CM3070 Computer Science Final Project  
  
Kane and Abel : AIs that play games

2024

Table of Contents

**Introduction2**

**Literature Review4**

First Literature Review4

Second Literature Review5

Third Literature Review7

Fourth Literature Review8

**Project Design11**

Project Overview11

Overall Structure11

Key Technologies12

Work Plan13

**Implementation - Abel18**

**Implementation – Kane 20**

**Evaluation for Each AI agent31**

**Evaluation for Kane31**

**Evaluation for Abel34**

**Comparative Evaluation37**

**Conclusion39**

**REFERENCE43**

**Project github link:  
https://github.com/SJLEE411/SuperMarioBros-AIs-that-play-games-CM3070**

**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.
5. **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 :
   1. 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.
   2. 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 :
   1. Kane(The first AI) : Developing a preprogrammed pattern for an example genetic algorithms with neural networks.
   2. 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.

1. Training and Testing.
   1. Training Abel using reinforcement learning techniques and will be overviewed with TensorBoard.
   2. Kane will have same settings( for an example , steps sizes) and will be trained with same amount of time.
2. Evaluation and Analysis
   1. 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
   1. 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
   1. 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
   1. 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
   1. 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
   1. 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
   1. 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:

A chart with text and numbers

Description automatically generated with medium confidence

*Image #1 (Gannt Chart)*

A white sheet with black text

Description automatically generated

*Image #2 (Gannt Chart details)*

**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:
   1. For Kane: Collect and analyze performance data focusing on rewards earning over time, completion times, survival rates, and consistency.
   2. 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

A screen shot of a computer code

Description automatically generated

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**

A screen shot of a computer program

Description automatically generated

*Image #3 (Abel Code Screenshot : PPO model training save)*

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 throughout the game.

**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 other than pre implemented self-made DQN network as well as we are implementing preprogrammed behaviors. For an example when Kane meeting a blockages and performs same action over certain times of frames then it will perform a long jump.

**Setup and Dependencies:** A screen shot of a computer program

Description automatically generated

*Image #4 (Kane Code Screenshot : Initialization with DQN)*

We are running environments with a few necessary imports. Since we are using OpenAI’s Gym environment again, we will be importing libraries similar to those used in Abel’s setup. For example, we will import gym, gym\_super\_mario\_bros, SIMPLE\_MOVEMENT, JoypadSpace, cv2, and numpy.

A new import specifically for Kane’s environment is PyTorch. We are incorporating PyTorch to train the agent using a DQN (Deep Q-Network) model. With PyTorch’s tools, such as nn.Module, nn.Linear, and activation functions like ReLU, we can efficiently train neural networks, leveraging GPU support for faster processing.

**Preprocessing:**

**Gray scaling the frames with stacked states:**

In this section, we implement essential image processing and frame stacking functions to prepare input data for the DQN model. These functions ensure that the agent receives relevant, simplified visual information while retaining the temporal context necessary for decision-making. By converting raw game frames into a more manageable format and stacking consecutive frames, the agent can better understand the dynamics of the game environment, allowing it to learn more effectively and make informed choices based on both current and past visual data.A screen shot of a computer program

Description automatically generated

*Image #5 (Kane Code Screenshot : Grayscale)*

The preprocess\_observation function is used to process raw game frames by converting them to grayscale, resizing them to 84x84 pixels, and normalizing the pixel values between 0 and 1, which helps reduce the complexity and size of the input data for the neural network. The stack\_frames function handles the temporal aspect of the game by maintaining a stack of four consecutive frames, either resetting the stack at the start of a new episode or adding a new frame to the existing stack. This stacked state is returned as a 4-channel numpy array, allowing the agent to understand movement over time and make more informed decisions.

**Action baseplates with pre programmed behaviours:**

In this section, we modify the pre-programmed action function to include a long jump feature for Mario. The function processes a game frame by applying Gaussian blur and adaptive thresholding to reduce noise and highlight important areas in the image. Contours are detected to identify obstacles in Mario's path. If an obstacle is detected, the function initiates a long jump by holding the "A" button for 25 consecutive frames to simulate a longer jump duration. If no obstacles are found, Mario continues to move right. The function also ensures that if a long jump is already in progress, it continues for the specified number of frames.

A computer screen shot of text

Description automatically generated

*Image #6 (Kane Code Screenshot : Action preparation with preprogrammed actions)*

To integrate this feature, we track the hold\_jump\_steps variable, which decreases with each frame during a long jump. This ensures that the jump button is held down for an extended period. The function can be incorporated into the overall action selection process, allowing Mario to dynamically adjust his behavior based on environmental cues detected from the pre-processed frames.

**Action Selections:**

This section implements an Epsilon-Greedy action selection function, which combines both learned behavior from the DQN (Deep Q-Network) and pre-programmed behavior for specific scenarios, such as triggering long jumps when obstacles are detected or repetitive actions occur. The function allows dynamic decision-making by balancing between exploration (random actions) and exploitation (choosing the best-known action) based on the current value of epsilon.

A screen shot of a computer program

Description automatically generated

*Image #7 (Kane Code Screenshot : Action selections with preprogrammed actions)*

The **select\_action** function chooses an action either by using the DQN model or a pre-programmed behavior based on the epsilon value. If **use\_pre\_programmed** is set to True, the pre-programmed action logic, including long jump detection, is used. Otherwise, the DQN model predicts the best action, or a random action is chosen with probability epsilon. The function also tracks repeated actions; if the same action is repeated more than 60 times, a long jump is automatically triggered. This is done by setting the action to "right + A" (represented by 2) and holding the jump for 25 frames. This combination of learned behavior and pre-programmed logic ensures that the agent can adapt to both familiar and new situations in the environment.

**Q learning updates:**

This section defines the Q-learning update function compute\_loss, which is crucial for training the Deep Q-Network (DQN) in reinforcement learning. The function computes the loss between the predicted Q-values and the target Q-values, allowing the network to adjust its predictions over time based on the agent's interactions with the environment. It leverages GPU acceleration for faster computation, utilizing the device variable to run operations on a CUDA-enabled GPU if available.

A screen shot of a computer program

Description automatically generated

*Image #8 (Kane Code Screenshot, Q learning)*

The compute\_loss function takes a batch of experience tuples (states, actions, rewards, next states, and done flags) and performs the following steps:

1. **Tensor conversion**: The experience data is converted to PyTorch tensors and moved to the appropriate device (GPU if available).
2. **Q-value prediction**: The current Q-values are predicted by the **policy network** for the current states, and the next Q-values are estimated by the **target network** for the next states.
3. **Q-value extraction**: For the actions taken in each state, the corresponding Q-values are gathered from the network’s predictions. The maximum Q-value for the next state is also extracted to estimate the future reward.
4. **Expected Q-value calculation**: The expected Q-value is calculated using the Bellman equation: reward + gamma \* max(next\_q\_value) \* (1 - done), where gamma is the discount factor and done indicates whether the episode has ended.
5. **Loss computation**: The difference between the predicted Q-values and the expected Q-values is calculated using **Mean Squared Error (MSE)**. This loss is then used to update the neural network weights during training.

By minimizing this loss, the agent's DQN model learns to make better predictions over time, improving its performance in the environment.

**Initializing Environment and Model Training:**

In this section initializes the environment for the DQN agent and sets up the critical components required for training. These components include the Super Mario environment, key hyperparameters to control the learning process, the DQN networks (policy and target networks), the optimizer for training, and the replay memory for storing experiences during the learning process.

A screen shot of a computer program

Description automatically generated

*Image #9 (Kane environment Initialization)*

The environment is created using the gym\_super\_mario\_bros module and simplified with JoypadSpace for a limited set of movements. We also define critical hyperparameters such as the discount factor (gamma), learning rate, and exploration rate (epsilon), which control the agent’s learning and decision-making. The DQN policy network learns to approximate Q-values for the agent’s actions, while the target network provides stable Q-value targets during training, helping to avoid instability in learning. The Adam optimizer adjusts the network's weights based on the computed loss. Additionally, we set up a replay memory to store past experiences, which are sampled during training to improve the agent’s learning by breaking correlations between consecutive experiences and enabling more stable updates. The combination of these elements prepares the agent for effective interaction with the environment and model training.

**Training:**

This section implements the core training loop for the DQN agent. Over multiple episodes, the agent interacts with the environment, collects experiences, and updates its policy network using Q-learning. The training loop handles action selection using an epsilon-greedy strategy, stores experiences in replay memory, and periodically trains the DQN model based on sampled experiences. Additionally, the target network is updated periodically to stabilize learning.

A screen shot of a computer program

Description automatically generated

*Image #10 (Kane Code Screenshot, execution)*

In each episode, the environment is reset, and the agent starts with a fresh stack of frames. The epsilon-greedy strategy is used to select actions: initially, the agent explores (choosing random actions with a higher epsilon), and as training progresses, it exploits learned strategies by selecting actions predicted by the DQN model. Pre-programmed behavior is optionally used when epsilon is high. For each step, the agent interacts with the environment, receiving rewards and transitioning to a new state. These experiences (state, action, reward, next state, and done flag) are stored in the replay memory for future training.

Once enough experiences are collected (i.e., memory has at least batch\_size samples), the agent samples a batch from the memory and updates the policy network by minimizing the loss between the predicted Q-values and the expected Q-values. The target network is updated periodically every target\_update episodes to ensure stability during training. As the episodes progress, the epsilon value decays, reducing exploration and encouraging exploitation of the learned policy. The training loop continues until the desired number of episodes is reached, allowing the agent to learn and improve its performance over time.

A screenshot of a computer

Description automatically generated

*Image #11 (DQN running simulation)*

**Evaluations for Each AI Agent**

**Evaluations: Kane (DQN with Pre-programmed behavior)**

1. Model Performance
   1. Kane’s total reward per episode gradually increased over time showing steady leaning progress. However, there was a noticeable blockage on learning after 20-30 episodes, suggesting limitations in further improvement with the current DQN and pre-programmed behavior.
2. Action Selection:
   1. The action distribution showed a heavy reliance on rightward movement and jumping. Kane occasionally displayed diversity by using long jumps, but this behavior was mostly triggered by the pre-programmed conditions rather than learned strategies. The model lacked adaptability in real-time, often repeating the same set of actions especially areas with limited obstacles like gaps.
3. Long Jump Features(Preprogrammed actions):
   1. The long jump system worked as intended, activating effectively when obstacles were detected. However, there were instances where the jump was overusing it triggered unnecessarily, leading to inefficiencies. In stages where quick reactions were critical the long jump occasionally led to collisions due to poor timing.
4. Q-values:
   1. The average Q-values showed improvement during training but plateaued after a certain number of episodes. This indicated that the model reached a point where it could no longer improve decision-making. The Q-values suggested moderate confidence in action selection but lacked the depth of decision-making seen in more complex environments.
5. Exploration vs Exploitation:
   1. Kane maintained a good balance between exploration and exploitation during early training. However, as the exploration rate decayed (epsilon), the model became more exploitative, focusing on repetitive actions. This led to less exploration in later stages, reducing Kane’s ability to learn and adapt to new challenges.

**Challenges and Considerations for Kane:**

The main challenge in Kane's development was integrating the pre-programmed long jump feature with the self-made Deep Q Network (DQN). Initially, we attempted to implement Finite State Machine (FSM) logic with pre-programmed behavior, but the learning rate was too static and ineffective, leading to slow learning and poor adaptability, especially compared to Abel's dynamic reinforcement learning approach. To address this, we shifted to a DQN, which proved more reliable, and combined it with pre-programmed behaviors like triggering a long jump when the same actions were repeated multiple times. While this approach worked in certain scenarios, the static nature of the long jump feature caused issues in more dynamic levels, and Kane’s performance plateaued, suggesting that the architecture was functional but not as efficient as Abel’s learning methods.

A graph of a graph

Description automatically generated with medium confidence

*Image #12 (FSM result graph)*

A screenshot of a computer

Description automatically generated

*Image #13 (FSM operation Screenshot)*

**Evaluations: Abel (Externally sourced logic behavior : PPO)**

1. Model performance:
   1. Abel consistently achieved higher rewards as training progressed. After approximately 1 hour of training, Abel cleared the first stage surpassing Kane’s model performance.
2. Effective Preprocessing:
   1. Grayscale conversion and frame stacking were employed to capture game dynamics without unnecessary color information, enhancing training efficiency. These preprocessing techniques allowed the model to focus on important visual features, contributing to its improved performance.
3. Algorithm suitability:
   1. As highlighted in the literature review, the Proximal Policy Optimization algorithm is particularly robust for discrete action spaces and capable of handling complex environments like SuperMarioBros. PPO’s ability to manage the action space efficiently played a crucial role in Abel’s adaptability and success in the game.
4. Training loss and Explained Variance:
   1. Using TensorBoard, it was observed that Abel’s training loss steadily decreased throughout the training process, indicating that the PPO algorithm was successfully learning from interactions with the game environment. The explained variance also increased, further showing that Abel’s model was becoming more confident in its decision-making. As training continued, both metrics showed significant improvement, reinforcing the model's growing effectiveness. After an hour of training, Mario successfully completed the first stage of the game, reflecting the model's strong learning capabilities.
5. Exploration vs Exploitation:
   1. Abel maintained a balanced approach to exploration and exploitation throughout training. The model adapted to new challenges effectively, exploring different strategies when faced with obstacles. Unlike Kane, Abel showed a greater ability to learn from its mistakes and adjust its gameplay dynamically.

**Challenges and Considerations for Abel:**

The primary challenge for Abel was occasional freezing in the game environment, which caused spikes in TensorBoard statistics. This slowed down training cycles and required more time for Abel to achieve optimal performance. Debugging this issue remains a priority, but overall, Abel demonstrated superior adaptability and performance.

A screenshot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generated  
  
*Image #14 (TensorBoard: Abel stats output )*

A screenshot of a computer

Description automatically generated  
*Image #15 (Tensor Board: Abel Screen shot)*

**Comparative Evaluations:**

1. **Learning Approach:**

* **Kane (DQN with Pre-programmed Behavior)**: Kane’s learning approach was a combination of a self-made Deep Q Network (DQN) and pre-programmed behaviors such as the long jump. While this approach worked in predictable scenarios, it lacked adaptability. The pre-programmed logic often restricted the flexibility of the model, making it less effective in dynamic situations where quick decision-making and novel strategies were required.
* **Abel (PPO with Externally Sourced Logic)**: Abel, on the other hand, used the Proximal Policy Optimization (PPO) algorithm, a robust reinforcement learning technique. Abel’s learning process was highly adaptive and dynamic, allowing it to improve continuously as it encountered new obstacles and challenges. The algorithm's capacity to handle the complexity of Super Mario Bros led to superior learning, especially in levels with more intricate layouts and enemy patterns.

**Advantage**: **Abel** outperformed Kane in adaptability and learning efficiency due to the dynamic nature of PPO compared to the static nature of Kane’s pre-programmed logic.

1. **Performance Metrics:**

* **Kane**: Kane showed steady improvement in total rewards, but performance plateaued after 50 episodes. The reliance on pre-programmed behaviors caused Kane to struggle in more complex levels, resulting in inconsistent completion times and lower survival rates.
* **Abel**: Abel consistently achieved higher rewards as training progressed, with its performance surpassing Kane’s after about 50 episodes. Abel’s ability to explore new strategies and exploit learned knowledge allowed it to complete levels more efficiently, with faster completion times and higher survival rates.

**Advantage**: **Abel** consistently performed better in terms of rewards, level completion times, and survival rates compared to Kane.

1. **Generalization:**

* **Kane**: Kane’s generalization ability was limited. The model overfitted to specific scenarios due to the reliance on pre-programmed actions, struggling to adapt when faced with new or unseen game elements.
* **Abel**: Abel demonstrated strong generalization capabilities, adapting well to unseen levels and new challenges. The PPO algorithm allowed Abel to efficiently adjust its strategies, enabling it to handle both familiar and unfamiliar stages with relative ease.

**Advantage**: **Abel** was more versatile and effective in generalizing across new environments, whereas **Kane** struggled with overfitting.

1. **Policy Stability:**

* **Kane**: Kane’s policy exhibited stability during training but lacked flexibility. While the pre-programmed behaviors ensured consistency, they also limited Kane’s ability to adapt to new challenges. The policy was often too rigid, leading to repeated mistakes in levels with unexpected obstacles.
* **Abel**: Abel’s policy was not only stable but also flexible. The PPO algorithm enabled Abel to adjust its strategy dynamically, maintaining stability while still responding to new challenges. Abel’s adaptability made it more robust in handling varying game environments.

**Advantage**: **Abel** displayed greater policy flexibility while maintaining stability, which **Kane** lacked due to the rigid nature of its pre-programmed logic.

1. **Exploration vs. Exploitation:**

* **Kane**: Kane showed limited exploration due to its reliance on pre-programmed behavior. While it was able to exploit learned knowledge, the exploration phase was often cut short, reducing its ability to discover new strategies or adjust effectively.
* **Abel**: Abel maintained a balanced exploration-exploitation tradeoff throughout training. It explored new strategies and improved its performance through trial and error. Abel’s exploration capabilities allowed it to discover more efficient solutions to various challenges in the game.

**Advantage**: **Abel** had a better balance between exploration and exploitation, leading to more effective gameplay strategies than **Kane**.

1. **Pre-programmed vs. Reinforcement Learning:**

* **Kane**: The integration of pre-programmed behavior, such as the long jump feature, provided Kane with a deterministic method for handling repetitive actions. However, this rigidity became a limitation in dynamic levels where adaptability was crucial. The pre-programmed behavior made Kane more predictable and less capable of handling the game’s unpredictable elements.
* **Abel**: Abel, relying solely on reinforcement learning, did not suffer from these limitations. Abel’s learning process was driven by experience, enabling it to develop more sophisticated and adaptive strategies that were effective across a broader range of levels.

**Advantage**: **Abel’s** pure reinforcement learning approach proved more effective in handling a variety of challenges compared to Kane’s mixed DQN and pre-programmed behavior approach.

**Conclusion:**

The Kane and Abel project aimed to develop and compare two AI systems capable of playing the classic game Super Mario Bros using two distinct approaches: Kane, the pre-programmed AI leveraging deterministic logic with Deep Q Networks (DQN), and Abel, the learning-based AI utilizing the Proximal Policy Optimization (PPO) algorithm. This study provided critical insights into the effectiveness, strengths, and limitations of each approach, contributing to a broader understanding of how artificial intelligence can be applied to gameplay scenarios.

The project highlighted several fundamental differences between rule-based AI and reinforcement learning-based AI, with implications for both the theoretical and practical applications of each method. Through extensive trials, it became clear that while Kane's pre-programmed actions offered predictability and reliability in some cases, Abel's reinforcement learning model demonstrated superior adaptability, learning efficiency, and overall performance in dynamic environments.

**Key Findings and Observations**

**Learning Efficiency and Model Performance:**

One of the most prominent differences between Kane and Abel was their approach to learning and performance during gameplay. Kane's reliance on a pre-programmed long jump feature and DQN led to limitations in its adaptability, especially in levels that required more dynamic decision-making. Kane exhibited consistent but slower learning, with total rewards plateauing after a certain number of episodes. This was due to the fact that the model's pre-programmed behaviors were rigid, leading to over-reliance on repeating the same actions when faced with new obstacles.

In contrast, Abel, powered by PPO, demonstrated a much more adaptive learning process, constantly improving as it encountered more complex stages. By leveraging PPO's strengths in reinforcement learning, Abel's model was able to balance exploration and exploitation more effectively, which enabled the model to learn from its mistakes and adapt strategies as needed. Over time, Abel surpassed Kane in terms of both total rewards and completion times, achieving much faster and more efficient level completions across all stages.

The use of effective preprocessing techniques, such as grayscale conversion and frame stacking, further enhanced Abel's performance by reducing the complexity of input data and improving training efficiency. These techniques allowed the model to focus on the most relevant features of the game, leading to a more streamlined learning process and improved generalization.

**Adaptability and Generalization:**

Another significant area of distinction between the two AI models was their ability to generalize across different levels and environments. Kane, due to its deterministic and pre-programmed behaviors, struggled to generalize to new game scenarios. The model often overfit to specific patterns it had learned during training, making it less capable of handling unexpected challenges or variations in the environment. For example, while the pre-programmed long jump worked effectively in certain stages, it became a hindrance in more dynamic levels where precise and adaptive jumps were necessary.

Abel, on the other hand, excelled in generalization due to its reinforcement learning foundation. The PPO algorithm enabled Abel to learn and adapt dynamically, even in unseen environments. This adaptability allowed Abel to perform well in a variety of scenarios, making it a more versatile AI agent than Kane. Abel's ability to generalize across different levels demonstrated the power of learning-based models in handling complex environments with changing conditions.

**Policy Stability and Decision-Making:**

Both AI agents exhibited stability in their respective policies, but the nature of their decision-making processes differed significantly. Kane's policy was highly stable, primarily because of its reliance on pre-programmed behaviors. However, this stability came at the cost of flexibility, as the model struggled to adjust its strategies when faced with new challenges. The DQN model's decisions were often static, leading to repeated actions and inefficient decision-making in more complex levels.

In contrast, Abel's decision-making process, driven by PPO, was both stable and flexible. PPO allowed Abel to adapt its policies based on the game environment, leading to more dynamic and context-aware decision-making. This balance of stability and flexibility gave Abel a significant advantage in terms of performance, particularly in levels that required quick reactions and adaptive strategies.

**Exploration vs. Exploitation:**

One of the key challenges in developing AI systems for gameplay is finding the right balance between exploration (trying new actions) and exploitation (choosing the best-known action). Kane, due to its pre-programmed behaviors, showed limited exploration, focusing primarily on exploiting the actions it had learned through its pre-programmed logic. While this led to consistent performance in predictable environments, it limited Kane's ability to discover new strategies or improve its performance in unfamiliar levels.

Abel, on the other hand, demonstrated a more effective balance between exploration and exploitation. The PPO algorithm enabled Abel to explore different strategies during gameplay while still making decisions based on the most effective actions it had learned. This balance allowed Abel to continuously improve its performance over time, discovering more efficient solutions to various challenges in the game. The dynamic exploration process was crucial to Abel's success in outperforming Kane, particularly in levels with complex obstacle layouts and enemy patterns.

**Challenges and Limitations:**

Despite Abel's overall superiority, both AI agents faced certain challenges during development and evaluation. For Kane, the primary challenge was integrating the pre-programmed long jump feature with DQN. While this approach worked well in some scenarios, the static nature of the long jump caused issues in more dynamic levels, leading to slower learning and reduced adaptability. Kane's limited exploration capabilities and reliance on pre-programmed actions ultimately hindered its ability to perform efficiently in more complex levels.

Abel's main challenge was the occasional freezing of the game environment, which caused spikes in TensorBoard statistics and slowed down training cycles. This issue was attributed to the Super Mario engine, which sometimes froze its frames during gameplay, impacting Abel's ability to train consistently. Debugging this issue remains a priority, and addressing it could further enhance Abel's training efficiency and performance.

**Broader Implications and Future Directions:**

The Kane and Abel project offers valuable insights into the practical applications of rule-based and learning-based AI systems in game environments. The findings from this study highlight the limitations of deterministic AI approaches in handling complex, dynamic environments, while also showcasing the potential of reinforcement learning techniques in developing more adaptive and robust AI agents.

Looking forward, future work could focus on further refining Abel's learning algorithms, addressing the technical challenges related to freezing frames, and exploring additional reinforcement learning techniques, such as multi-agent learning or hybrid approaches that combine the strengths of both pre-programmed and learning-based AI models.

In conclusion, while both Kane and Abel contributed valuable insights into AI development for gaming, Abel's reinforcement learning-based approach proved to be more effective, adaptive, and capable of handling a wide range of challenges. The results from this project demonstrate the importance of flexibility, exploration, and adaptability in AI systems, particularly in environments as dynamic and unpredictable as Super Mario Bros.

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.*

APPENDIX