممکن وجود دارد.
  + چون هر مثلث ۳ رأس دارد، تعداد ترکیب‌های ممکن برای مختصات‌ها برابر است با ^(width×height)3 :

**رنگ (RGB)**:

* + هر رنگ با سه مقدار (R, G, B) و هر کدام از ۰ تا ۲۵۵ تعریف می‌شود.
  + 256 به توان 3 حالت داریم

**شفافیت (Transparency)**:

* + مقدار شفافیت می‌تواند از ۰ تا ۲۵۵ باشد، یعنی ۲۵۶ مقدار ممکن دارد.

بنابراین، فضای حالت برای یک مثلث برابر است با:

(Width\*height)^3 \* 256^3 \* 256

**فضای حالت برای کل کروموزوم**

اگر فرض کنیم هر کروموزوم از n مثلث تشکیل شده باشد، تعداد ترکیب‌های ممکن برای کل کروموزوم (فضای حالت) برابر است با:

((Width\*height)^3 \* 256^3 \* 256) ^ n

1. دو تا از ایده هایی که از نظر شما می تواند باعث سریع تر همگرا شدن این مسئله شود را توضیح دهید.

پاسخ:

کروموزوم‌ها ممکن است به دلیل مشکل بهینه‌های محلی (local optima) در راه‌حل‌های کمتر بهینه گیر کنند؛ جایی که جهش‌های کوچک منجر به بهبود نمی‌شوند. اگر عملیات جهش یا ترکیب (crossover) در الگوریتم ژنتیک تنوع کافی نداشته باشد یا بیش از حد محافظه‌کارانه عمل کند، الگوریتم ممکن است زودهنگام به یک راه‌حل غیر بهینه همگرا شود. پیاده‌سازی نرخ جهش تطبیقی (adaptive mutation rates) یا استفاده از مکانیزم‌هایی مانند نخبه‌گرایی (حفظ بهترین افراد) می‌تواند به فرار از بهینه‌های محلی کمک کرده و جستجوی گسترده‌تر را تشویق کند.

نخبه‌گرایی یک روش رایج در الگوریتم‌های ژنتیک است که طی آن بهترین کروموزوم‌ها بدون تغییر به نسل بعد منتقل می‌شوند. به این ترتیب، راه‌حل‌های باکیفیت از بین نمی‌روند و تضمین می‌شود که ویژگی‌های کروموزوم‌های بهتر حفظ می‌شود.

در این پروژه، با انتقال مستقیم کروموزوم‌های با امتیاز بالا به نسل‌های بعد، الگوریتم می‌تواند با سرعت بیشتری به سمت بهترین راه‌حل‌ها حرکت کند و زمان همگرایی کاهش یابد.

یکی از مشکلات رایج در الگوریتم‌های ژنتیک این است که تنوع جمعیت با گذر زمان کاهش می‌یابد و جمعیت در نقاط بهینه محلی گیر می‌کند. برای جلوگیری از این مشکل، می‌توان از نرخ جهش تطبیقی استفاده کرد که در آن نرخ جهش بر اساس وضعیت جمعیت یا نسل تغییر می‌کند.

* **نحوه عملکرد**:
  + در ابتدای الگوریتم، نرخ جهش بالا در نظر گرفته می‌شود تا تنوع بیشتری ایجاد شود و الگوریتم بتواند فضای حالت را بهتر کاوش کند.
  + با پیشرفت نسل‌ها و همگرا شدن الگوریتم به سمت یک راه‌حل مناسب، نرخ جهش کاهش می‌یابد تا کروموزوم‌ها حول راه‌حل‌های بهینه باقی بمانند.

1. استراتژی های متفاوتی برای انتخاب نسل بعد در الگوریتم های ژنتیک وجود دارد. درباره دو مورد از آن ها توضیح دهید.

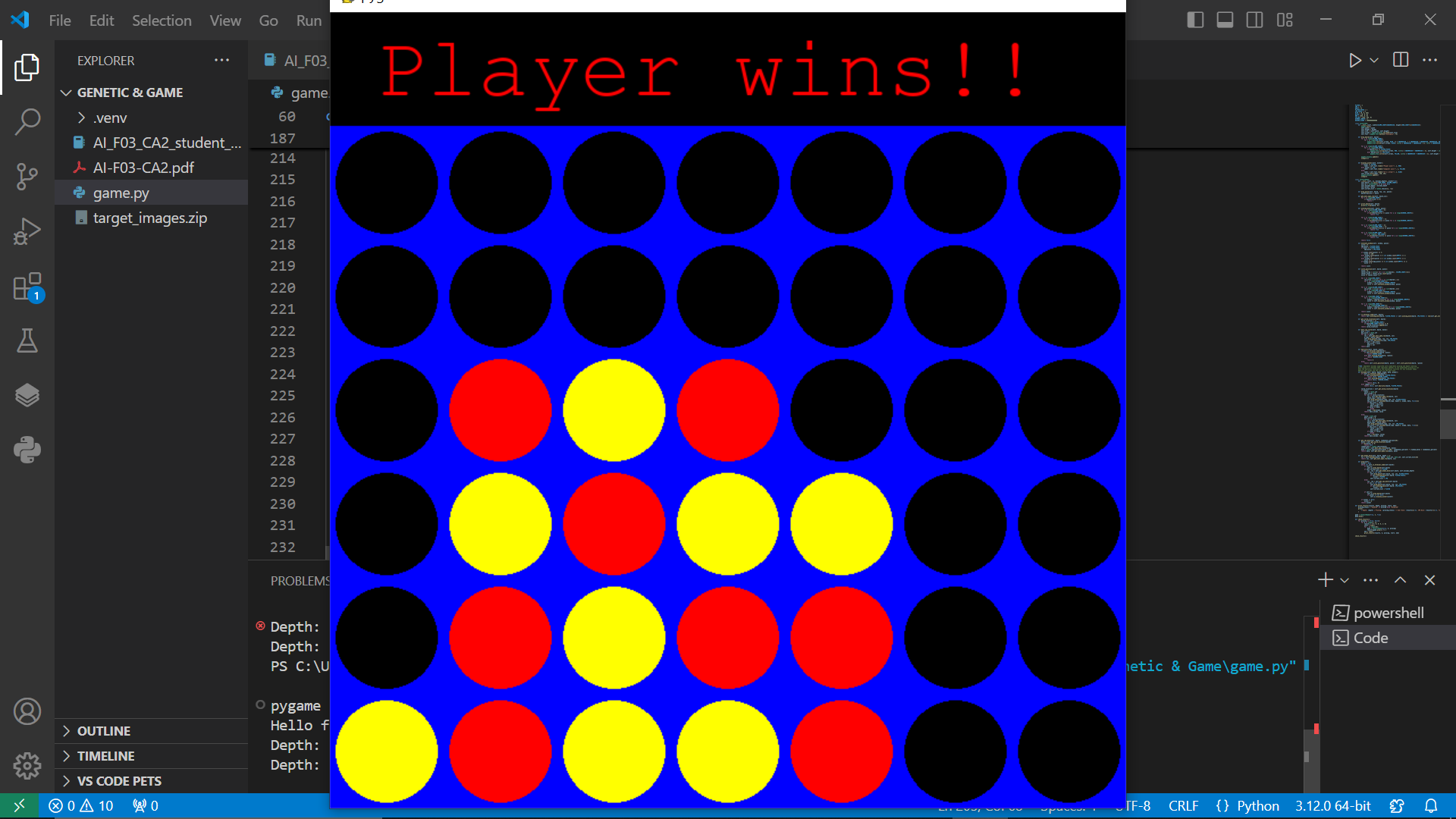
پاسخ:

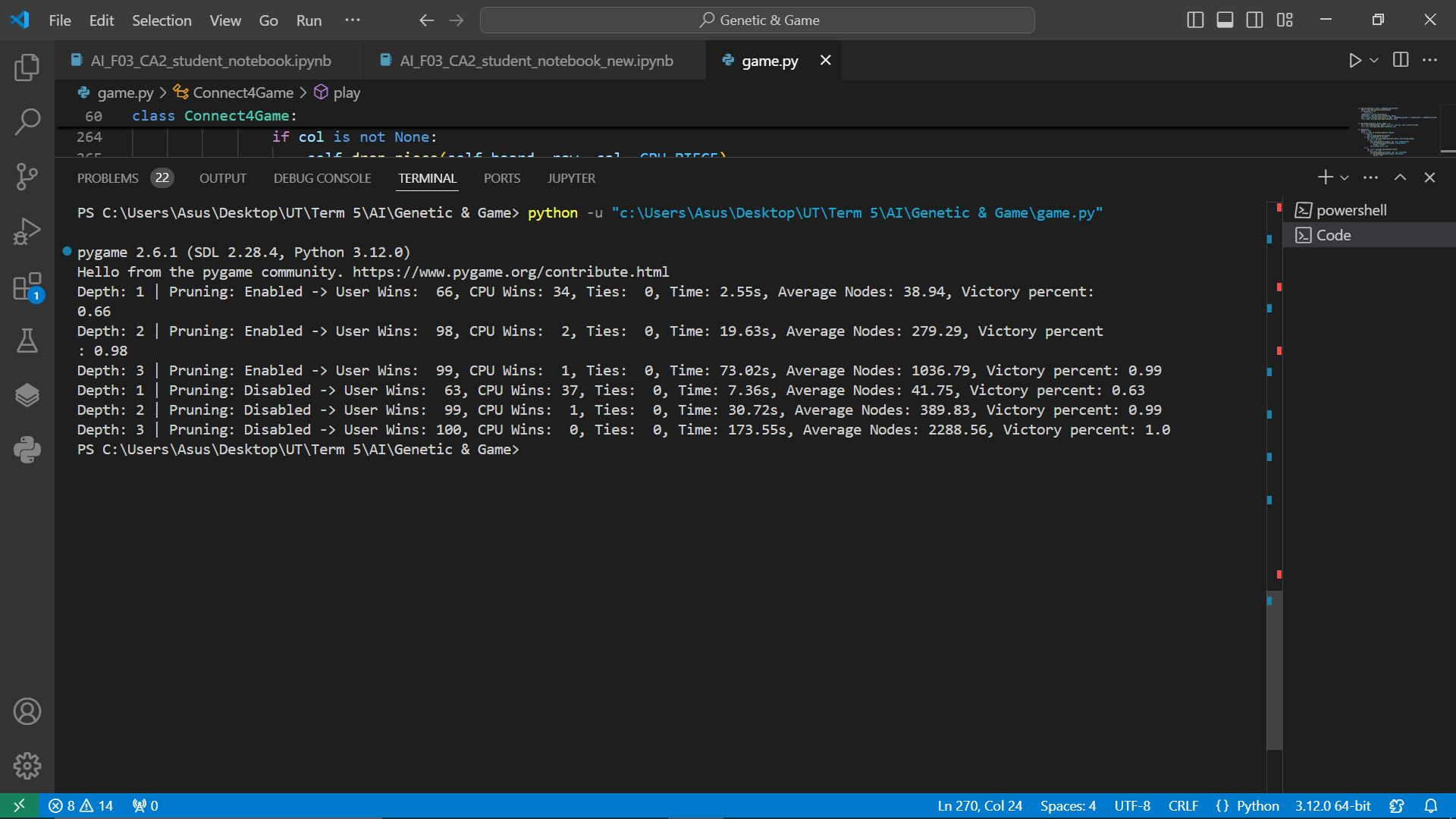
استراتژی‌های مختلف برای انتخاب شامل موارد زیر هستند:

* **انتخاب چرخ رولت (Roulette Wheel Selection)**: انتخاب احتمالی بر اساس نسبت شایستگی؛ کروموزوم‌هایی با شایستگی بالاتر شانس بیشتری برای انتخاب دارند. این روش ممکن است به همگرایی سریع منجر شود، اما خطر از دست رفتن تنوع را نیز دارد.
* **انتخاب تورنمنت (Tournament Selection)**: گروهی از کروموزوم‌ها با هم رقابت می‌کنند و کروموزومی که شایستگی بهتری دارد انتخاب می‌شود. این روش فشار انتخاب را با حفظ تنوع بهتر از انتخاب چرخ رولت متعادل می‌کند.
* **انتخاب مبتنی بر رتبه (Rank-Based Selection)**: کروموزوم‌ها بر اساس شایستگی رتبه‌بندی می‌شوند و انتخاب بر اساس رتبه آن‌ها (نه شایستگی خام) صورت می‌گیرد. این روش فشار انتخاب را کنترل می‌کند و از تسلط افراد با شایستگی بسیار بالا جلوگیری می‌کند.

هر روش، توازن میان حفظ تنوع و فشار انتخاب را به شکل متفاوتی برقرار می‌کند. برای وظایفی که نیاز به کاوش بیشتری دارند، انتخاب مبتنی بر رتبه یا انتخاب تورنمنت ترجیح داده می‌شود تا از همگرایی زودرس جلوگیری شود.

قسمت دوم: Game





تعریف شانس پیروزی:

تعداد پیروزی هایuser تقسیم بر تعداد کل بازی ها (در اینجا 100)

پاسخ سوالات:

1. آیا میان عمق الگوریتم و پارامترهای حساب شده در بخش باال روابطی میبینید؟ بررسی کنید که عمق الگوریتم چه تاثیراتی بر روی شانس پیروزی، زمان و گرههای دیده شده میگذارد.

پاسخ:

**عمق الگوریتم (Depth)**:

* **شانس پیروزی بازیکن**: با افزایش عمق، شانس پیروزی بازیکن افزایش می‌یابد، زیرا عمق بیشتر به الگوریتم اجازه می‌دهد که حالات بیشتری از بازی را بررسی و تحلیل کند. بنابراین، تصمیم‌گیری‌های بهتری برای بازیکن انجام می‌شود.
* **تعداد گره‌های مشاهده‌شده**: افزایش عمق، تعداد گره‌های بررسی‌شده را به‌شدت افزایش می‌دهد، زیرا هر چه عمق بیشتر باشد، فضای جستجوی بزرگتری مورد بررسی قرار می‌گیرد.
* **زمان اجرا**: با افزایش عمق، زمان اجرای الگوریتم بیشتر می‌شود. این افزایش به دلیل تعداد بیشتر گره‌هایی است که باید بررسی شوند.

**alpha-beta pruning**:

* + **تاثیر بر زمان و گره‌های بررسی‌شده**: فعال کردن alpha-beta pruning باعث کاهش قابل توجه در تعداد گره‌های مشاهده‌شده و در نتیجه کاهش زمان اجرا می‌شود. این ویژگی از محاسبه حالات غیرضروری جلوگیری می‌کند.
  + **شانس پیروزی بازیکن**: تاثیر کمی بر شانس پیروزی بازیکن دارد، اما بازیکن با فعال بودن pruning و در عمق‌های بیشتر احتمال بیشتری برای برد دارد، زیرا pruning به الگوریتم امکان جستجوی عمیق‌تر با همان محدودیت زمانی را می‌دهد.

با ترکیب depth و pruning شانس بیشتری برای پیروزی داریم و الگوریتم با بهینه‌سازی در زمان می‌تواند عملکرد بهتری داشته باشد.

1. آیا میتوان ترتیب دیدن فرزندان هر نود را به گونهای انتخاب کنیم که بیشترین هرس را داشته باشیم؟ اگر جواب شما مثبت است روش خود را توضیح دهید و در غیر اینصورت توضیح دهید که چرا این عمل امکان پذیر نیست

پاسخ:

بله، می‌توان ترتیب بررسی فرزندان هر نود را طوری تنظیم کرد که بیشترین هرس (pruning) در الگوریتم minimax با alpha-beta pruning انجام شود. این عمل به کاهش تعداد گره‌های بررسی‌شده و در نتیجه کاهش زمان اجرای الگوریتم کمک می‌کند.

**روش برای بهینه‌سازی هرس:**

برای دستیابی به بیشترین میزان هرس، باید فرزندان هر نود را به ترتیبی بررسی کنیم که احتمال یافتن بهترین حرکت زودتر وجود داشته باشد. این کار به ما کمک می‌کند تا بازه‌های alpha و beta سریع‌تر محدود شوند و شاخه‌های بیشتری هرس شوند.

به عنوان مثال، برای حالت بازیکن

اگر بتوانیم فرزندانی که احتمالاً امتیاز بیشتری دارند را زودتر بررسی کنیم، احتمال به‌روزرسانی مقدار alpha در سریع‌ترین زمان ممکن افزایش می‌یابد، و شاخه‌های بعدی که امتیازشان کمتر از alpha باشد، سریع‌تر هرس می‌شوند.

**پیاده‌سازی این روش:**

**ترتیب‌دهی فرزندان براساس امتیاز احتمالی**: قبل از شروع جستجو در هر نود، فرزندان نود را بر اساس تابع heuristic یا امتیاز تخمینی آن‌ها مرتب کنید. فرزندانی که امتیاز بالاتری دارند را در حالت maximizing و فرزندانی که امتیاز پایین‌تری دارند را در حالت minimizing ابتدا بررسی کنید.

**استفاده از نتایج قبلی**: اگر از minimax در یک بازی با جستجوهای مکرر استفاده می‌شود، می‌توان از نتایج جستجوهای قبلی برای تعیین ترتیب فرزندان در جستجوهای بعدی استفاده کرد. برای مثال، حرکت‌های قبلی که به بهترین نتایج منجر شده‌اند، اولویت بیشتری در جستجو دارند.

1. Branching factorرا توضیح دهید و بگویید که با پیشرفت این بازی چه تغییراتی میکند؟

پاسخ:

**Bramching factor یا ضریب انشعاب** به میانگین تعداد حرکت‌های ممکن در هر نود از درخت جستجو گفته می‌شود. به عبارت دیگر، ضریب انشعاب نشان می‌دهد که در هر مرحله از بازی، بازیکن به طور متوسط چند گزینه‌ی مختلف برای حرکت دارد.

هر چه ضریب انشعاب بیشتر باشد، فضای جستجوی بزرگتری ایجاد می‌شود که حل مسئله را پیچیده‌تر و زمان‌برتر می‌کند.

**ضریب انشعاب در بازی Connect 4**

در Connect 4، ضریب انشعاب به تعداد ستون‌های خالی بورد بستگی دارد. بازی با یک بورد خالی شروع می‌شود و در هر مرحله بازیکن می‌تواند قطعه خود را در یکی از ستون‌های خالی قرار دهد.

1. **ابتدای بازی**:
   * در ابتدای بازی، تمام ستون‌ها خالی هستند، بنابراین ضریب انشعاب در این حالت حداکثر است. برای یک بورد استاندارد با ۷ ستون، ضریب انشعاب در اولین حرکت برابر با ۷ است.
2. **میانه‌ی بازی**:
   * با گذشت بازی و پر شدن بعضی از ستون‌ها، تعداد حرکت‌های ممکن کاهش می‌یابد. در این مرحله، ضریب انشعاب به تدریج کمتر می‌شود زیرا برخی ستون‌ها پر شده‌اند و دیگر امکان انتخاب آن‌ها وجود ندارد.
3. **پایان بازی**:
   * در نزدیکی پایان بازی، بیشتر ستون‌ها پر شده‌اند و گزینه‌های کمتری برای حرکت باقی می‌ماند. در این حالت، ضریب انشعاب به حداقل مقدار خود می‌رسد.
4. **توضیح دهید که چرا به هنگام هرس کردن الگوریتم بدون از دست دادن دقت خود سریعتر میشود.**

پاسخ:

هنگام استفاده از alpha-beta pruning در الگوریتم minimax، الگوریتم با حذف شاخه‌های غیرضروری در درخت جستجو بدون بررسی کامل آن‌ها، به راه‌حل بهینه می‌رسد. این کار باعث می‌شود که الگوریتم بدون از دست دادن دقت، سریع‌تر عمل کند.

**کاهش تعداد گره‌های بررسی‌شده**:

با هرس کردن، الگوریتم از بررسی شاخه‌هایی که نتیجه نهایی را تغییر نمی‌دهند، صرف‌نظر می‌کند. در نتیجه، تعداد گره‌های بررسی‌شده به شدت کاهش می‌یابد. این به کاهش زمان اجرای الگوریتم کمک می‌کند، چرا که برخی محاسبات که نتیجه را تغییر نمی‌دهند، نادیده گرفته می‌شوند.

**حفظ دقت با حفظ نتیجه بهینه**:

هرس کردن هیچ ‌گاه شاخه‌هایی که می‌توانند بهترین امتیاز را تغییر دهند، حذف نمی‌کند. در واقع، alpha-beta pruning تنها زمانی شاخه‌ها را هرس می‌کند که مطمئن باشد آن شاخه نتیجه‌ای کمتر از alpha یا بیشتر از beta خواهد داشت و این نتیجه در تصمیم نهایی تأثیر نخواهد گذاشت. بنابراین، هرس کردن دقت را از بین نمی‌برد.

**کاهش پیچیدگی زمانی مؤثر**:

بدون alpha-beta pruning، پیچیدگی زمانی الگوریتم minimax به صورت نمایی افزایش می‌یابد. اما با استفاده از alpha-beta pruning، اگر ترتیب بررسی بهینه باشد، پیچیدگی مؤثر الگوریتم به حدود نصف کاهش می‌یابد. این امر باعث می‌شود الگوریتم در زمان کوتاه‌تری به همان نتیجه دقیق برسد.

1. چرا در حاالتی که حریف به صورت شانسی عمل میکند (مانند این پروژه)، استفاده از minimax بهینه ترین روش نیست؟ چه الگوریتمی میتواند جایگزین این الگوریتم باشد؟ توضیح دهید.

پاسخ:

**الگوی تصادفی حریف**: الگوریتم minimax بر اساس این فرض عمل می‌کند که حریف بهترین حرکات را برای به حداقل رساندن امتیاز بازیکن انجام می‌دهد. اما اگر حریف به طور تصادفی حرکت کند، دیگر فرض "حداکثر تلاش برای بهینه بودن" حریف وجود ندارد و حرکات حریف قابل پیش‌بینی نیست. در این صورت، الگوریتم minimax به طور کامل پتانسیل خود را از دست می‌دهد، زیرا الگوریتم برای حرکات حریف، بر اساس امتیازدهی دقیق طراحی شده و رفتار حریف را به درستی نمی‌تواند پیش‌بینی کند.

**تخصیص منابع محاسباتی نامتناسب**: الگوریتم minimax، به خصوص با عمق‌های بالاتر، نیازمند منابع محاسباتی زیادی است، زیرا تعداد حالات ممکن به سرعت افزایش می‌یابد. این پیچیدگی محاسباتی برای مقابله با یک حریف تصادفی که حرکت‌های بهینه انجام نمی‌دهد، مقرون به صرفه نیست.

**الگوریتم جایگزین: Monte Carlo Tree Search (MCTS)**

الگوریتم **جستجوی درختی مونت کارلو (MCTS)** یک جایگزین مناسب و مؤثر برای بازی در برابر حریف تصادفی است.

**نحوه کار MCTS**

MCTS از شبیه ‌سازی‌های تصادفی برای ارزیابی حرکات مختلف استفاده می‌کند. این روش به چهار مرحله تقسیم می‌شود:

1. **انتخاب (Selection)**: انتخاب گره‌هایی از درخت که ارزش بررسی بیشتری دارند، تا زمانی که به گره ای برسیم که باید توسعه داده شود.
2. **توسعه (Expansion)**: گسترش گره‌های جدید که فرزند گره‌های انتخاب‌شده‌اند.
3. **شبیه‌سازی (Simulation)**: اجرای بازی از گره توسعه یافته تا رسیدن به پایان بازی، به صورت تصادفی.
4. **بازگشت و به‌روزرسانی (Backpropagation)**: بازگشت به گره‌های بالا و به‌روزرسانی اطلاعات مربوط به بردها و باخت‌ها در هر گره، بر اساس نتایج شبیه‌سازی‌ها.

**مزایای MCTS در برابر حریف تصادفی**

1. **تناسب با الگوی تصادفی حریف**: MCTS به جای تلاش برای بهینه‌سازی حرکات حریف، حرکات مختلف را با شبیه‌سازی‌های تصادفی بررسی می‌کند. این روش برای بازی در برابر حریف تصادفی بسیار مناسب است، زیرا از شبیه‌سازی‌های متنوع استفاده می‌کند و رفتار حریف را بر اساس الگوهای تصادفی ارزیابی می‌کند.
2. **انعطاف‌پذیری و تطبیق‌پذیری**: MCTS با توجه به شبیه‌سازی‌های متعددی که انجام می‌دهد، انعطاف بیشتری در انتخاب حرکات دارد. به جای تکیه بر امتیازدهی دقیق، از تعداد زیادی شبیه‌سازی برای تعیین حرکت بهینه استفاده می‌کند. این امر به الگوریتم اجازه می‌دهد تا به طور مداوم با شرایط غیرقابل پیش‌بینی سازگار شود.
3. **بهبود تدریجی با افزایش شبیه‌سازی‌ها**: در MCTS، هر چه تعداد شبیه‌سازی‌ها بیشتر باشد، دقت ارزیابی‌ها بهتر می‌شود. این ویژگی به الگوریتم اجازه می‌دهد که حتی با محدودیت‌های زمانی و محاسباتی، حرکات بهینه را پیدا کند و به تدریج کیفیت تصمیمات را افزایش دهد.