Applying Machine Learning to Pokemon Pinball: Ruby/Sapphire

George Lim, Daniel Gruhn, Kenneth Velarde, Jarod Nakamoto

Abstract

Pinball is a game ubiquitously played because of the simplicity of its design and controls, and the difficulty of achieving mastery. Pokemon Pinball: Red and Blue(Nintendo 1999) is a modification on the classic formula where the player takes on the role of a Pokemon trainer, catching Pokemon not through the archaic formula of battle but through the delicacies and intricacies of a game of pinball. The goal of our project was to make a typical reinforcement learning program learn how to play the game and eventually, accommodate for our additional objectives. In this report we will document our base methodology, as well as the multiple issues and hurdles we encounter in an attempt to implement the program. While our project may have reached an unexpected conclusion, we learned a lot about the process that goes into AI research and understand what it would take to get results (theoretically) in future projects.

I. Introduction

This project is interesting because it is not only based in a franchise that is known and loved internationally, it builds upon one of the most popular games in the world, pinball. What it does differently than classic pinball is the fact that it has bonus modes:

- **Catch 'Em All Mode**
- **Egg Mode**
- **Evolution Mode**
- **Travel Mode**

Each of these different modes follow a similar mechanic in which players must hit specified areas to trigger their respective event. In Catch 'Em All, players are able to catch different Pokemon by hitting them three times during the stage. In Egg Mode, the player must trigger a specified target area four times, where then a baby Pokemon appears moving across the board. Only when it is hit twice will the player capture it. Evolution Mode allows players to evolve their captured Pokemon by hitting all the evolution markers along the board. Lastly, Travel Mode is where the player can leave the board and capture exclusive rare Pokemon based on their respective maps.

Figure 1: In game screen capture.

In the time we spent in researching this game, there was surprisingly minimal amount of examples of AI when it comes to Pokemon Pinball. Our main focus in this project is to not only get our AI to achieve the highest score while being able to complete the four previously mentioned stages. In addition to this, there is actually an in-game shop where players are able to purchase using in-game currency to give them life-savers and improved Pokeballs.

Through machine learning. we planned to implement a way for an AI to get the highest score, play the four bonus stages if it encounters them, and access the in-game shop (if we eventually tested our AI on the Ruby and Sapphire versions of Pokemon Pinball). We are attempting to complete this project in order to understand more about machine and reinforced learning and how capable our AI is

of understanding a game with mechanics as intricate as Pokemon Pinball.

Since we are limited on processing power and Pokemon pinball is a game that could have very long rounds, we concluded that a Monte Carlo search would not be ideal to teach our AI how to play the game. In order to meet the necessary speed requirements, our AI needs to make excellent decisions quickly. A particular problem that our AI might encounter is the version exclusive Pokemon problem. Since the game is intentionally designed so that a player can not catch all the Pokemon in a specific version and must play both versions to catch them all, our AI might get stuck in one version of the game playing infinitesimally in the hopes of encountering new Pokemon. To counteract this problem we intend to also train our AI to recognize which Pokemon are exclusive to each version (the pinball map).

II. Related Work

The game of pinball itself is classic enough to have been adapted across multiple platforms. One such iteration is *Video Pinball* on the Atari, which happens to be another one of the environments contained in OpenAI Gym.(OpenAI 2016) Research has been conducted at the University of Virginia using a variation of the Deep Q Learning algorithm combined with deep learning and a neural network in order to train an AI to run through the game.(Rosenberg and Somappa 2017) There also exists the Small Planet's pinball project, which also uses a similar deep Q learning method in order to implement an app that allows the program to play physical pinball machines.(Planet 2017) As a quick overview of the concept, Deep Q Learning involves a neural network which allows the program to analyze the vast amount of states generated by the game and forgo the usual proportionally large amount of memory cost. These states then are processed using an epsilon-greedy approach towards the optimal result through your typical reinforcement learning methodology. Given the identical core nature of our project, one could imagine that our group would take a similar approach to our problem. As this is essentially a hybridization of the two techniques of Deep and Q learning, it would only be fitting that the aim of our project is the addition of a few more techniques in tandem to account for our secondary objectives.

In regards to the acquisition of our own game environment, we found multiple resources. One was to use the open source deconstruction of the game provided by the dedicated fan base of the genre.(Github 2018) We also have the work of a particular group in Thailand who worked towards developing a process to train AI through the use of emulation.(Thunputtarakul and Kotrajaras 2006) The summation of their procedure was to develop a technique to observe relevant states in the game, and normalize this data into values the AI can use and allow response through input in the emulator. While the article may be dated, there are still concurrent updated models of VisualBoyAdvance and Python today that could follow the principles of their techniques to obtain a similar acquisition of our data set.

Originally, our planned methodology was based on utilizing a Gameboy Advanced emulator to run Pokemon Pinball Ruby and Sapphire version in a similar fashion, with our intentions being quite similar to develop an AI trained with reinforcement learning to maximize score and complete our ultimate objective. However, due to technical difficulties, we had to switch to the original Pokemon Pinball. Our ultimate objective along with the standard maximization of the play duration and score differ since it is as any true Pokemon fan's objective: to catch all the Pokemon in a given version. Furthermore, we intend to catch all the Pokemon within Pokemon Pinball in both the original and Ruby and Sapphire versions, so our reinforcement algorithm needs to train the AI for both.

III. Notable Platforms and Resources

In this section, we go into detail about the libraries that are considered core to our program

A.OpenAI gym/gym-retro

Prior to the release in 2017 of the OpenAIs Gym toolkit, a good majority of the effort put towards research of reinforcement learning algorithms had to be put into getting the game data sets that must be trained on in the first place. Game researchers looking for a test environment would be forced to manually alter the program of an existing emulator in order to receive data about the states of the game, then after feeding it into their learning model would need to further implement the program necessary to simulate input directly into the emulator. With the subsequent release of the Gym library however, an Atari emulator would be packaged wrapped already with the tools and program necessary to automatically implement the their test models and streamline the entire process. They would continue to expand upon the principle next year in 2018 with the open-source release of the Gym Retro(OpenAI) library, which further widened the library of games that can be implemented by expanding the consoles in which they could support and including methods for users to implement their own ROMs into the library.

The gym library also offers a dedicated UI which is intended to assist with multiple aspects of the the development process, including the visual representation of the training process, the ability to modify the game state to start at a specific instance, and with the ability to record individual replay files for later review. Unfortunately however, the UI is currently only available for Mac and Linux computers, and after that we ran into further issues regarding the installation of the UI which will be expanded upon later in the report.

B.Keras

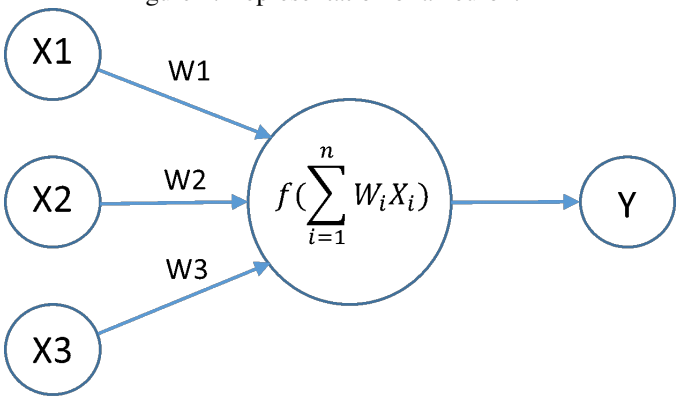
Early on into the project, we ran into issues with Pytorch, a deep learning platform that we had used in our previous projects. Thus we were recommended by our professor, Dr. Summerville, to supplement Pytorch by installing the Keras(Chollet and others 2015) neural network library. Using the users choice of machine learning library as a back-end, Keras is designed as a higher-level API than Pytorch, allowing it to be more simple and interchangeable by nature, but trading performance in return. As of December 2018 however, Keras is not compatible with the current 3.7 version of Python, so further time was needed from our group to downgrade Python and re-install the platforms associated with our project.

C.keras-rl

The purpose of the keras-rl(Plappert 2016) library is to further trivialize the process of implementing reinforcement learning with methods working in direct tandem with OpenAI and running automatic implementations of learning algorithms with the assistance of the h5py(Collete 2008 NNNN) module to handle the large amounts of numbers. By taking components and parameters like the environment, neural model, and a processor class as fields, the keras-rl platform can run standardized training and tests in almost a black box style. While we ran into some issues with the formatting of the action space and the testing method, overall the process of development associated with the emulation itself was greatly facilitated.

IV. Methodology

Figure 2: Representation of a neuron.



This section is dedicated to explaining the technical terms and theory behind our project

A.Convolutional Neural Networks, the ”deep” part of the equation

For those unfamiliar with the concept of neural networks, they are essentially groups of specially designed nodes called neurons organized in a way fashioned after the organization pattern of biological neurons in the brain. In order to have the computer learn how to distinguish specifically non-linear features, multiple value sets of data that are fed into the network, which is referred to as the input layer of nodes, are multiplied together to get a value which is then further transformed by a non-linear activation function in order to get a smaller output layer. The process of convolution works very much in the same way, in which a matrix of values are stepped across by a matrix of weights called a kernel producing a smaller matrix of values. Combining these two techniques into a convolutional neural network in essence allows the agent to learn features by scanning over images or image sets. After each layer, values determining the presence of a slightly more specific non-linear feature is found within the image, which is often repeated over multiple 'hidden' layers in between the initial data and the final output in order to eventually identify high-level features. This multitude of layers in the neural model is where the term 'deep learning' comes from.

B.Reinforcement 'Q' Learning

Figure 3: Q Learning Formula.

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}}$$

Reinforcement learning stems from the principle that the intelligent agent is given an initial state and a set of actions to transition to other states. Its overall goal is to choose actions that max out the reward it gains by assuming favorable states determined by a reward function. Q learning is a specific instance of choosing actions based upon calculating the 'quality' (hence the Q) of an action leading to a certain state by teaching itself values based on a variety of variables. Firstly, the agent must decide through random chance whether to 'explore' or 'exploit'. This is essentially the implementation of an epsilon-greedy policy, wherein based on a probability deemed the learning rate, the program either decides to take the most optimal reward based on its current knowledge or pick something entirely random in an attempt to find an overall greater answer and avoid local minima. Algorithms usually start out with a high learning rate, taking completely random actions to learn different routes, then gradually scale down into more 'exploits' once the computer has hopefully examined a sufficient amount of data. The perceived amount of reward given by a specific action is based off the amount of potential reward that could result from an action, which is then modified by two main factors, the actual probability of receiving that reward and the discount factor, which scales the value off based on how far away it is in the perceived future steps.

V. Implementation

In this section, we go over the development process and the issues we encountered.

Originally we had planned on running the Gameboy Advance version of Pokemon Pinball, due to the increased complexity of the game added in with separate game instances such as the shopping system. After thorough searching across the Internet however, we came to the conclusion that a majority of the ROMs available online for both the Color and Advance versions had disabled the right flipper as a measure against piracy. Luckily, collaboration with our professor allowed us to locate a working copy of the game which we could run.

Figure 4: process-observation method in retro-processor.

```
def process_observation(self,_obs):
    assert _obs.ndim == 3  # (height, width, channel)
    img = skimage.color.rgb2gray(_obs)
    img = skimage.transform.resize(img,INPUT_SHAPE)  # resize and convert to grayscale
    assert img.shape == INPUT_SHAPE
    return img  # saves storage in experience memory
```

In order to implement keras-rl, we were required to set up a class called Retro-Processor, which essentially allowed modification of the way frames and rewards were handled. Our general methodology followed the typical construction of a model for similar tasks. Rather then feed our neural network output layers of single frames, keras-rl would use the process-observation method to simplify and downscale the image array using the method in Figure 4. While a typical convolutional neural network searches a single height x width x color channel image, our input layer is in a empty x height x width x frame format, taking random windows of 4 frames and inserting them together and organizing them by number. This process allows the model to take into account the speed of the ball while in motion. As seen in the code above, we found a library that allowed us to simply streamline the process the of gray-scaling and shrinking the frames in the form of scikit-image or skimage(van der Walt et al. 2014).

Figure 5: The argument dictionary.

```
args = edict({
    'mode': 'train',
    #'mode': 'test',
    '--env-name': 'PokemonPinball-Gbc',
    'weights': None
})
```

We would then run into another snag as we tried to pass arguments into the keras-rl agent. The main program we were using to code, Jupyter Notebook does not support the direct parsing of arguments. In order to work around this limitation, we installed the Easydict(Leplatre 2018) library, which allows the creation of special dictionaries like the one seen in Figure 5 where the entries can be called as attributes and thus as arguments, and then manually manipulated the argument states in order to toggle between testing and training mode.

Figure 6: Setting up variables for keras-rl.

```
env = retro.make(game='PokemonPinball-Gbc', record='.')
np.random.seed(123)
nb_actions = env.action_space.n
```

The next step is to use gym-retro to create the environment space the game is going to run in. Due to issues with testing, we had to run an alternative method which allows the system to keep a .bk2 file on record which will eventually be processed for viewing by a separate file dedicated to the conversion of .bk2 to .mp4 files. Other components, such as the action state and shape of the input layer are also being instantiated at this point.

Figure 7: The neural network model we tested with.

```
input_shape =(WINDOW_LENGTH,)+INPUT_SHAPE
model = Sequential()
if K.image_dim_ordering() == 'tf':
    # (width, height, channels)
    model.add(Permute((2, 3, 1), input_shape=input_shape))
elif K.image_dim_ordering() == 'th':
    # (channels, width, height)
    model.add(Permute((1, 2, 3), input_shape=input_shape))
else:
    raise RuntimeError('Unknown image_dim_ordering.')
model.add(Conv2D(32, kernel_size=(8, 8), strides=(4, 4),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Conv2D(64, kernel_size=(4, 4),activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3),activation='relu'))
model.add(Flatten())
model.add(Dense(3136, activation='relu'))
model.add(Dense(512)) #'activation=softmax'
model.add(Dense(nb_actions))
model.add(Activation('linear'))
```

After that we begin construction of our neural network. Following the guidance of the Atari implementation of -rl, we used the standard Keras Sequential Model to create a network with layers as seen in the figure above. It should be noted that the purpose of the Relu layers beneath is to set any negative values generated by the system to zero to preserve the systems non-linearity. After specific features are recognized by the convolutional part of the network, the values are then flattened and run through fully-connected network layers in order to adjust the weights and learn from the input frames. We did not need take a soft-max of the output data because our output layer was already in the 1x9 array we needed to insert as the next state's action space.

Figure 8: The assembly of the DQNAgent.

```
memory = SequentialMemory(limit=1000000, window_length=WINDOW_LENGTH)
processor = retro_processor()
policy = LinearAnnealedPolicy(EpsGreedyQPolicy(), attr='eps', value_max=1., value_min=.1, value_test=.05,nb_steps=1000000)
dqn = DQNAgent(model=model, nb_actions=nb_actions, policy=policy, memory=memory,
               processor=processor, nb_steps_warmup=50000, gamma=.99, target_model_update=10000,
               train_interval=4, delta_clip=1.)
dqn.compile(Adam(lr=1), metrics=['mae'])
```

Once all the components are set, the DQNAgent can be instantiated. Setting variables such as learning rate and memory size are able to be modified simply by changing the corresponding variables in Jupyter Notebook. After compiling, it then receives an argument for the mode in order to determine whether to enter training or testing mode.

Figure 9: The code that runs if the mode is set to train.

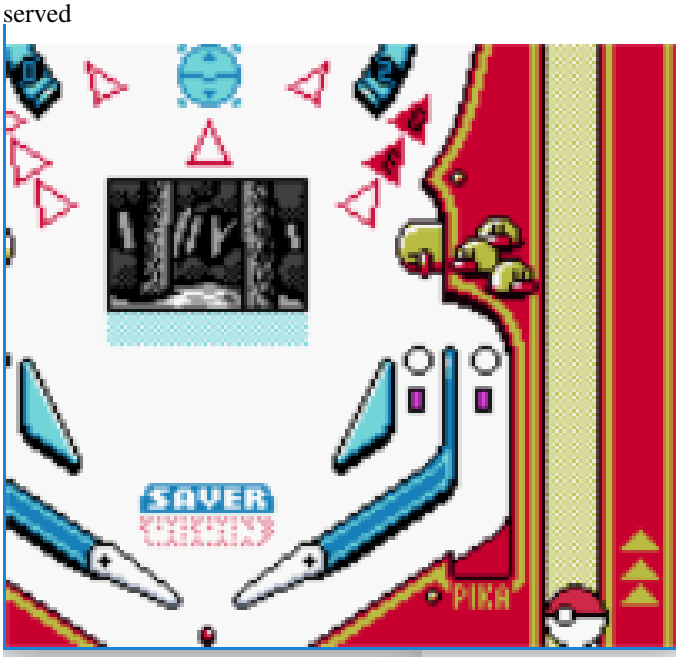
```
weights_filename = 'dqn_{}_weights.h5f'.format('PokemonPinball-Gbc')
checkpoint_weights_filename = 'dqn_' + 'PokemonPinball-Gbc' + '_weights_{step}.h5f'
log_filename = 'dqn_{}_log.json'.format('PokemonPinball-Gbc')
callbacks = [ModelIntervalCheckpoint(checkpoint_weights_filename, interval=250000)]
callbacks += [FileLogger(log_filename, interval=100)]
history =dqn.fit(env, callbacks=callbacks, nb_steps=175000, log_interval=10000)

# After training is done, we save the final weights one more time.
dqn.save_weights(weights_filename, overwrite=True)
```

The training code first sets up a save point for both the weights and a callback log, then proceeds with the keras-rl fitting, displaying the progress in the output log. After a number of steps/epochs determined by the user, one final save is done to the weights.

VI. Results, Issues, and Speculation

This section provides the overview of the overall results and the our approach to meeting the initial objectives set out at the start of this project

Figure 10: Definite signs of the pregame/running can be observed



It is when we tried to run the testing algorithm that the most prominent of the issues with our project surfaces. The program would then soft-lock itself into the state depicted above, never deciding to launch the ball in the first place with only the black/white screen in the middle shifting scenery in the pregame animation. During our testing phase, the ball would just sit in the initial state indicating that the AI was apparently unwilling to play the game. Since, test was the method we had to visualize if something went wrong with the training, we were forced to rely on alternative measures, namely the downloading of ffmpeg and conversion of previous runs of the training phase into movies through a complex process involving another program and running a command in the Anaconda prompt, in order to confirm what exactly was happening in the program. After reviewing the training videos, we found that the training code at least appeared to be working as intended by playing the game repeatedly. Whatever our issue was, it appeared to be exclusive to keras-rl's testing code, but that was improbable which consisted of two steps, loading in the weights from the current file, and testing those weights based on a method in keras-rl's library. In order to determine what was going on, we set the learning rate to 1 and observed the ball attempting to apply input but without moving. We suspect it may have to do with over-punishing the death state, and thus opting to preserve the reward by not playing the game at all. We figured that if we could adjust the initial game state to a point where the ball is already leaving or outside of the plunger area, it would force the game to act and hopefully break the lock. But to do so would require access to the OpenAI UI, which proved to be fairly difficult when the majority of our operating systems were Windows. The one member of our group that did have a Mac attempted to install the UI software, but failed overall due to an inability of his laptop specifically not being able to run Cmake in order to install the assets. Meanwhile the majority of us were trying to run Linux on virtual machines in order to get the UI running, but we hit a dead end there too due to time constraints and opposing schedules. Alternatively we could have gone into the program and adjusted the rewards formula, but alas, we came to the conjecture too late, as finals week are already upon us and report is due soon.

Reflecting back on the goal we set out at the onset of the semester, our objectives have shifted quite a bit. While we were stoked to learn that a majority of the time-intensive processes associated with our project could be streamlined, we also had to compromise on the fact that the we wanted to train on from the get go would be handicapped by anti-piracy measures. We expected our program to get to the extra modes which it would have to learn eventually as we wished to observe, and thus expected most of our problems from trying to teach it other modes, rather then getting the game to run in the first place. There was a general improvement of scores in the epochs near the end of the training algorithm when compared to simply telling the game to take actions randomly, however. Though the argument has been brought up that even the random algorithm can have a

few 'lucky' streaks by spamming the tilt buttons enough to slow the ball to the point where the random timing on the flippers is enough. A fine example of the changes over time towards our project scope when issues began to appear while trying the develop the basic algorithm is when you look at our original timeline during the project proposal phase as well as our initial statement regarding the possible issues that would arise.

Estimated Timeline

- Firstly, we decided to build an AI to play an emulated Gameboy Advanced Game. After some deliberation, we decided that the Gameboy Advanced game that we would create an AI for would be Pokemon Pinball since it has two easily evaluated criteria, score and Pokedex (a record of unique Pokemon caught).
- By the 7th week, we intended to have developed an interface between our artificial intelligence and our emulator. We also plan on having our AI's reinforcement learning algorithm developed concurrently also.
- We plan on building a prototype for our AI by the 10th week and intend to test and improve it afterwards.
- We plan on having our AI completed by the final week and more autonomous in the way it learns how to play the game.

An issue that we expected our AI to encounter is the evolution problem. In the game, some Pokemon are only obtainable through evolution and thus our AI needs to know how to evolve Pokemon and keep track of which Pokemon whose evolved forms that it does not already have. In the main series, evolution is usually accomplished through combat and acquiring experience to level up until the Pokemon reaches the level threshold for its particular species and then it will evolve. However, in Pokemon pinball, evolution is accomplished through a mini-game called evolution mode. Since evolution mode is the only evolution method available in Pokemon pinball, then the evolution problem is reduced from knowing the many different combinations of combat, trading, and item usage to merely keeping track of which Pokemon our AI has not evolved yet and how to access evolution mode. After the evolution mini-game has been activated, evolving the selected Pokemon is merely an issue of selecting which Pokemon to evolve and recapturing it using the game's normal capture mechanics.

Another issue we expected the AI to experience that we would have to account for is if the AI opens the in game shop or some other menu (like the Pokemon evolution menu). Since both the evolution and shop menu will not time out naturally, there would probably be an issue that our AI will be engulfed in a menu that it does not know how to exit out of or the selection of an unfavorable outcome. An unfavorable outcome for the evolution menu would be that it selected a Pokemon with an already explored evolution tree while Pokemon with unexplored evolution trees are available. An unfavorable outcome for the shop menu would be wasting all of the AI's valuable coins on something with minimal utility instead of not buying anything to save for something that might be incredibly useful like an extra life. We intended to have the AI learn which item to buy or not buy on its own since the individual utility of a purchase is dependent on the situation. For example, a ball upgrade might be more useful when catching a particularly challenging Pokemon like a legendary Pokemon while the extra life might be more valuable if the AI is on its last life.

The AI would have had to collect data on moves that cause it to lose lives and actions that cause successful Pokemon capture and elongated play times. This data will be utilized to improve the rate of capture and play time, maximizing the amount of Pokemon caught and score. We expected the AI's movements collected as they are inputted in the AI-Emulator interface. Also, the game state recognition part of the AI will record whenever a Pokemon is caught, a special game mode is triggered or the ball is dropped.

VII. Concluding Thoughts

In the current state of our project, we have no real way to determine how close we are to the goal, which could be certainly be interpreted as being quite a distance away from achieving it. If one were to continue our research or attempt to do a similar task, we would definitely recommend first making sure the group can access the OpenAI UI, as a vast majority of the convenience and utility offered by gym stems from the application. While it is clear that there are imperfections within the systems designed to facilitate emulation, the fact that a rookie team like us could get this far in the first place is a testament to how simplified the process has become relatively. At the very least, having had to troubleshoot most of the problems have allowed us to gain a deeper understanding of the process and code in itself as an attempt to fix it.

References

Chollet, F., et al. 2015. Keras. https://keras.io.

Collete, A. 2008-NNNN. H5py library. http://docs.h5py.org/en/latest/quick.html.

Github. 2018. Dissembly of pokemon pinball. https://github.com/pret/pokepinball.

Leplatre, M. 2018. Easydict library. https://pypi.org/project/easydict/1.2/.

Nintendo. 1999. Pokemon Pinball. [GameBoy Color].

OpenAI. Openai gym-retro.

OpenAI. 2016. Openai gym. https://gym.openai.com/envs/.

Planet, S. 2017. Pinball wizard. http://www.smallplanet.com/soapbox/toolkit/pinball-wizard/.

Plappert, M. 2016. keras-rl. https://github.com/keras-rl/keras-rl.

Rosenberg, A., and Somappa, G. 2017. Pixel to pinball: Using deep q learning to play atari. http://www.cs.virginia.edu/ gs9ed/reports/Reinforcement$_{L}earning_{P}aper.pdf$.

Thunputtarakul, W., and Kotrajaras, V. 2006. Aitem: Testing ai in commerical game with emulator. https://www.cp.eng.chula.ac.th/ vishnu/gameProg/papers/wpj$_{c}games063.pdf$.

van der Walt, S.; Schönberger, J. L.; Nunez-Iglesias, J.; Boulogne, F.; Warner, J. D.; Yager, N.; Gouillart, E.; Yu, T.; and the scikit-image contributors. 2014. scikit-image: image processing in Python. *PeerJ* 2:e453.