Introduction

Project Concept

The project focuses on developing AI systems capable of playing the classic game Super Mario Bros. This undertaking is aligned with Project Idea **Title 1 from the CM3020 - Artificial Intelligence course template, titled "Kane and Abel: AIs that play games."** The essence of the project is to implement and compare two distinct AI approaches to game playing. One AI will be designed with a pre-programmed behavior using a finite state machine (FSM) or a similar deterministic method. The other AI, which has already been developed, utilizes **reinforcement learning techniques** to learn how to play the game autonomously.

The Idea

The primary goal of this project is to explore and demonstrate the differences between rule-based AI systems and those that learn from experience. By developing two AIs, one pre-programmed and one learning-based, the project aims to provide insights into the effectiveness, strengths, and weaknesses of each approach in the context of a real-world gaming scenario. The pre-programmed AI will serve as a controlled benchmark, while the learning-based AI showcases the adaptability and potential of modern machine learning techniques. This dual approach allows for a comprehensive analysis of how AI can be applied to game playing, shedding light on the practical applications and theoretical underpinnings of each method.

Chosen Template

The template chosen for this project is from the CM3020 - Artificial Intelligence course, specifically Project Idea Title 1: Kane and Abel: AIs that play games. This template provides a structured framework for comparing different AI approaches to game playing, ensuring that the project adheres to innovation and creativity in AI development.

Motivations

The motivation behind this project stems from several key factors:

1. **Educational Value**: Game playing has long been a benchmark for AI research. Developing AI that can play games provides an excellent opportunity to apply theoretical AI concepts in a practical setting. This project allows students to engage with both traditional AI techniques and modern machine learning methods, fostering a deeper understanding of both. The practical application of these concepts in a gaming environment makes learning more engaging and effective, as students can see the tangible results of their work.

2. **Technological Advancement**: The comparison between a pre-programmed AI and a machine learning-based AI can highlight the advancements in AI technology. This can provide valuable insights into the state of AI development and its potential future

directions. By implementing both approaches, the project can demonstrate the capabilities and limitations of each method, offering a clear view of where AI technology stands today and where it might be headed in the future. This dual approach also allows for a detailed analysis of how different AI techniques can be optimized for specific tasks, contributing to the broader field of AI research.

3. **Engagement and Entertainment**: Games like Super Mario Bros. are not only iconic and universally recognized but also provide a visually engaging and interactive platform for showcasing AI capabilities. This makes the project interesting and accessible to a broader audience, including those without a deep technical background. The familiar setting of Super Mario Bros. serves as an excellent medium for demonstrating complex AI concepts in an understandable way, making the project appealing to a wide range of viewers. The game's visual and interactive nature also makes it an ideal platform for testing and demonstrating the effectiveness of different AI approaches in real-time scenarios.

4. **Skill Development**: This project encourages the development of various skills, including programming, algorithm design, machine learning, and critical evaluation. These skills are highly valuable in both academic research and industry applications. Working on this project requires a multidisciplinary approach, integrating knowledge from computer science, mathematics, and cognitive science. This hands-on experience helps students develop a well-rounded skill set that is applicable to a wide range of fields, from AI research to software engineering and beyond.

6. **Public Interest and Outreach**: The use of a popular game like Super Mario Bros. can help raise public interest in AI research. Demonstrating AI capabilities in a familiar and entertaining context can make the concepts more accessible and engaging for the general public. This increased visibility can help demystify AI technology and promote a better understanding of its potential and limitations, fostering greater public support and interest in AI research and development.

Literature Reviews

First literature :

Świechowski, Maciej. "Game AI competitions: Motivation for the imitation game-playing competition." 2020 15th Conference on Computer Science and Information Systems (FedCSIS). IEEE, 2020.

Motivation within this project:

This literature is one of the suggestions from the Template collection. This paper highlights various AI game competitions and notable participants who have achieved remarkable records. It provides guidance on the kind of previous work that this project can follow and reference, enabling more informed decisions during development. The goal is to create an AI that can learn and replicate the behavior of specific human players using their game records, as discussed in this literature.

Overview:

As stated in the literature, early milestones include games like checkers and chess, with AI systems such as IBM's Deep Blue defeating Garry Kasparov. Modern developments include AlphaGo with the game Go, AlphaStar with StarCraft, and OpenAI's Dota Five with the game Dota 2. The article also surveys major modern competitions regarding game-based AI:

1. General Game Playing (GGP) : Hosted by Stanford Logic Group since 2005, this competition challenges AI to play any finite deterministic synchronous games, even unknown ones. Game AIs have minimal preparation time and short decision times, reaching quarterfinals twice with Monte Carlo Tree Search algorithms using the Game Description Language (GDL).

2. General Video Game AI (GVG-AI) : Similar to GGP but focuses on video games using Atari computers and the Video Game Description Language (VGDL). It features fast-paced actions with the Rolling Horizon Evolutionary Algorithm.

3. Arimaa Challenge : Designed to be more challenging for computer agents than humans, it has a higher branching factor than chess. In 2015, a program named Sharp won the competition.

4. Starcraft AI : Aims to create a robust StarCraft bot capable of strategic and tactical reasoning, resource gathering, base building, and managing build orders. UAlbertaBot won in 2013, becoming a benchmark for new developers.

5. Visual Doom AI Competition (VizDoom) : AI platform based on the old third person shooter game called Doom. Bots are given with raw pixels instead of some form of structed state representation as in the case of other game AI competitions. The agents have to reason about the surroundings, navigate through the levels, fin the interesting sports and weapons and fight with opponents. This competition involves two tracks, one

being which AI finishes as the fastest and second being getting as much kills as possible. This competition involved with reinforcement learning however there is not AI agent that can beat human players.

6. Many more other competitions: Include Hearthstone AI, Strategy Card Game AI, Geometry Friends, Bot Bowl, Angry Birds Level Generation, and Generative Design in Minecraft, among others.

Project Relations and Critical evaluation:

The VizDoom competition has the most direct relevance to this project, given its similarities to the Super Mario Bros. environment. While Doom is a third-person shooter and Super Mario Bros. is a 2D platformer, both can use reinforcement learning. The goal of making the fastest stage clear in Super Mario Bros. parallels one of VizDoom's objectives. Researching VizDoom further can provide valuable insights and techniques applicable to this project and close the gaps. Other competitions mentioned in the paper offer general insights into AI methods and mediums but have less direct relevance. However, the variety of competitions highlights the flexibility and potential of different game environments and AI agents, allowing for broader exploration and selection in our project.

Second literature :

Justesen, Niels, Michael S. Debus, and Sebastian Risi. "When are we done with games?." 2019 IEEE Conference on Games (CoG). IEEE, 2019.

Motivation within this project:

This literature is the second reading suggestion from the template. As this paper provides a discussion on designing and evaluating fairness between human and AI competitions, it aligns well with the first literature review, which discussed various types of AI game competitions. Implementing fairness considerations from this paper can be particularly useful for our project, which involves comparing "Kane" and "Abel" ; two AI agents playing Super Mario Bros.

Overview:

This paper introduces rules that come into play when designing competitions between humans and AI. These rules aim to ensure fairness and can be categorized into six dimensions:

1. Perceptual: same input space
2. Motoric: same output space
3. Historic: spend the same amount of time on training
4. Knowledge: same access to declarative knowledge about the game
5. Compute: same computational power
6. Common-sense: same knowledge about all other things

The researchers concluded that "a completely fair competition can only be achieved against an artificial system that is essentially equivalent to a flesh and blood human." (Justesen, Niels, Michael S. Debus, and Sebastian Risi, 2019 , p.1)

The Black Box approaches:

This paper presents a practical approach for comparing AI to human intelligence in game competitions. For fair comparison, we treat AI systems as black boxes, ignoring their training, knowledge, and operational specifics.

Game Extrinsic and Intrinsic Factors

The paper also introduces extrinsic and intrinsic factors in the fairness of AI-human competition.

1. Game Extrinsic Factors**:**
   These factors involve added metagames that regulate competition structure and participant qualifications, such as ladder systems in online games and tournament formats in sports. These systems aim to minimize arbitrary advantages, ensuring that the best player wins. In eSports, gender segregation is a debated topic, reflecting ongoing discussions about fairness.

2. Game Intrinsic Factors:
   Intrinsic factors involve the mechanics and configurations within the game itself. Contemporary digital games often include various mechanical systems and configurations, such as different maps or character selections. These elements significantly impact strategies and outcomes, necessitating fair regulation in competitions.

Project Relations and Critical evaluation:

This literature is invaluable for our project as it offers a comprehensive framework for ensuring fairness in AI competitions. By applying the six dimensions of fairness ; perceptual, motoric, historic, knowledge, compute, and common-sense, we can better structure the competition between Kane and Abel. Moreover, the black box approach is particularly relevant for evaluating our AI systems impartially, focusing on performance rather than underlying methodologies.

The insights on game intrinsic factors provide additional layers of fairness and balance that are crucial for competitive AI development. By regulating both external competition structures and internal game mechanics, we can create a more robust and fair evaluation environment for our AI agents.

Overall, this literature fills a critical gap in our project by addressing the fairness of AI competitions, ensuring that our approach to developing and evaluating Kane and Abel is grounded in well-established principles. This enhances the validity and reliability of our comparative analysis and contributes significantly to the field of AI game development.

Third literature :

Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.

Motivation within this project:

This literature is the third reading suggestion from the template. It provides comprehensive information about Deep Q-Networks (DQN) and reinforcement learning, detailing experiment processes and outcomes. Given that our project aims to build an AI for Super Mario Bros using reinforcement learning for the second AI (Abel), this paper's insights could be highly valuable. If critically convincing, the methods discussed in this paper could be adapted to enhance the training model for Abel.

Overview:

Reinforcement Learning vs. DQN:

The paper claims that using a non-linear function approximator, like a neural network or DQN, to represent the action-value (Q) function can cause reinforcement learning to become unstable or even diverge.

DQN addresses this issue through two primary mechanisms:

1. Experience Replay: This mechanism randomizes over the data, removing correlations in the observation sequence and smoothing over changes in the data distribution.

2. Iterative Update: This adjusts action values towards target values that are only periodically updated, reducing correlations with the target.

Experimental Results:

1. The paper presents a comparison of DQN with existing reinforcement learning methods using various Atari games. The results indicate that DQN outperforms the best existing reinforcement learning methods on 43 out of 49 games without incorporating additional prior knowledge about the Atari 2600 games.
2. The DQN agent performed at a level comparable to a professional human games tester, achieving more than 75% of the human score on more than half of the games.

Significance of DQN:

1. The DQN approach is notable for its success without requiring game-specific adjustments or feature engineering. It uses a single architecture that learns directly

from raw pixel inputs and game scores, making it a versatile and robust solution for a variety of games.

Project Relations and Critical evaluation:

This paper offers significant insights into the DQN agent, which might outperform the reinforcement learning methods we initially considered for our project. Although the paper was published in 2015 and AI has evolved considerably since then, the foundational concepts introduced in DQN remain influential. More recent advancements have built upon these concepts, potentially leading to even more effective algorithms.

For our project, the techniques discussed in this paper are directly applicable. Experience replays and iterative update mechanisms could enhance the stability and performance of our reinforcement learning-based AI, Abel. Additionally, understanding the limitations and strengths of DQN helps us design more robust training processes and evaluation criteria. However, we plan to use Proximal Policy Optimization (PPO) rather than DQN due to its more recent advancements and suitability for continuous action spaces and provides better convergence/performance rate, which are more relevant to our game environment.

The insights from this literature are particularly relevant for developing Abel. The mechanisms of experience replay, and iterative updates could be adapted within the PPO framework to improve stability and performance. Furthermore, the paper's emphasis on a single architecture that learns from raw inputs aligns with our goal of creating a versatile AI capable of handling various challenges within Super Mario Bros.

Moreover, the comparison between DQN and traditional reinforcement learning methods provides a benchmark for evaluating our AI's performance. By understanding the strengths of DQN, we can set more realistic performance goals for Abel and identify potential areas for improvement.

In conclusion, this literature not only highlights the potential advantages of advanced reinforcement learning methods over traditional ones but also provides practical insights that can be adapted to our project. This contributes significantly to our understanding of AI game learning and offers a robust foundation for enhancing our AI's capabilities in playing Super Mario Bros. By integrating these insights with PPO, we aim to create a highly effective and stable AI.

Fourth literature :

ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning

Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek & Wojciech Ja´skowski

Motivation within this project:

As we can see from the First Literature the paper on ViZDoom is highly relevant to our project because the AI agent developed for the game DOOM shares many similarities with the AI we aim to create for Super Mario Bros. Both games involve navigating through an environment, making strategic decisions, and overcoming obstacles. While DOOM is a 3D first-person shooter and Super Mario Bros is a 2D platformer, the underlying principles of visual learning and reinforcement learning are applicable to both. This literature provides valuable insights into reinforcement learning test beds, which can inform our approach to training AI in a game context with potentially fewer obstacles due to the simpler 2D environment of Super Mario Bros.

Overview:

The ViZDoom paper describes the development and utilization of a reinforcement learning test bed for the game DOOM, leveraging recent advancements in GPU technology and 3D environments to enhance visual learning in AI agents using neural networks. The paper outlines the success of deep architectures in reinforcement learning through experiments with AI agents playing the pixel-based game Atari 2600 using raw pixel data. The choice of DOOM as a domain is justified by several factors:

1. Open Source: The game is freely available for modifications.
2. Lightweight: It runs efficiently without heavy resource demands.
3. No Bottlenecks: The game's architecture does not impose significant limitations on performance.
4. Total Control: Developers have complete control over the game environment.
5. Customizable Resolution: The pixel resolution can be adjusted for various experiments.
6. Multiplayer Abilities: The game supports multiplayer scenarios, offering diverse testing environments.

The paper highlights the unique feature of DOOM's software renderer, which allows the game to run without a desktop environment and access the screen buffer directly, facilitating faster and more efficient learning.

ViZDoom API's features include:

1. Control Modes: Various player modes to simulate different scenarios.
2. Scenarios: The ability to run custom scenarios for targeted learning.

3. Depth Buffer Access: Access to the renderer’s depth buffer for more detailed analysis.
4. Off-screen Rendering and Frame Skipping: Supports heavy machine learning experiments by rendering frames off-screen and skipping frames to speed up learning.

Experiments detailed in the paper show that:

1. Frame Skipping: Increased frame skipping leads to faster learning.
2. MedKit Collecting Experiment: In a scenario with random drops and obstacles, the AI learned to navigate and collect MedKits efficiently after 1,000,000 steps, demonstrating policy consistency similar to human players.

Project relations and Critical Evaluation

The ViZDoom paper offers valuable insights and methodologies relevant to developing AI agents for Super Mario Bros. Despite the different game genres, the discussed reinforcement learning techniques and technical implementations are highly applicable.

Reinforcement Learning Framework: The framework used in ViZDoom, particularly the successful application of deep Q-learning, can be adapted to our project. This aligns well with our plan to use PPO for training Abel, our reinforcement learning-based AI agent.

Technical Implementations: Using a software renderer to access the game’s screen buffer efficiently is a technique we can adopt. This ensures our AI agents can interact with the game environment with minimal overhead, facilitating faster learning and more responsive gameplay.

Experimentation and Results: The experiments with frame skipping and MedKit collection demonstrate techniques to accelerate the learning process. We can apply frame skipping in our Super Mario Bros environment to improve the learning speed of our AI agents.

Adaptation to 2D Environment: While DOOM is a 3D game, the principles of reinforcement learning and AI development are adaptable to a 2D environment like Super Mario Bros. The simpler 2D context may reduce complexities, allowing us to optimize learning algorithms and achieve human-level performance.

Project Design

Project overview

This project involves developing two AI systems to play Super Mario Bros, based on the provided AI Game Development Template. The first AI system, named Kane, will have pre-programmed behavior using techniques such as a finite state machine or other appropriate methods developed in-house. The second AI system, named Abel, will utilize statistical machine learning techniques, specifically reinforcement learning, developed with external sources. This design documentation will detail the project plan, expected outcomes, and any necessary adjustments that may arise and will be discussed in the final term submission.

Template Used

As stated in Introduction of this Preliminary report the CM3002: Kane and Abel: AIs that play game template was used for this project, which includes sections for the project overview, domain and users, justification of design choices, project structure, key technologies, work plan, and evaluation plan. This structured approach ensures comprehensive coverage of all critical aspects of the project.

Domain and users

The domain of this project is game AI development, specifically focusing on two AI with different development cycle and having comparison of results. The primary users for this project would be those two AI that is getting developed and myself who is developing those as well as other researchers and developers who are in field of Artificial Intelligence. Here I am choosing Super Mario Bros as the game environment ensuring that the project remains relevant to the literature review above and having sample resources for developing and testing.

Justification

1. Popularity and Relevance: It is a well-known and widely studied game, making it an ideal platform for AI research due to its complexity and the availability of tools and past research.

2. Existing Tools and Research: Using this game allows us to leverage existing tools and research, making it easier to develop and test the AI systems. The availability of the gym-super-mario-bros library provides a ready-made environment for AI experimentation.

3. Resource Availability: The popularity of the game ensures that there is ample support and resources, including documentation, community contributions, and past studies, which can aid in the development and testing process.

Overall Structure

The project is structured into several key components, each addressing different aspects of the development and evaluation process:

1. Environment Setup :
    a. Setting up the game environment using gym-super-mario-bros, an OpenAI Gym environment for Super Mario Bros on the Nintendo Entertainment System (NES) using the nes-py emulator. This setup is primarily for developing Abel, the reinforcement learning-based AI.

    b. For Kane, the preprogrammed AI, development will involve creating specific behavior patterns using techniques such as finite state machines or genetic algorithms.

2. AI Development :
    a. Kane(The first AI) : Developing a preprogrammed pattern for an example genetic algorithms with neural networks.

    b. Abel(The second AI) : Developing and implementing a reinforcement learning based AI using Stable baselines library(external sources).

Abel will be developed first since it is one of the most common way to develop an AI with reinforcement learning alongside with the Open AI gym environment. Kane will be developed after Abel is completed since there are many different ways to develop this. Current option for Kane is developing with genetic algorithm with neural networks however it will have more clear picture after Abel.

3. Training and Testing.
    a. Training Abel using reinforcement learning techniques and will be overviewed with TensorBoard.
    b. Kane will have same settings(for an example , steps sizes) and will be trained with same amount of time.

4. Evaluation and Analysis
    a. We will after analyze the result by comparing two AI's time and scores of ech agent , we will make changes to variables to see what variables can make those AIs to perform better.

Key Technologies

1. Programming Language: Python
    a. Python is chosen for its versatility, ease of use, and extensive support for AI and machine learning libraries.

2. Game Environment: gym-super-mario-bros using the nes-py emulator
    a. This library provides a Python interface for interacting with the Super Mario Bros game, making it easy to integrate with AI algorithms and tools.

3. Reinforcement Learning Library: Stable Baselines
    a. Stable Baselines is a set of improved implementations of reinforcement learning algorithms based on OpenAI Baselines. It provides a variety of RL algorithms, such as PPO (Proximal Policy Optimization) and DQN (Deep Q-Network), which are suitable for training game-playing agents.

4. Machine Learning Framework: PyTorch or TensorFlow
    a. Depending on the specific implementation of Stable Baselines, either PyTorch or TensorFlow will be used as the underlying machine learning framework. Both frameworks are widely used in the AI community and provide robust support for deep learning and reinforcement learning.
5. Preprocessing Techniques: Grayscale conversion
    a. Converting the game screen to grayscale helps in detecting subtle changes and allows the RL algorithms to focus on important features, improving the learning process.
6. Analysis Tool: TensorBoard
    a. TensorBoard will be used to monitor the training process and visualize key metrics, providing insights into the learning progress and performance of the AI systems.

Work Plan

The work plan outlines the major tasks and their timelines, providing a visual representation of the project schedule. The following Gantt chart illustrates the work plan:

🟩	no obstacle, should process successfully in time	Gannt Chart for Project template CM3002									
🟥	obstacle occurred, unexpected, required more time	First half of term: Week									
		1	2	3	4	5	6	7	8	9	10
Current Tasks	Set up development environment	🟩	🟩								
	Choosing template	🟩	🟩								
	Install necessary libs and dependencies		🟩			🟩					
	Literature review, research		🟩	🟩	🟩						
	Familize with Open AI gym and NES					🟩					
	Develop and test the reinforcement AI Abel					🟥	🟩	🟩	🟩		
	Project Design							🟩	🟩		
	Implement the basic behaviour : Abel							🟥	🟩	🟩	
	Obseve statistics with tensorboard : Abel								🟩	🟩	
	Prototype Document									🟩	🟩
	Experiments with variables : Abel										
Future Tasks	Develop and test the FSM	🟩	🟩								
	Implement the basic behaviour : Kane		🟩	🟩	🟩	🟩	🟩				
	Observe statistics : Kane						🟩	🟩			
	Experiments with variables : Kane										
	Compare results of both AI										
	Analysis and get documents and demo for Final submission										
Weekly log and comments as project processes	Week 1 - 2	No delay and no obstacle setting up environment : Anaconda, Jupyter ; Choosing template as CM3002 Kane and Abel: Ais playing game									
	Week 2 - 3	No delay and no obstacle setting up environment, Started literatures with suggested ones from templates, writing reviews and researching									
	Week 3 - 4	Kept doing the literature review on suggested literatures and making project proposal video, as well as researching about more literature									
	Week 4 - 5	Dependency error occurred, couldn’t correctly implement Stable baselines since it was not installed correctly. Using virtual environment to experiement more. Kept researching and writing reviews on literatures									
	Week 5 - 6	Started on project desgin section on document, keep on developing Abel									
	Week 6 - 7	Another error occurred on tensorflow, had to remove all and reinstall again.									
	Week 7 - 8	implementing basic behaviours like how far can mario go in screen, making training samples and see if it progresses.									
	Week 8 - 9	Observe statistics with tensorboard with all the trainings, checking to see if positive results are there									
	Week 9 -10	Logging on protoype section of document with all the results from codes for Abel									
	Week 10 - 11										

Second half of the Gannt chart for future plans

		Second Half of term : Week									
		11	12	13	14	15	16	17	18	19	20
Future Tasks	Develop and test the genetic algorithm, neural networkAI	■	■	■	■						
	Implement the basic behaviour : Kane			■	■	■	■				
	Observe statistics : Kane					■	■				
	Experiments with variables : Kane						■	■			
	Compare results of both AI							■	■	■	
	Analysis and get documents and demo for Final submission							■	■	■	■

Evaluation Plan

The evaluation plan details the methods and criteria to ensure the project meets its objectives and provides valuable insights. The plan includes performance metrics, evaluation methods, and tools to be used.

Performance Metrics:

1. Score: Total points accumulated by the AI during gameplay. Higher scores indicate better performance.

2. Completion Time: Time taken to complete a level. Faster completion times indicate more efficient play.

3. Survival Rate: Percentage of levels completed without losing all lives. Higher survival rates indicate better AI robustness.

4. Loss and explained variance: Monitored via TensorBoard for Abel and separate statistical method to observe this for Kane is needed. A progressive reduction in loss and increase in explained variance indicates effective learning and training of models.

Evaluation Methods:

1. Quantitative Analysis:
    a. For Kane: Collect and analyze performance data focusing on scores, completion times, survival rates, and consistency.
    b. For Abel: In addition to performance data, monitor and analyze the training loss using TensorBoard. A progressive decrease in loss values indicates the reinforcement learning model's convergence and effectiveness.

2. Qualitative Analysis: Observe and document the behaviors and strategies used by both AIs during gameplay. Identify notable patterns or differences in their approaches.

3. Comparative Analysis: Compare the performance of Kane and Abel across multiple trials, highlighting the strengths and weaknesses of each approach.

Tools and Techniques:

1. Data Collection: Use logging and monitoring tools to collect performance data during gameplay for both AIs. Ensure data is recorded consistently and accurately.

2. Data Visualization: Use TensorBoard to monitor Abel's training process and visualize the reduction in loss over time. Create charts and graphs to illustrate performance trends and comparisons between Kane and Abel using matplotlib library.

3. Statistical Analysis: Apply statistical methods to analyze the collected data, focusing on key metrics and identifying significant differences in performance between the two AI systems.

Implementation – Abel

This section of document outlines the prototype development of Abel: Reinforcement Learning application for the NES game Super Mario Bros, using the OpenAI Gym framework and stable-baseline3 library and PPO. With this Abel aims to achieve autonomous playing Super Mario Bros. Kane; AI that is going to be preprogrammed will be developed after satisfactory development of Abel, currently Abel's development is completed as below implementation specifications yet we reached to a level where Abel can learn and clear its first stage of the game within an hour, details of testing out and experiments and processes are recorded in video. These are overall important details that I have used for building Abel.

Setup and Dependencies

First, we are ensuring that all necessary dependencies are installed. This included OpenAI Gym, the NES emulator and stable abselines3 for RL(reinforcement learning)

```
%pip install gym_super_mario_bros==7.3.0 nes_py
%pip install stable-baselines3[extra] --ignore-installed TBB
%pip install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu121
%pip install gym matplotlib
```

As you can see I do have specified the versions and downloads to make it compatible with my local environment.

Environment Setup

```
import gym_super_mario_bros
from nes_py.wrappers import JoypadSpace
from gym_super_mario_bros.actions import SIMPLE_MOVEMENT

env = gym_super_mario_bros.make('SuperMarioBros-v0',
apply_api_compatibility=True, render_mode="human")
env = JoypadSpace(env, SIMPLE_MOVEMENT)
```

In here we are importing the game, wrapping, and importing JoypadSpace which gives action abilities (Keys to press). As well as SIMPLE_MOVEMENT, which lower down the keybindings from 256 to 7.

```
SIMPLE_MOVEMENT
```

```
[['NOOP'],
 ['right'],
 ['right', 'A'],
 ['right', 'B'],
 ['right', 'A', 'B'],
 ['A'],
 ['left']]
```

Pre - Processing

In Pre-Processing stage we are turning the colored frames to something that is simpler for computer to process

```
from gym.wrappers import GrayScaleObservation
from stable_baselines3.common.vec_env.vec_frame_stack import VecFrameStack
from stable_baselines3.common.vec_env.dummy_vec_env import DummyVecEnv

env = GrayScaleObservation(env, keep_dim=True)
env = DummyVecEnv([lambda: env])
env = VecFrameStack(env, 4, channels_order='last')
```

`GrayScaleObservation` is a wrapper from OpenAI Gym that converts the environment's observations (frames) to grayscale.

```
env = GrayScaleObservation(env, keep_dim=True)
```

converts the color frames of the game to grayscale. And reduces the complexity of the input data by removing color information which is unnecessary for training an RL(Reinforcement Learning) agent. And it ensures that the number of dimensions in the observation space remains the same. This means that even though the color channels are removed, the observation shape is maintained.

`VecFrameStack` is a wrapper from stable-baselines3 that stacks multiple frames together to create a single observation. This is useful for capturing temporal information.

```
env = DummyVecEnv([lambda: env])
```

This line wraps the grayscale environment into a ‘DummyVecEnv’ that is needed for stable baseline 3 to work. It allows parallel environments.

`DummyVecEnv` is a simple vectorized environment wrapper that allows us to use environments in a way compatible with stable-baselines3's algorithms.

```
env = VecFrameStack(env, 4, channels_order='last')
```

This line wraps the vectorized environment in a ‘VecFrameStack’ with stacking a specified number of consecutive frames together to form the ovservation space.

Model Training

```
import os
from stable_baselines3 import PPO
from stable_baselines3.common.callbacks import BaseCallback

class TrainAndLoggingCallback(BaseCallback):
    def __init__(self, check_freq, save_path, verbose=1):
        super(TrainAndLoggingCallback, self).__init__(verbose)
        self.check_freq = check_freq
        self.save_path = save_path

    def _init_callback(self):
        if self.save_path is not None:
            os.makedirs(self.save_path, exist_ok=True)

    def _on_step(self):
        if self.n_calls % self.check_freq == 0:
            model_path = os.path.join(self.save_path,
'test_best_model_{}'.format(self.n_calls))
            self.model.save(model_path)
        return True

CHECKPOINT_DIR = './train/'
LOG_DIR = './logs/'
callback = TrainAndLoggingCallback(check_freq=10000, save_path=CHECKPOINT_DIR)

model = PPO('CnnPolicy', env, verbose=1, tensorboard_log=LOG_DIR,
learning_rate=0.000001, n_steps=512)
model.learn(total_timesteps=1000000, callback=callback)
```

In model training we are employing PPO(Proximal Policy Optimization) algorithm from stable-baselines3 to train the RL agent as well as we are using call back mechanism to save the model at regular intervals through out the game.

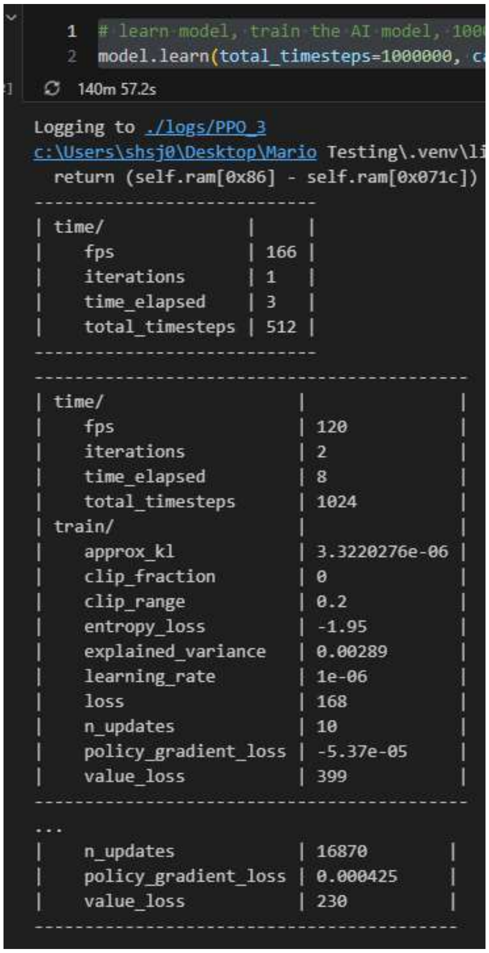
In conclusion above codes demonstrates the base touch of the flow of Abel, the details and full code running is available with the prototype video.

Evaluation

Effective preprocessing: Grayscale conversion and frame stacking are appropriate for capturing game dynamics without unnecessary color information, enhancing training efficiency

Algorithm suitability: As mentioned in literature review PPO is robust Reinforcement learning algorithm suitable for discrete action spaces and capable of handling complex environments like Super Mario Bros.

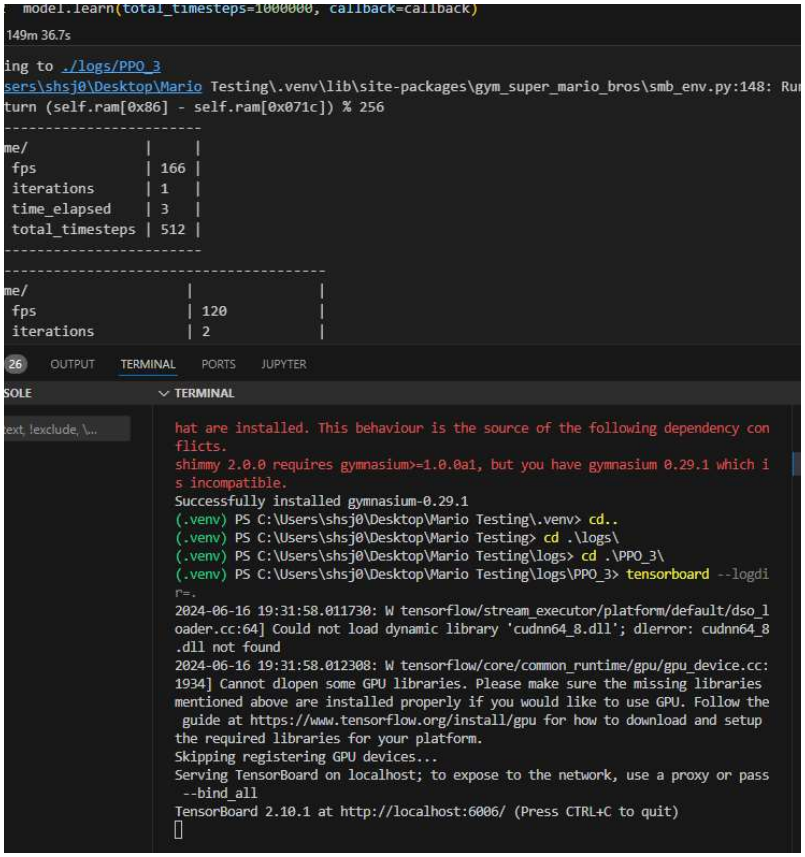
As we progress with the project and continuously monitor the statistics, the loss keeps decreasing over each cycle of the game, and the explained variance is increasing. Additionally, we can observe graphs in TensorBoard that show how the model is performing.



```
# learn model, train the AI model, 100
model.learn(total_timesteps=1000000, c
```

140m 57.2s

```
Logging to ./logs/PPO_3
c:\Users\shsj0\Desktop\Mario Testing\.venv\li
  return (self.ram[0x86] - self.ram[0x071c])
----------------------------
| time/              |     |
|    fps             | 166 |
|    iterations      | 1   |
|    time_elapsed    | 3   |
|    total_timesteps | 512 |
----------------------------
------------------------------------------
| time/                   |              |
|    fps                  | 120          |
|    iterations           | 2            |
|    time_elapsed         | 8            |
|    total_timesteps      | 1024         |
| train/                  |              |
|    approx_kl            | 3.3220276e-06 |
|    clip_fraction        | 0            |
|    clip_range           | 0.2          |
|    entropy_loss         | -1.95        |
|    explained_variance   | 0.00289      |
|    learning_rate        | 1e-06        |
|    loss                 | 168          |
|    n_updates            | 10           |
|    policy_gradient_loss | -5.37e-05    |
|    value_loss           | 399          |
------------------------------------------
...
|    n_updates            | 16870       |
|    policy_gradient_loss | 0.000425    |
|    value_loss           | 230         |
------------------------------------------
```

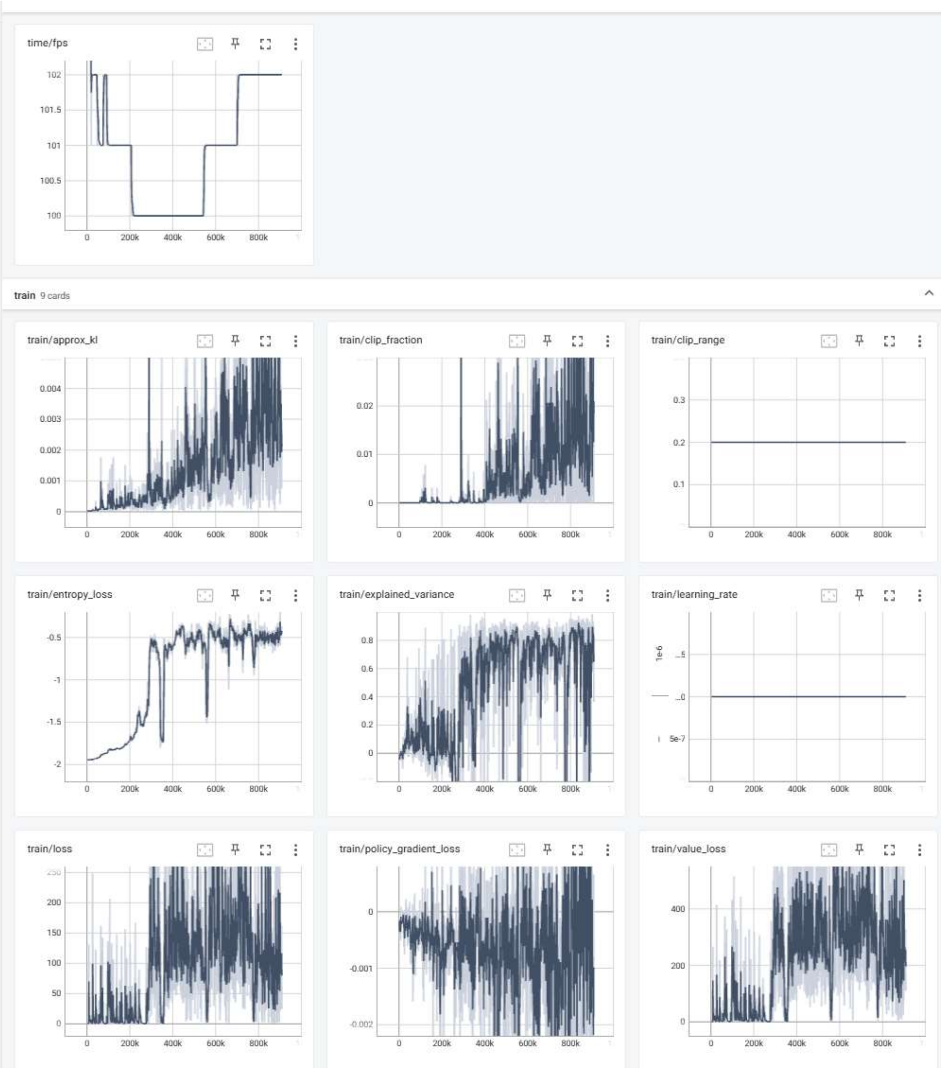


```
  model.learn(total_timesteps=1000000, callback=callback)
```

149m 36.7s

```
ing to ./logs/PPO_3
sers\shsj0\Desktop\Mario Testing\.venv\lib\site-packages\gym_super_mario_bros\smb_env.py:148: Ru
turn (self.ram[0x86] - self.ram[0x071c]) % 256
------------------------
me/               |     |
 fps              | 166 |
 iterations       | 1   |
 time_elapsed     | 3   |
 total_timesteps  | 512 |
------------------------
------------------------------------
me/                    |          |
 fps                   | 120      |
 iterations            | 2        |
```

```
hat are installed. This behaviour is the source of the following dependency con
flicts.
shimmy 2.0.0 requires gymnasium>=1.0.0a1, but you have gymnasium 0.29.1 which i
s incompatible.
Successfully installed gymnasium-0.29.1
(.venv) PS C:\Users\shsj0\Desktop\Mario Testing\.venv> cd..
(.venv) PS C:\Users\shsj0\Desktop\Mario Testing> cd .\logs\
(.venv) PS C:\Users\shsj0\Desktop\Mario Testing\logs> cd .\PPO_3\
(.venv) PS C:\Users\shsj0\Desktop\Mario Testing\logs\PPO_3> tensorboard --logdi
r=.
2024-06-16 19:31:58.011730: W tensorflow/stream_executor/platform/default/dso_l
oader.cc:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8
.dll not found
2024-06-16 19:31:58.012308: W tensorflow/core/common_runtime/gpu/gpu_device.cc:
1934] Cannot dlopen some GPU libraries. Please make sure the missing libraries
mentioned above are installed properly if you would like to use GPU. Follow the
 guide at https://www.tensorflow.org/install/gpu for how to download and setup
the required libraries for your platform.
Skipping registering GPU devices...
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass
 --bind_all
TensorBoard 2.10.1 at http://localhost:6006/ (Press CTRL+C to quit)
```



After an hour of training, there was a successful record of Mario beating the first stage of the game.

Challenges and Considerations:

Logging: It is crucial to log all the history of AI agent Abel's training progress. This data will be essential for comparing Abel's performance with Kane (a future development of a preprogrammed AI agent). Abel and Kane should have identical settings, including training time, environment setup, number of training steps, and more.

Debugging: The Python Super Mario engine occasionally freezes its frames, which is a major issue that could potentially slow down Abel's training cycles. This issue might also contribute to the spikes observed in the graphs on tensorboard with explained_variance and train/loss. We anticipate that resolving this issue will be a priority as we approach the end of this term.

Implementation – Kane

This section outlines the prototype development of Kane, a self-made AI application for the NES game Super Mario Bros. We will use the OpenAI Gym framework but will not use other pre-implemented AI methods. Instead, we will use the Finite State Machine (FSM) learning method. The development of Kane is ongoing, and this document is a draft report that will be continuously updated as the project progresses.

Firstly, we will use the OpenAI Super Mario Gym environment because it has the best compatibility for creating AI for the game.

Setup:

```
import gym_super_mario_bros
from nes_py.wrappers import JoypadSpace
from gym_super_mario_bros.actions import SIMPLE_MOVEMENT
from gym.wrappers import TimeLimit
import numpy as np
import random
import matplotlib.pyplot as plt

# Reset method for JoypadSpace
JoypadSpace.reset = lambda self, **kwargs: self.env.reset(**kwargs)

# Create the environment
env = gym_super_mario_bros.make('SuperMarioBros-v0',
apply_api_compatibility=True, render_mode="human")
env = JoypadSpace(env, SIMPLE_MOVEMENT)
env = TimeLimit(env, max_episode_steps=5000)

# Confirm the action space and observation space
print("Action space:", env.action_space)
print("Observation space shape:", env.observation_space.shape)
print("Available actions:", SIMPLE_MOVEMENT)
```

We will run these setups to see where we are starting with Kane. Similar to Abel, we will have SIMPLE_MOVEMENT wrapped with JoypadSpace since we still want to minimize the keys and buttons pressed by AI to 7 keys.

Preprocessing:

Then we define the FSM:

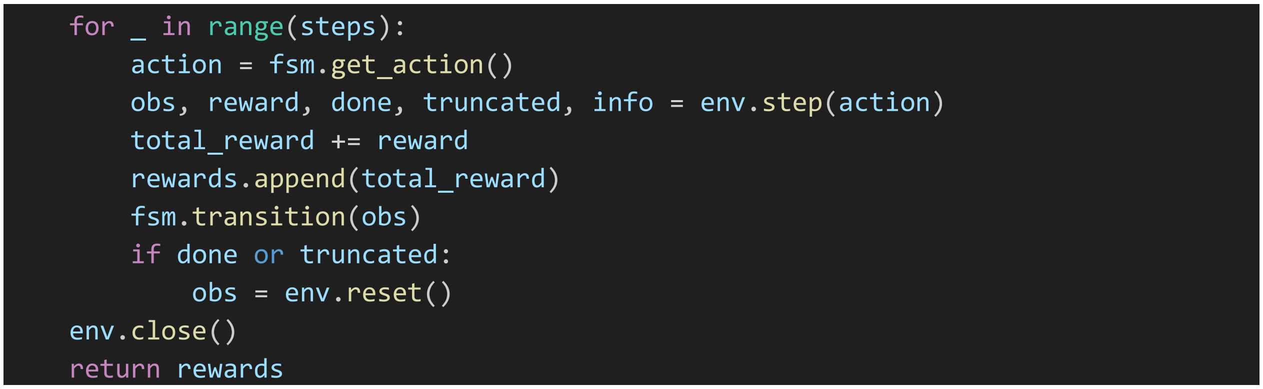
```
class MarioFSM:
    def __init__(self):
        self.state = "START"
        self.transition_matrix = {
            "START": ["MOVE_RIGHT", "JUMP"],
            "MOVE_RIGHT": ["JUMP", "RUN", "STOP"],
            "JUMP": ["MOVE_RIGHT", "LAND"],
            "RUN": ["JUMP", "STOP", "MOVE_RIGHT"],
            "LAND": ["MOVE_RIGHT", "STOP"],
            "STOP": ["START"]
        }
```

The __init__ method initializes the FSM with a starting state of “START”. The matrix defines possible state changes, showcasing which state can be reached after a certain initial state. For example, Mario can either “MOVE_RIGHT” or “JUMP” from “START”.

```
   def transition(self, observation):
        if self.state == "START":
            next_state = "MOVE_RIGHT"
        elif self.state == "MOVE_RIGHT":
            if random.random() > 0.5:
                next_state = "JUMP"
            else:
                next_state = "RUN"
        elif self.state == "JUMP":
            next_state = "LAND"
        elif self.state == "RUN":
            next_state = "MOVE_RIGHT"
        elif self.state == "LAND":
            next_state = "MOVE_RIGHT"
        elif self.state == "STOP":
            next_state = "START"
        self.state = next_state
        return self.state
```

We determine simple transition logic for the AI with a random parameter, where the AI probabilistically decides to "JUMP" or "RUN".

```
# Run the environment with FSM-based control
def run_fsm_env(steps=100000):
    obs = env.reset()
    total_reward = 0
    rewards = []
```

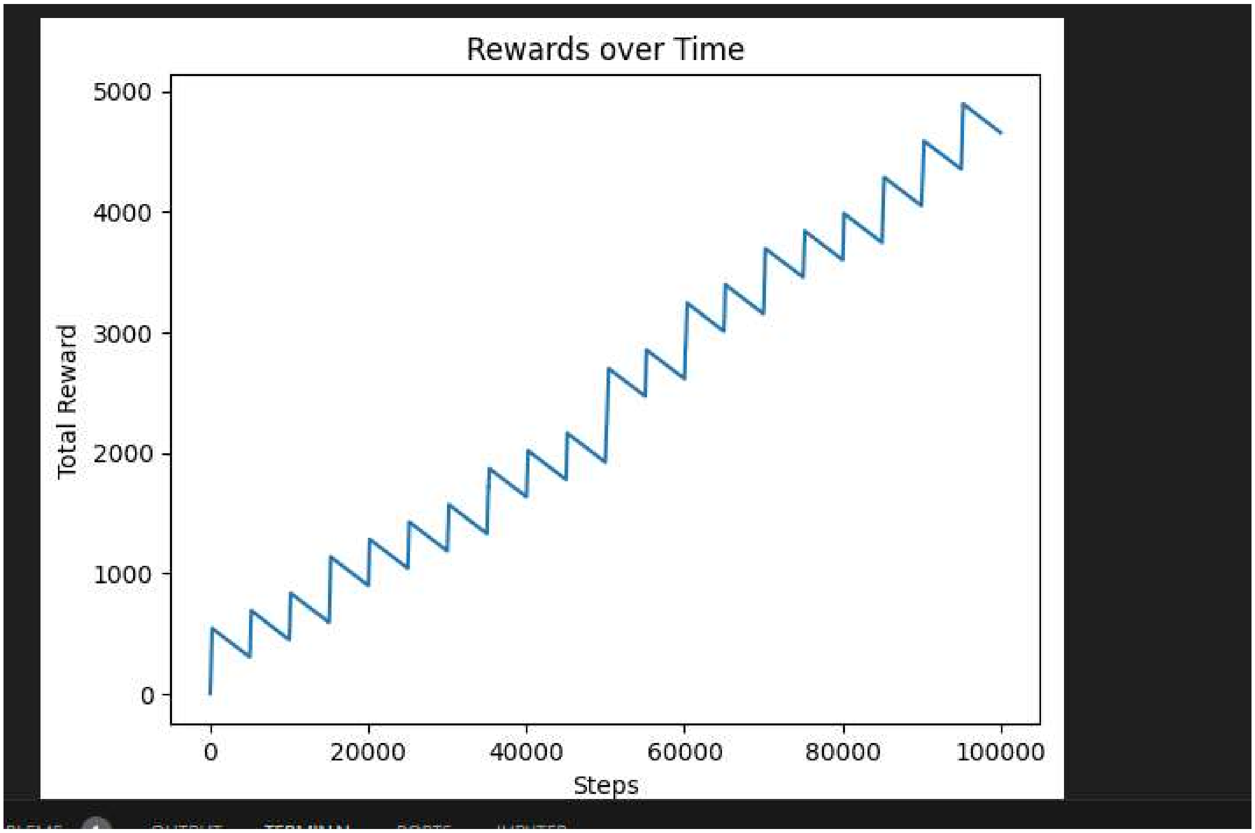


```
    for _ in range(steps):
        action = fsm.get_action()
        obs, reward, done, truncated, info = env.step(action)
        total_reward += reward
        rewards.append(total_reward)
        fsm.transition(obs)
        if done or truncated:
            obs = env.reset()
    env.close()
    return rewards
```

We will run the environment with:

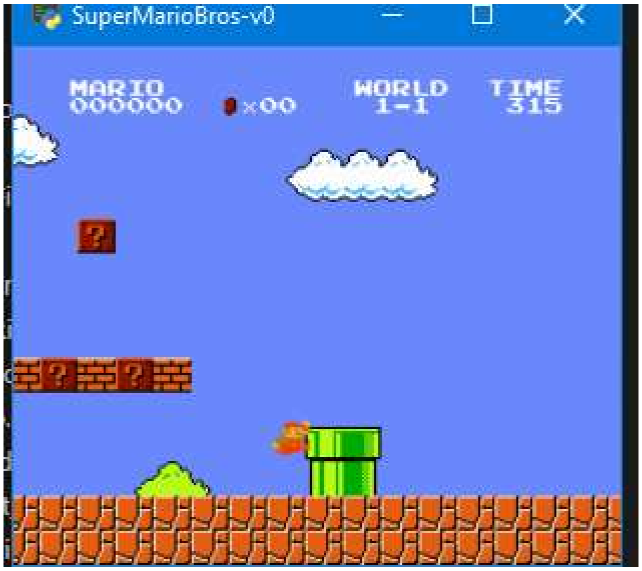
- env.reset(): Resets the environment.
- fsm.get_action(): Retrieves the action based on the current state.
- step(action): Executes a number of steps in the game (the more steps, the longer it runs).
- fsm.transition(obs): Updates the FSM's state based on the new observation.
- reset(): If the episode ends (done or truncated), the environment is reset.

The rest of the code belongs to statistics after testing.



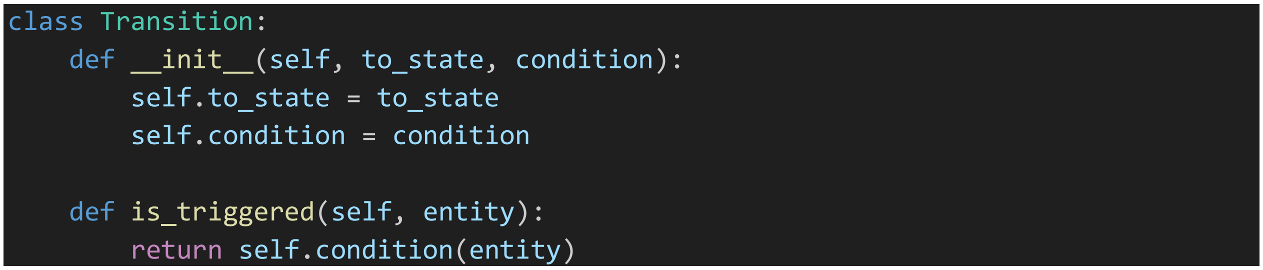
As we can see from the graph, the AI shows very repetitive progress. However, compared to Abel, Kane has a very slow learning process. Even after an hour, it was not able to finish the first stage.

Debugging:



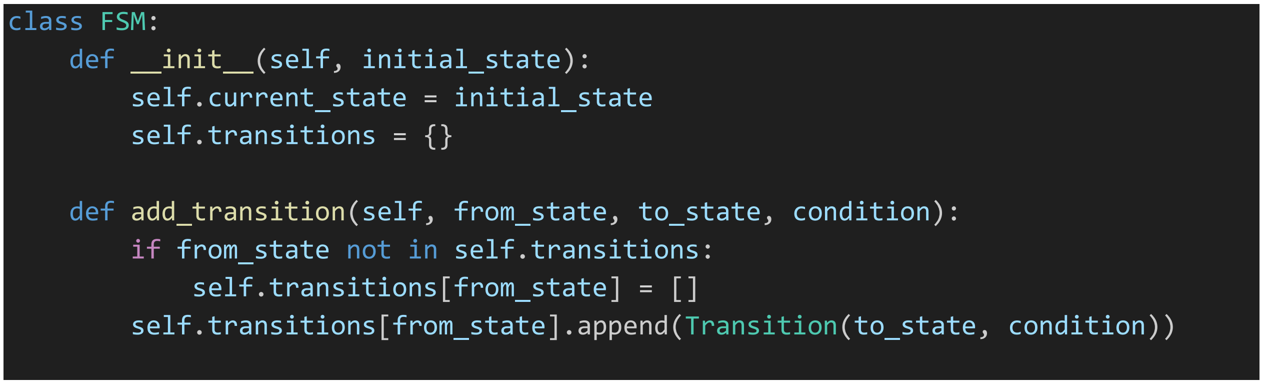
Most of the time, the Mario agent gets stuck at the first or second pole and never makes extensive progress like Abel. This slow progress might be due to the simple configuration or other issues.

Therefore, I came up with an idea to debug the codes for states and transitions. For simplicity, I changed the conditions in transitions to a True and False-based system to check the condition.

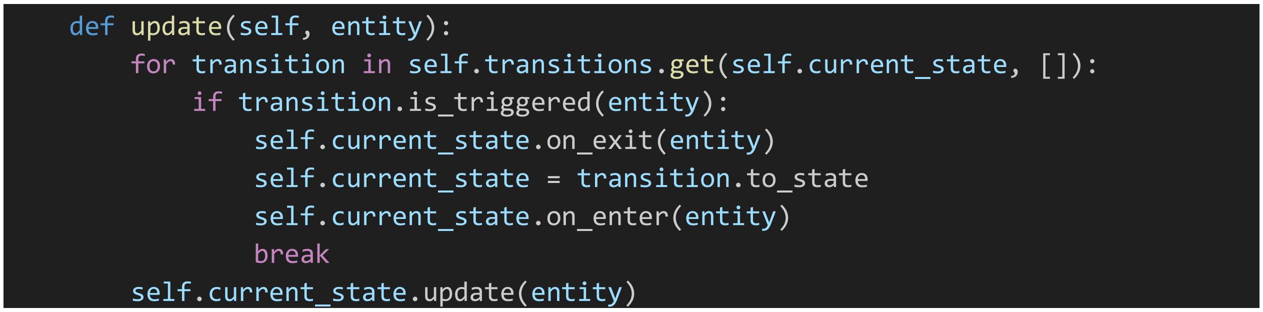


```
class Transition:
    def __init__(self, to_state, condition):
        self.to_state = to_state
        self.condition = condition

    def is_triggered(self, entity):
        return self.condition(entity)
```

In the FSM class, I added a method to add new transitions from one state to another given a condition. The update method checks if any transitions should be triggered and updates the state of the entity accordingly.

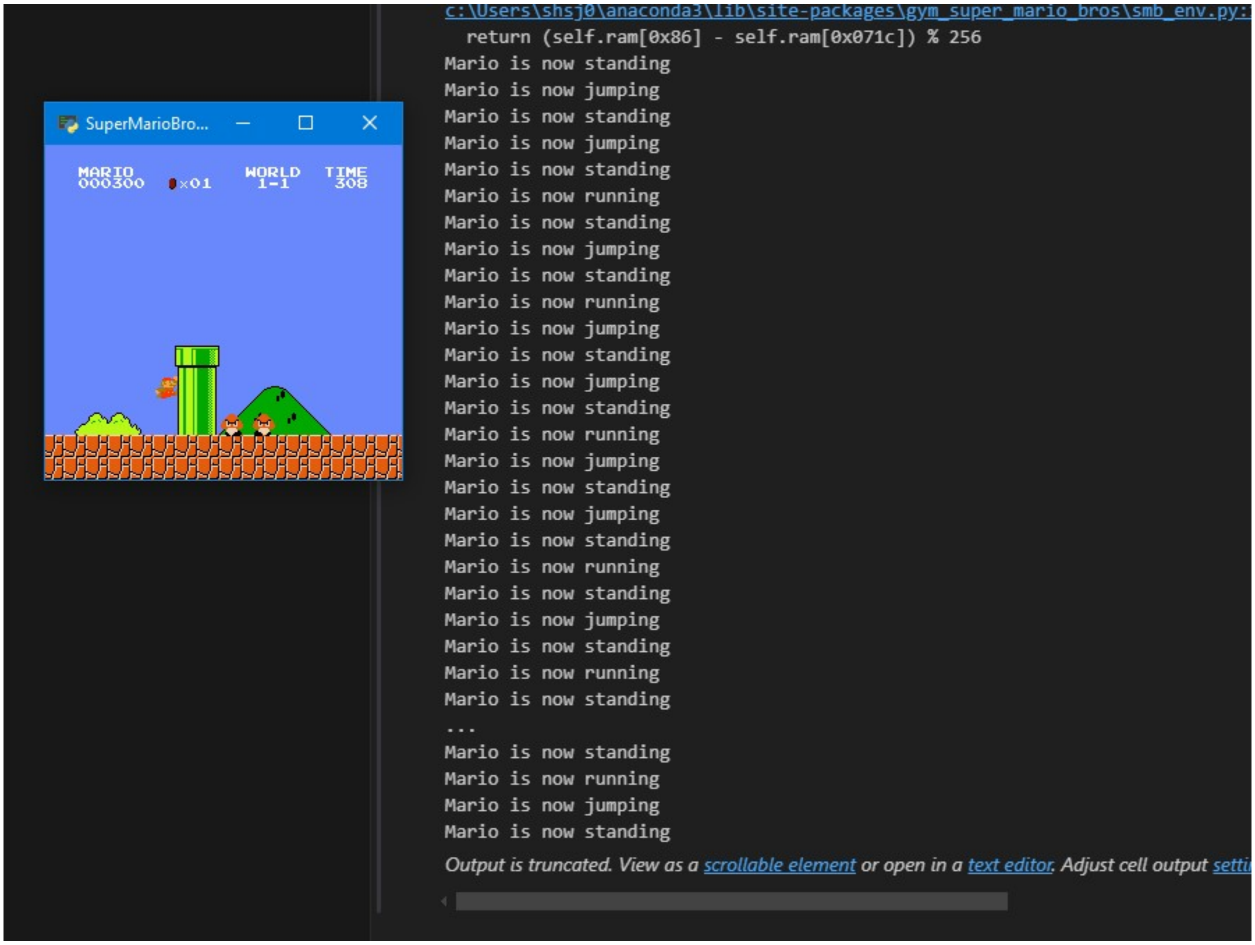


```
class FSM:
    def __init__(self, initial_state):
        self.current_state = initial_state
        self.transitions = {}

    def add_transition(self, from_state, to_state, condition):
        if from_state not in self.transitions:
            self.transitions[from_state] = []
        self.transitions[from_state].append(Transition(to_state, condition))
```



```
    def update(self, entity):
        for transition in self.transitions.get(self.current_state, []):
            if transition.is_triggered(entity):
                self.current_state.on_exit(entity)
                self.current_state = transition.to_state
                self.current_state.on_enter(entity)
                break
        self.current_state.update(entity)
```

The update method of the FSM is called regularly with every game iteration. The FSM checks if any transition should be triggered based on the current state and conditions. If a transition is triggered, the FSM handles the state change by calling the on_exit method of the current state, switching to the new state, and calling the on_enter method of the new state. We also implemented matplotlib to showcase training loss and entropy loss to compare with Abel's statistics.



```
c:\Users\shsj0\anaconda3\lib\site-packages\gym_super_mario_bros\smb_env.py:
  return (self.ram[0x86] - self.ram[0x071c]) % 256
Mario is now standing
Mario is now jumping
Mario is now standing
Mario is now jumping
Mario is now standing
Mario is now running
Mario is now standing
Mario is now jumping
Mario is now standing
Mario is now running
Mario is now jumping
Mario is now standing
Mario is now jumping
Mario is now standing
Mario is now running
Mario is now jumping
Mario is now standing
Mario is now jumping
Mario is now standing
Mario is now running
Mario is now standing
Mario is now jumping
Mario is now standing
Mario is now running
Mario is now standing
...
Mario is now standing
Mario is now running
Mario is now jumping
Mario is now standing
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output setti
```

Conclusion:

As this report is now for draft submission, I showcased how I am working towards building Kane. After testing, I observed that the Kane AI easily gets stuck. I have modified my code as described in the debugging section, and I can see that it is making better progress. With entropy loss and train/loss graphs, we can finally compare Kane and Abel.

Graph data lost – will be inserted here when retrieved.

As we can see from the graph, Kane has a lower learning rate than Abel, making it evident that the reinforcement learning algorithm is a well-built algorithm.

REFERENCES

In literature reviews

Świechowski, M. (2020) Game AI competitions: Motivation for the imitation game-playing competition. 2020 15th Conference on Computer Science and Information Systems (FedCSIS). IEEE.

Justesen, N., Debus, M. S., & Risi, S. (2019) When are we done with games?. 2019 IEEE Conference on Games (CoG). IEEE.

Mnih, V., et al. (2015) Human-level control through deep reinforcement learning. Nature, 518(7540), pp. 529-533.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., & Ja´skowski, W. (2015) ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning.