UTILIZING DEEP Q-LEARNING VARIATIONS TO PLAY *GALAGA*

TEAM NEOCOGNITRON

TEAM NEOCOGNITRON

Nathan Byrnes

ndb3w@mtmail.mtsu.edu

https://github.com/NathanByrnes

Chris Gerspacher

cag7a@mtmail.mtsu.edu

https://github.com/cag7a

Jonathan Gregory

jbg4a@mtmail.mtsu.edu

https://github.com/vat880

Cameron Justice

cj4g@mtmail.mtsu.edu

https://github.com/cameron-justice

Ian Seal

ins2c@mtmail.mtsu.edu

https://github.com/masturffect

Justin Wade

jww5f@mtmail.mtsu.edu

https://github.com/JustinWade

INTRODUCTION

- **Goals**
  - Comparisons of Different Deep Q-Learning variations by playing the Classic Arcade Game Galaga
  - Analyze their end-performance and learning rate over a short-term learning environment
  - Discover an understanding of the Deep Q-Learning architecture

- **Tools/Dependencies**
  - OpenAI Gym-Retro Environment
    - For accessing *Galaga* and sending inputs
  - Virtual Environment
    - For managing dependencies and python versions

MOTIVATIONS & KEY AIMS

- Compare & Contrast Four Implementations of the Galaga Net based on Score
  - Single Q-Learning with Uniform Replay Memory
  - Single Q-Learning with Prioritized Replay Memory
  - Double Q-Learning with Uniform Replay Memory
  - Double Q-Learning with Prioritized Replay Memory
- Find the best variation of the Net to play Galaga by direct comparison of results

GENERAL STRATEGY

Our approach to this research was to implement the simplest of DQN agents, the Uniform Memory agent, and adjust hyperparameters that work best for it, as all other implementations are expansions on this agent.

Primarily adjusting:

- Epsilon Decay
- Learning Rate
- Image Size

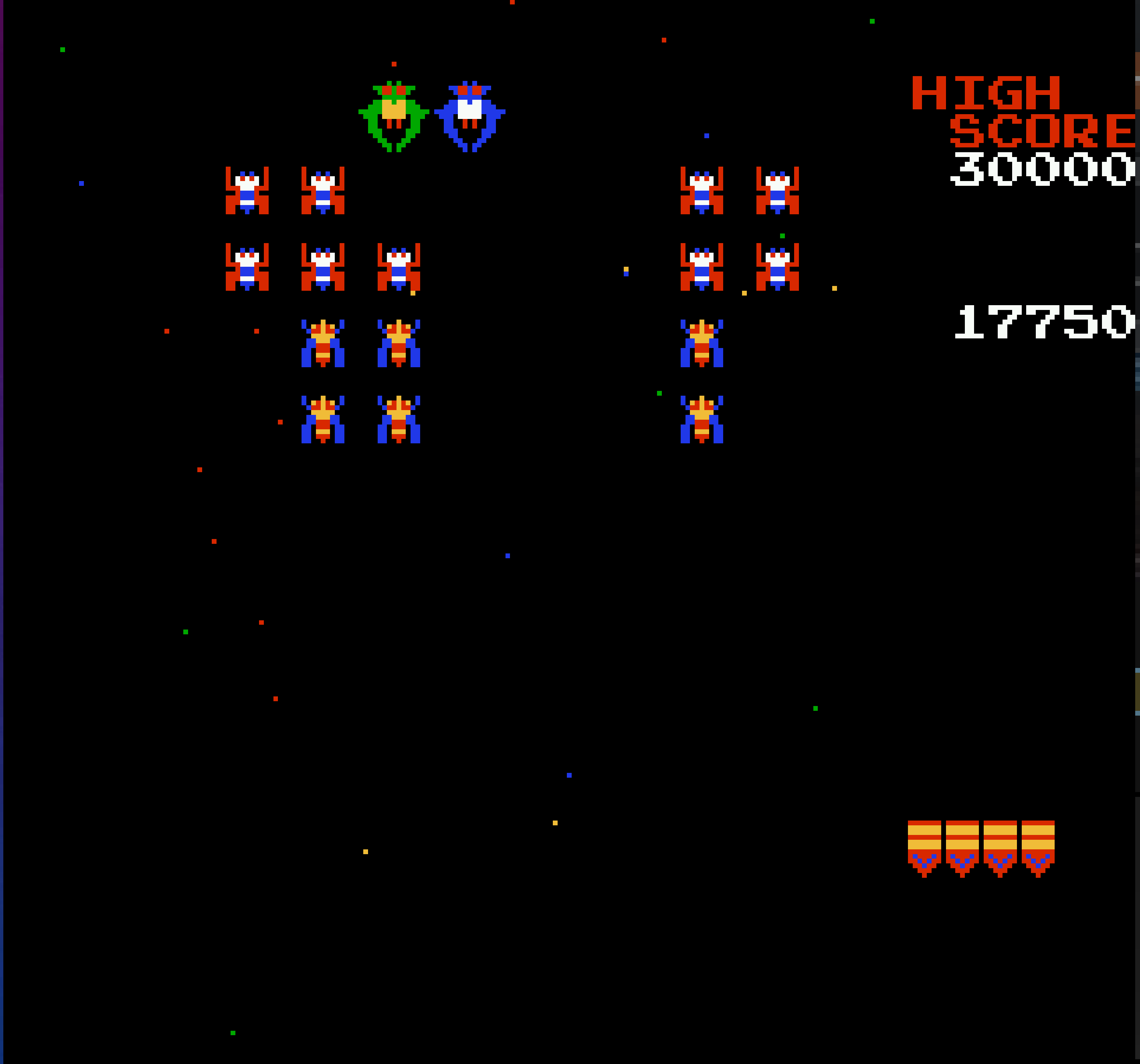
Once we felt the parameters were sufficient, we began training all networks for 1000 epochs (due to time constraints) averaging at five million frames experienced.

Our further comparisons are based on comparative data from the original DQN, Double DQN, and Prioritized Experience Replay results.

Network Input

- Single frame of the game, processed to an 84x84 pixel grayscale* image.

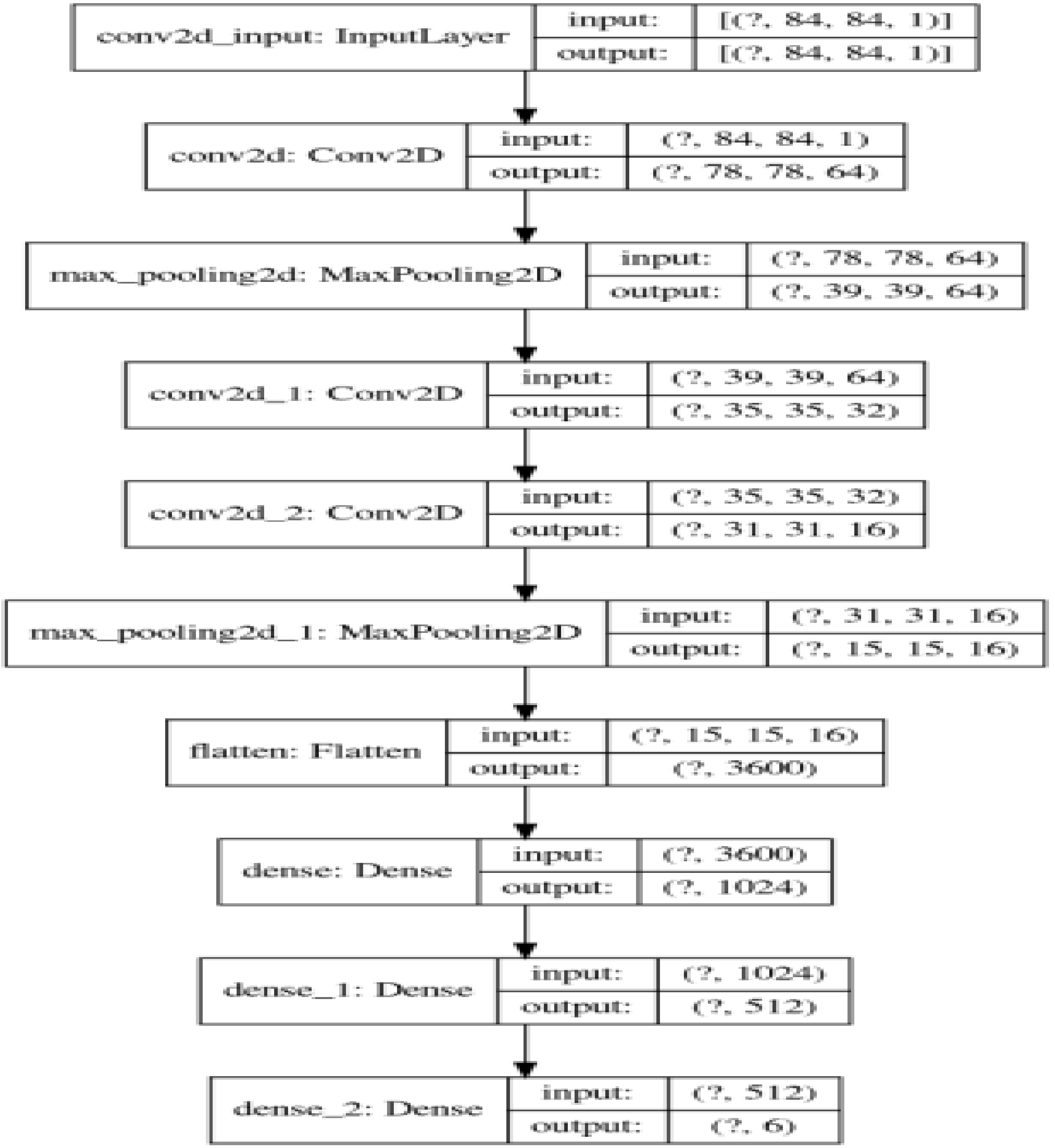
- Passes through



*Grayscale not shown for design

NETWORK ARCHITECTURE

- Categorical_crossentropy loss
- Nadam optimizer with a 0.0025 learning rate



UTILITIES

- Grayscale
  - An optimized algorithm to convert a 3-dimensional array of RGB pixel representations to a 2-dimensional grayscale pixel representation
  - The elimination of an entire dimension allows image processing to happen significantly faster with minimal changes in accuracy
- Resize
  - A very simple function to resize the image array to our supplied hyperparameters
- Preprocessing
  - A simple wrapper function to call resize and grayscale

UTILITIES - LOGS

- A series of functions to make logging data easier
  - log_create – generates the logfile and returns its path
  - log_params – initializes the log with information about the model
  - log_output – a print wrapper that both writes the output of the net to the logfile and stdout

REPLAY MEMORY

- Replay Memory
  - Stores Past Experiences for the Net to reuse
  - Learn at a faster rate rather than not storing anything (does not immediately discard experience after update)
- Uniform
  - Most basic form of Replay Memory
  - Possible to break temporal correlations
  - Rare experiences used more often
- Prioritized
  - "Prioritize" experiences that might be more useful to learn from later
  - Assign their sample probability based on their TD-Error
  - These experiences deemed more useful, so are more likely to be learned from

Q-LEARNING

- Implementation
  - We chose to implement Q-Learning in our project because of the standards it sets and popular usage in similar Atari game applications. It's also advantageous for its simplicity and ease of implementation when learning from a buffer. This allowed us to build onto already developed research instead of using our limited resources reinventing the wheel.
- Single Q
  - Algorithm allows agent to learn a response from a given action by setting a reward.
  - Agent learns the best action in a given state by trying every action and updating the expected reward with the actual reward for its action.
  - Often is overoptimistic and leads to overestimated values.
- Double Q
  - Solves overestimation (via 2nd evaluation of Q with an inner Q, hence Double Q)
  - Decouple the selection from the evaluation

OPTIMIZATION

- Architecture
  - Before Final Net
    - The sequential model only had one conv2D input layer of size 64 with a kernel size of 11 x 11 with a dense layer and an output layer
    - This proved to be inefficient due to the high kernel size and one conv2D layer. When running the network there seemed to be a loss of image data because of the high kernel size.
  - Final Net
    - A sequential model with three convolutional layers and two dense layers

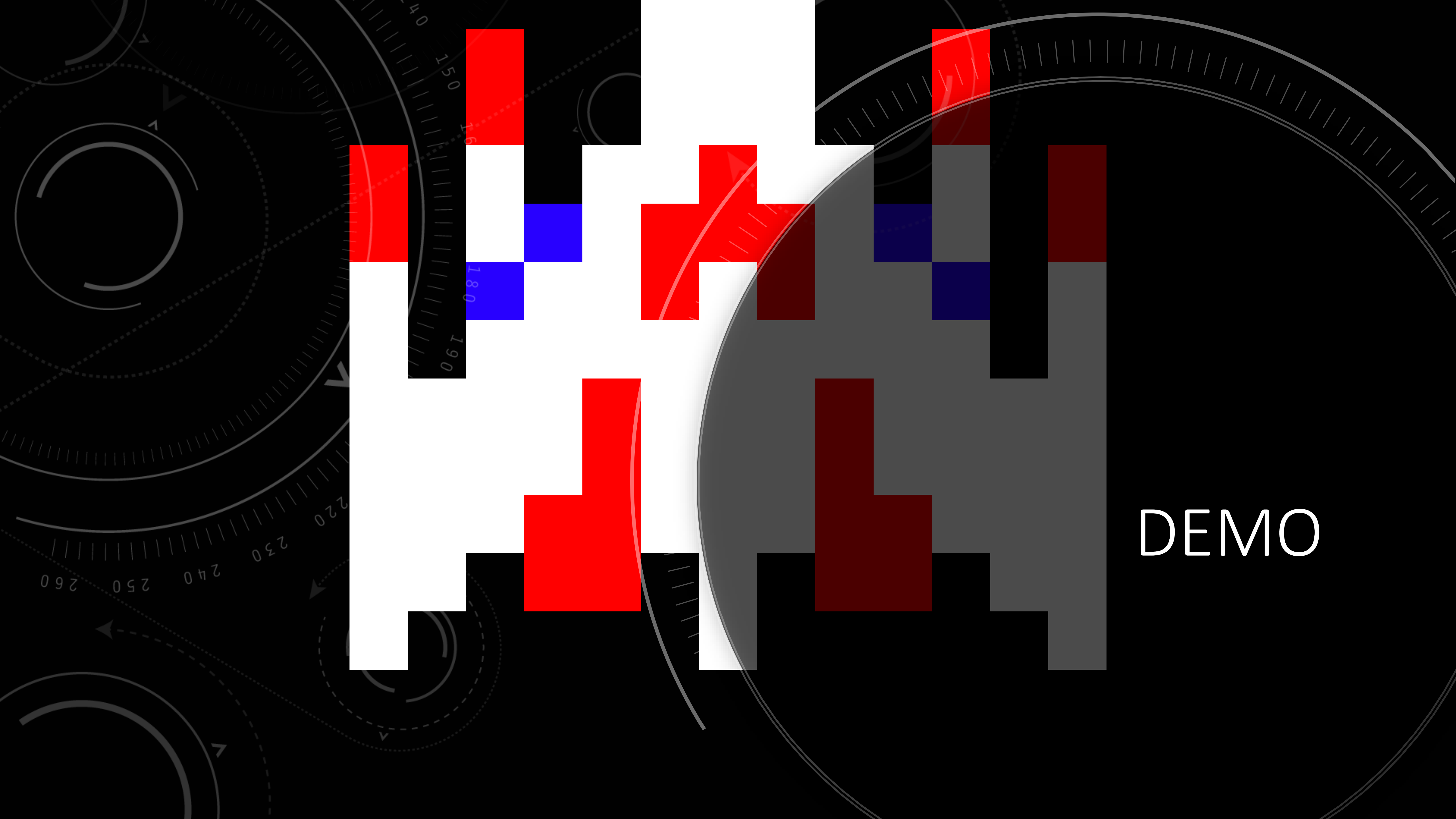
FINAL PRODUCT

- Implemented all four implementations, and were able to compare them within a short-term training session
- Failed to complete a full implementation of established DQN architecture (Frame skipping, Sumtree Prioritization)

- Access repository page here!

TEAM CONTRIBUTIONS

- Nathan : Double Q-Learning, Replay Memory & Presentation
- Chris: Utilities, Paper & GPU Training
- Ian: DQN Model & Paper
- Jonathan: DQN Model & Paper
- Cameron: Project Management, Prioritized, RetroGym & Paper
- Justin: Single Q-Learning & Presentation



DEMO

QUESTIONS?