**“Mini-flex” Breakout Agent**

**Overview**

The primary goal of my project was to construct an agent that combines elements of reflex agents and minimax agents in the hopes of producing something that resembles human behavior when playing a game, and to compare the performance of this agent with its constituent agents in their pure forms. I chose Breakout because it was one of the only Atari games I actually understood the rules for, and because I thought it provided a straightforward basis for bisecting the algorithm. After days of testing my limits and doing my best to get a hang of the Open AI Atari environment, I decided my agent would react to the ball when it is moving downward towards the paddle, and as this is happening the agent will run a minimax algorithm to determine which direction to move the paddle and prevent loss of life. It wasn’t too much trouble coding a reflex agent, but coding a useful minimax agent proved to be impossible (for me anyway). I hoped that bridging the gap would solve the problem, but it did not. As a result, my objectives for the project changed a lot as it went on, and I ended up learning about different things than I initially sought to learn.

**What I Wanted to Learn**: Does combining the reactive element of reflex agency and the predictive element of minimax agency produce a human like agent? Does it produce a high-performance agent? How does the mini-flex agent compare to minimax and reflex agents? Do the minimax or miniflex agents offer any utility in an environment that can clearly be dominated by a sufficiently sophisticated reflex agent?

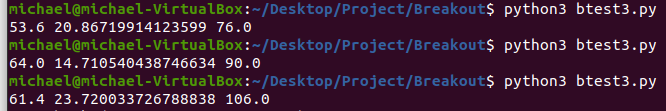
As we will see I didn’t end up answering most of these questions, but I answered some other higher resolution questions in the process. The last may have been the most important, as it speaks to the overall value of the entire project.

**Methodology:**

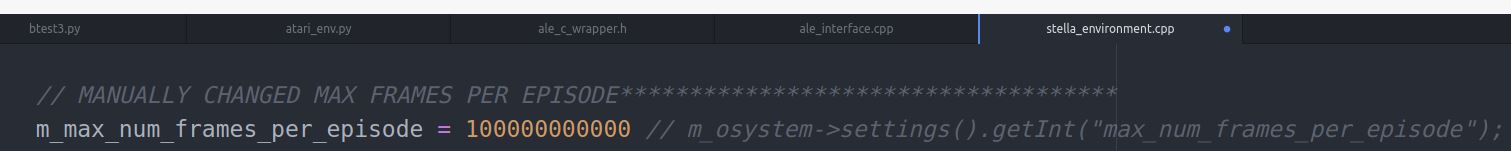
Reflex Agent: The reflex agent is a fairly unsophisticated and generic one, but the results show that it performs better than I do, and since I was going for a comparison to human behavior and techniques, I am happy with its output. Here is a video of it reaching 99 points, a score pretty high above anything I have scored.

<https://www.youtube.com/watch?v=ODKCEj-cVPA>

Here are the results from 3 test runs of the current version of the agent. Average score, standard deviation, and max score, respectively:

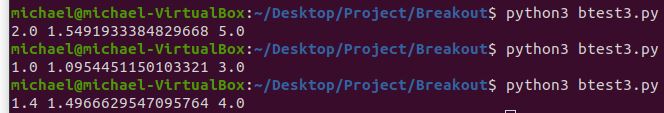
 Its average score was around a 55, which is better than I can do, and its high score was a 175 (in another trial), which I can’t even come close to. So, I left it at that and planned on using it as a benchmark. But as they say, “man makes plans and God laughs”…

Minimax Agent: I did my best to model my minimax algorithm by my submission for the Pacman assignment. After proofreading it countless times and looking for errors, I began to realize that it simply runs too slowly to be of any use. Even when I limit the depth of the minimax tree to 2 levels, the game takes place at about 2 frames per second, and the construction of a tree is completely unnecessary if it only has 2 levels. On top of this, I first encountered an issue where the game would cease at an earlier point the greater the depth of the minimax tree. To solve this was a journey not for the faint of heart. The step() method returns the value for is\_done, which I believed was being returned as True too early. From here I opened atari\_env.py to find the step() method, which called game\_over() to retrieve the terminal status. So, I opened ale\_c\_wrapper.h, which referred to ale\_interface.cpp(), which contained game\_over(), which read simply as “environment->isTerminal()”. This took me to stella\_environment.cpp, where I tried forcibly increasing the value of m\_max\_num\_frames\_per\_episode to like a million.

This appeared to do nothing, so I took it a step further and updated isTerminal(), which seemed to do nothing again. Finally, I overwrote isTerminal() with the line “return false” in an attempt to have each episode play until the game ended. This didn’t seem to work either, although I am still not sure why. Despite all that, I think that my problem was that I was trying to copy “env” by simply stating “tempEnv = env”, but because of the way Python handles pointers, tempEnv simply pointed to the same env, and every step I took in the minimax tree, it counted as a step in the original “env”. This is why I had to use the clone\_state() and restore\_state() functions in my miniflex algorithm. Somehow, this fix allowed me to run games until they were complete, even though all evidence pointed to the problem having nothing to do with the step count. But after all of that, I found that the clone and restore business made the algorithm even slower…

The conclusion of this grueling process of discovery and personal growth is that no matter what I did, the minimax trees that were being generated by the agent could not be deep enough to even reach the step at which the ball reaches the bottom of the screen, or at which the ball collides with a brick. This means that all of the levels of the trees being generated were returning rewards of 0, nullifying the minimax algorithm entirely. I either needed a supercomputer, or to seriously change up what I was doing.

MiniFlex Agent: So I did neither. I was hoping that this one would work better, as it would be building minimax trees less frequently. Unfortunately the result was that even with a very low tree depth, we see a Matrix-style slo-mo shot as the ball approaches the bottom of the game board, and any tree depth that is high enough to be of use runs too slowly to be of any use. Here is a video of the miniflex algorithm operating with a max tree depth of 2: <https://youtu.be/QxUeO1dfCMY>. As you can see it is painful to watch, and the tree depth is so shallow that it doesn’t even provide any useful information to the agent. The agent picks a random action in the cases where the tree provides no useful information, which explains the results below for average score, standard deviation, and max score:



This video shows how slow miniflex runs with a depth as low as 5: <https://youtu.be/SclAURxnBio>. It is easily conceivable that a player would need to start moving 5 steps before the ball reaches the paddle, as the paddle may be on the wrong side of the game board and require 5 steps to make it under the ball. However, constructing a minimax tree that could return this information would take so long that it cannot be implemented. Here lies my project, R.I.P in peace, F. No matter the changes I made, I could not find a way to fix this problem. While I am sure that wrappers would have helped, I cannot convince myself they could have cut a 43 second sequence of operations into a tenth of a second, and that’s what I needed to do.

**Outcome:** The results of this project were far from what I had hoped. I would consider it a total failure if I hadn’t learned so much in the process. So it wasn’t for nothing.

**Some things I did learn:**

* Even with simple agents, wrappers and preprocessing are essential in the process of creating something useful. I can’t shake the feeling that had I applied greyscale and down-sampling correctly, my minimax algorithm could have at least worked efficiently enough for me to test it and correct my blunders. Maybe it would have run at a reasonable speed with a max tree depth of 3 (still not high enough). Unfortunately I can’t say, as there was something I was seriously not understanding about how to apply the wrappers, and I Googled and Googled to no avail.
* No matter the complexity I was expecting, working with software packages like this is apparently worse than I could have dreamed of. This is the first time I have used any kind of software this complicated, and it took me by surprise. I worked on this project more-or-less as much as I possibly could, and the errors and hiccups I encountered every single step of the way were astounding. Every single shell command, every single new function call, and every single algorithm trial… all but a tiny handful, I encountered some new kind of error that either required Googling or a treacherous excursion into the sea of terror that is python\_packages/gym/. I knew I was in the thick of it when all of a sudden I was editing code in C++. The only time I have combined programming languages has been JavaScript and HTML, and I think the outcome of this project is a good reflection of that.
* Hard-coding any kind of agent on visual data alone is a nightmare. Wrappers would have helped, but the crux of the issue for me was having to scan through every pixel to know where each entity was, instead of having a grid-world type lay out that I am used to.

**Further Reflection**

This class is my first 3000 level CSCI course other than algorithms, which did not have many coding assignments. Although I have all of the prerequisites completed, I felt very unprepared for the final project. I do not, however, think it is because of the way the course is set up. I think it would be reasonable to assume that this far along in my degree I would have some experience coding with Python packages like Gym, but I have none. As a result I spent the better part of a week just trying to get the environment set up. I tried it in Jupyter Lab and through my Windows 10 shell before trying it in Jupyter Lab in linux, and I was finally able to get it set up in my Linux shell after giving up on each of the 3 previous attempts. This cost me way more time than it needed to because I spent far too long than I should have trying to resolve the errors I was encountering along the way. Had I known when to give up, I could have probably had the environment set up in a couple days, maybe even a few hours. Unfortunately the only environment I could get working was the last one I tried.

Then came the problem of Baselines. Using Baselines requires Tensorflow 1.14, but Tensorflow 1.14 requires python 3.7 or earlier, and I have Python 3.8 installed. I looked up multiple different ways of downgrading Python and each one I tried presented me with a problem I did not know how to solve and to which I could not find the solution. For this reason I decided to hard code the agents for my project, which limited the scope of what I felt capable of doing and ended up taking a very long time to refine and test, especially considering that an integral part of the algorithm I wanted to code ended up taking hours to execute a single game. This is also a problem I may not have encountered had I been able to properly integrate Baselines, but I’m not sure because I don’t know what kind of learning algorithms are included in that package. It may not even have a minimax algorithm that applies to a breakout game, as it is only single agent. That is another flaw worth mentioning; because breakout is a single agent game, the minimax algorithm I was trying to implement was really just a “max” algorithm, which ends up being fairly redundant especially in the absence of alpha-beta pruning.

**Relevant Links**

OpenAI Gym: <https://github.com/openai/gym>

My Project: [<https://github.com/lunchmeat27/csci3202_final>/](https://github.com/)

Screen Recorder: <https://jp9000.github.io/OBS/features/subregion.html#:~:text=To%20easily%20select%20the%20select,to%20show%20on%20stream%20later.>

Integral Searches:

Setup ALE in Windows (NOPE): <https://github.com/mgbellemare/Arcade-Learning-Environment/issues/251>

Enable Hardware Virtualization: <https://2nwiki.2n.cz/pages/viewpage.action?pageId=75202968>

Setup ALE in Linux: <https://coding.csel.io/user/mica6085/lab?> (RLenvsetup.md)

Weak references: <https://docs.python.org/3/library/weakref.html>

Why I didn’t do DQN: <https://becominghuman.ai/lets-build-an-atari-ai-part-1-dqn-df57e8ff3b26>

Nautilus: <https://askubuntu.com/questions/265641/how-to-open-a-folder-with-gui-from-terminal#:~:text=Use%20the%20command%20nautilus%20%3Cpath,run%20gksu%20nautilus%20.>

I couldn’t find help for a whole lot of this project, which was surprising.