References: [1] Author et al. (2023). Title of Paper. [2] Author et al. (2024). Title of Book.

DEEP LEARNING FRAMEWORK FOR "RUN 3"

Malcolm Grant, Kai Burgermeister, Ely Brayboy, Kaspar Pitblado
Department of Computer Science, Brown University

1. Introduction

We wanted to train a model to learn how to play Run3 on Coolmath Games. The goal of the game is to maneuver the "citizen" through the various 3D environments using the arrows on the keyboard. Unlike conventional RL benchmarks that expose structured game state information, a browser based setting offers no direct access to internal variables, requiring the agent to operate solely on the visual stream presented to the player. To address this, we developed a fully **vision-based learning system that captures real time screen observations and processes them through a convolutional neural network**. The system then selects keyboard actions using a Proximal Policy Optimization (PPO) framework.

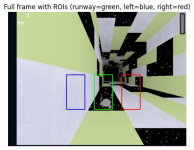
Although this game may be simple for a human, we faced many challenges:

- Difficult environment
  - The environment is 3D with hard-to see holes
  - Requires some looking ahead
  - Running on walls rotates the entire environment
  - Diverse obstacles like falling panels, speed boosts, ramps, boxes
- Slow data collection
  - Cannot simulate the environment, need to collect real time
  - Only about 3 pieces of data per second
  - Limited by actions and screenshot overhead
- Action representation
- Reward shaping

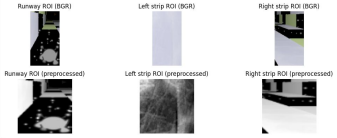
2. Dataset and Preprocessing

Our system employs a complete end-to-end pipeline that begins with real-time screen capture of the Run 3 browser game at 3-4 frames per second using the high-performance mss library.

Each captured frame (725×545 pixels) undergoes preprocessing: conversion to grayscale to reduce dimensionality, resizing to 128×128 pixels for computational efficiency, and stacking with the three most recent frames to create a 128×128×4 tensor that encodes temporal motion information (as illustrated in the accompanying figure).

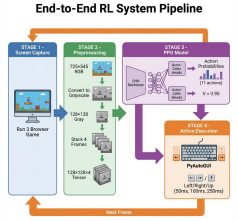


Additionally, we define three regions of interest (ROIs) within each frame:
a central runway region for calculating platform alignment rewards (0.0-0.25 bonus based on pixel occupancy), and left/right wall strips for penalizing not being in the correct orientation.



3. Methodology

Following data preprocessing, the observation feeds into a PPO-based Actor-Critic architecture featuring a CNN backbone with three convolutional layers (32, 64, and 64 filters with progressively smaller kernels and strides) that extract hierarchical visual features, followed by a shared fully connected layer (512 units) that branches into two heads: an actor network that outputs action probabilities over 11 discrete actions (no-op, directional movements at varying durations of 50ms, 100ms, and 250ms, plus diagonal combinations), and a critic network that estimates state values for advantage computation.



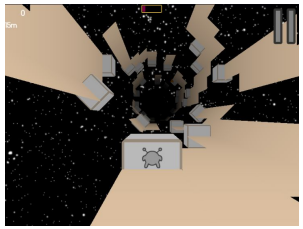
Have a look at our System Pipeline Diagram to understand how all the pieces fit together!

4. Results

After a few dozen hours of testing and training, we were able to achieve an algorithm which could play the game somewhat well and survive many obstacles. However it was very inconsistent, and still often jumps into the void for no reason. If you watch for minute or so, you will almost definitely see a few moments of brilliance from our little parkour alien.

Reward Exploitation (The "Box Strat")

After about 6 hours of training, our agent learned to exploit our reward function by sitting behind these boxes and infinitely accumulating survival points while never dying. This was extremely frustrating and warped many good trains and policy networks. To work around this we hard coded a solution to move the agent manually when it did this.



5. Discussion

There are several straightforward ways this project could be improved if we had more time. The most obvious is simply training the agent for much longer; running 200+ epochs instead of ~100 would almost certainly help smooth out the high-variance behavior we see now. We could also test different action spaces, to see what the best balance is between control and simplicity. Adjusting hyperparameters more systematically would probably make training more stable as well. And in the long run, we would love to be able to do parallel training with multiple computers, sa the training process is very inefficient. We might want to rethink the way we preprocess the images as well, as the breakable gray tiles are almost indistinguishable in grayscale from some other colors.