THE MONSTERS LEARN TO FIGHT BACK

The Computing Project CI601

Abstract

This technical report outlines the development of "The Monsters Learn to Fight Back," an innovative Artificial Intelligence-driven battle simulator designed for the tabletop role-playing game Dungeons & Dragons (D&D). Leveraging reinforcement learning, this simulator enables monsters to autonomously develop and execute strategic combat behaviours, thus enhancing the realism and complexity of player encounters. Implemented within the Unity game engine, this project prioritizes intuitive visualization, efficient integration of AI frameworks (ML-Agents), and rapid prototyping capabilities.

The initial implementation focuses on fundamental combat interactions between player characters and monsters—beginning with a simplified scenario involving a single Goblin. The simulator subsequently scales complexity by introducing multi-agent scenarios, incorporating advanced pathfinding algorithms, strategic decision-making processes, and adherence to established combat mechanics. Through comprehensive research, including comparative analyses of D&D editions and existing simulation models, this report thoroughly justifies design choices, AI methodologies, and technology adoption.

This project not only aids Dungeon Masters in orchestrating engaging, strategic combat encounters but also explores broader implications of applying reinforcement learning to complex decision-making scenarios within interactive entertainment.

George McDonald

Contents

Introduction

Dungeons and Dragons (D&D), a tabletop roleplaying game (TTRPG) first introduced in 1974, has captivated generations with its depth, strategic complexity and immersive storytelling. Central to its enduring popularity is a richly detailed, turn-based combat system, where players and monsters alike make tactical decisions that shape a dramatic narrative. However, as encounters become larger and more intricate, managing combat can grow increasingly challenging for even seasoned Dungeon Masters (DMs), often slowing gameplay and diminishing the experience.

"The Monsters Learn to Fight Back" addresses this complexity by introducing artificial intelligence (AI) into D&D combat scenarios. This project aims to automate and enhance the decision-making for monsters, free DMs to focus on narrative and player interaction. Utilizing reinforcement learning (RL)—specifically Unity's ML-Agent's toolkit—this simulation allows monsters to independently learn and adapt strategic behaviours, creating more dynamic, realistic and engaging encounters.

Initially, development focuses on the core mechanics involving a simplified scenario of combat between a single player-character and a goblin. This foundational system is then expanded progressively, incorporating multiple entities, strategic considerations, and complex tactical interactions.

This report details the systematic journey of the project beginning with comprehensive research covering relevant editions of D&D, evaluating existing simulation tools, and suitable AI algorithms. Subsequent sections delve into system design, development choices, ethical and legal considerations and the technical implementation of RL techniques. Ultimately defining a twofold objective: to demonstrate that RL can effectively assist DMs in managing expansive and complex combat situations, and to illustrate that sophisticated, adaptable combat strategies can be developed autonomously through RL techniques.

Research

Dungeons and Dragons

D&D has evolved significantly since its inception in 1974, with multiple editions that reflect gameplay philosophy, mechanics and community engagement. The two most popular versions of this game 3.5 Edition (3.5e) and 5th Edition (5e) are both covered by D&D's OGL (Open Gaming License) which is why these two were chose to compare for this project.

3.5e is known for its depth, detail and complexity, appealing to seasoned players who prefer the intricate mechanics. In contrast, 5e is streamlined and accessible, designed to prioritise storytelling and ease of entry (Kyle). Key differences include a simplified skill system, more balanced class design, a reduced dependency on magic items and shorter, less "crunchy" combat sequences.

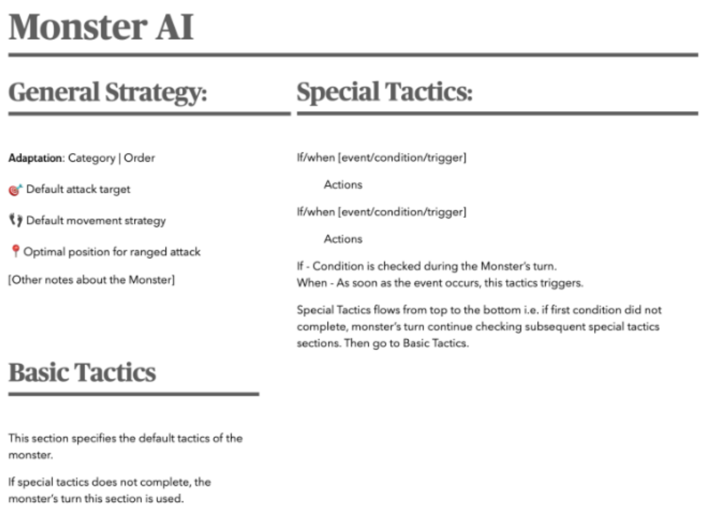
Given the goals of this project, to train AI to learn and adapt within combat scenarios, 5e was chosen; its reduced mechanical overhead and accessibility allows for faster iteration and easier implementation, making it ideal for reinforcement learning and simulation. The widespread popularity (Linward, 2024) of 5e also ensures greater relevance and user familiarity. Since the beginning of this project 5e was update to a new 2024 edition, however the 2014 rules will be used due to the resources for it being more widespread.

Analysis of Similar Projects

A review of existing D&D simulations and AI projects was conducted to benchmark current capabilities and identify innovation gaps.

The Monsters Know What They Are Doing (Ammann, 2019) is a popular tactical blog/book series offering monster strategy insights based upon the stat-blocks and lore available for the creatures. Due to ethical concerns expressed by the author, this content could not be used for training, however it provided context for what could be considered good AI decision making.

While the previous blog is considered the holy grail for D&D monster tactics, it doesn't have any AI within it, "The DND 5e Monster AI" (MugenMuso, 2021) on the other hand does, along with various resources to create unique stat blocks for monsters which gives a summary of their best possible options, during this research I found that the designer of this system had some interesting design choice regarding player targeting. The idea behind this concept was to create a DM-less D&D game where monster strategies are pre-determined. However, this approach is deterministic, not allowing for real-time adaptability or training via feedback.



Monster AI

General Strategy:

Adaptation: Category | Order

🎯 Default attack target

👣 Default movement strategy

📍 Optimal position for ranged attack

[Other notes about the Monster]

Special Tactics:

If/when [event/condition/trigger]

    Actions

If/when [event/condition/trigger]

    Actions

If - Condition is checked during the Monster's turn.
When - As soon as the event occurs, this tactics triggers.

Special Tactics flows from top to the bottom i.e. if first condition did not complete, monster's turn continue checking subsequent special tactics sections. Then go to Basic Tactics.

Basic Tactics

This section specifies the default tactics of the monster.

If special tactics does not complete, the monster's turn this section is used.

Figure 1 - Monster AI template

The D&D Mass Combat Simulation (Ohnemus, 2019) explores large-scale combat interactions, the kind this project eventually hopes to assist with. However, this original paper worked in a command line window using an ASSCI style display to show what is happening. This paper helped with a foundational understanding of how an application like this could be developed and expanded upon.

The other simulator that appeared during research into similar systems was the "D&D Battle Simulator" offers automated combat resolution, but using maths to calculate how a fight would fair, but lacks any form of AI or learning

Most recently a triple-A adaptation of D&D 5e was released, Baldur's Gate 3 by Larian Studios. This groundbreaking RPG shows the potential for what a largescale D&D simulation could look like, while it does feature AI, all behaviours are scripted, leaving no learning or adaptation. The AI navigates pre-programmed rules and decision trees.

These projects have shown that while tactical assistance and automation are well-explored areas; true learning-based AI in a D&D 5e context particularly with a reinforcement learning model and with a modern visualization system, remains a novel contribution. This project's uniqueness lies in enabling monsters to learn optimal strategies rather than be hard coded to follow rules.

AI Models and Techniques

Central to this project is the use of Reinforcement Learning (RL) as the backbone of the monster AI system. Unlike supervised learning, RL allows agents to learn optimal behaviours through trial and error while interacting with an environment and receiving feedback from the system in the form or rewards and penalties. This method aligns well with D&D combat, which features sequential decision making, delayed rewards and emergent strategy that only complexifies over time.

The primary choice of model is Proximal Policy Optimization (PPO) first introduce by (Schulman et al., 2017) which was selected for its balance between training stability and performance efficiency. It allows for continuous learning in adapting environments.

Initially Deep Reinforcement Learning (DRL) and Neuroevolutionary approaches were explored. However, due to licensing constraints that restricted access to a curated dataset ruled out the option of using DRL. Nueroevolution methods, such as genetic algorithms which are described as "The genetic algorithm is search-heuristic which is inspired by Darwin's theory of natural evolution. It reflects the process of the selection of the fittest element naturally." (Baeldung, 2024) or NeuroEvolution of Augmented Topologies (NEAT) (Stanley & Miikkulainen, 2002), a front runner in this field, showed promise. The ability to evolve network weights and structures almost mimicking real-world evolution would make for a vastly competitive AI, and as NEAT has shown promise in similar fields such as videogames, robotics and simulations it was a high contender for the AI to be implemented.

Ultimately, PPO was selected because it supports scalable training while also having a seamless integration with one of the visualization applications that ended up being used, whereas finding solutions to implementing NEAT showed much more difficult to find.

Evaluation Metrics and Training Data

Since RL doesn't rely on labelled datasets, ensuring meaningful learning, several evaluation metrics and reward mechanisms were developed. This would enable the model to learn over time while interacting with its environment.

The three ways of evaluating are, reward structures, episode structures and evaluation metrics. Initially reward structures start of simple, but as development further fine tuning the reward metrics to increase the performance of models to the best of their ability.

Reward structures can look something like this:

- Move Closer = +1
- Move Away = -1
- Attacking = +5
- Dying = -10

The values over the course of development were experimentally tuned to encourage survival, aggression and strategic movement, rather than simply moving to enemy and attacking.

Episode structures allow for the fine tuning of when PPO updates its policies, by making sure that certain thresholds were met before ending an episode it gives the model the best source of data to work with. Episodes can end for reasons such as a fixed number of turns being reached, death or a successful kill. But even after a single episode, agents were reset and rerun multiple times to increase the training, allowing for further exploration and fine tuning of strategies.

While evaluation metrics such as reward tracking, win rate and decision entropy give insight into how well the AI is performing with any specific task. These systems ensure that the monster AI is not only functional but learning meaningful patterns. Over time, trends in these metrics indicate whether strategies are improving plateauing or regressing, which gives informs adjustment in training parameters.

Design

Visualization Application

For a battle simulation rooted in tabletop mechanics like D&D, effective visualization is not just an aesthetic; its functional. The choice of platform directly correlates to development speeds, real-time debugging and AI-environment interaction. After evaluating three of the most popular engines of today: Godot, Unity and Unreal Engine, Unity was selected for its robust features, ML-Agent integration, and accessible 3D visualization capabilities.

Unity provides a clean interface for representing a grid-based system, which is foundational to D&D combat. Player and monster entities can be instantiated as simple visual representations to indicate position, orientation and states. Through the Unity Editor, game designers can rapidly prototype and test interactions, which is invaluable for reinforcement learning, where feedback loops require ongoing adjustment. Unity provides an asset system alongside scene management tools allowing for scalable environments, setting the stage for future additions like larger maps, environmental hazards and party-versus-party encounters.

Programming Languages

This project makes use of a hybrid programming stack, driven by the requirements of both game development and artificial intelligence integration:

C# is the primary language used in Unity for implementation, while it is more than possible to code in other languages such as C++ or Java, Unity has been developed with a C# forward presence. The use of this language is to handle the game systems, such as combat logic, entity management and UI systems.

Python on the other hand used alongside Unity's ML-Agents for training reinforcement learning models. Python's ecosystems, especially with libraries such as TensorFlow, Numpy and PyTorch, supports advanced AI workflows, logging and evaluation.

JavaScript Object Notation (JSON) is leveraged for data storage capabilities. weapons, character Stats and entity configurations are all stored in JSON files, which are loaded dynamically at runtime. This allows for flexible content management without modifying source code, supporting both rapid iteration and long-term scalability. See Figure 13 - Goblin JSON file

Multi-Agent Considerations

While the initial training environment focuses on a single monster versus a single player character, the simulation is designed with multi-agent expansion in mind. D&D is inherently a multi-agent system, where multiple entities operate semi-independently, having shared goals or opposing objectives.

Multi-Agent Reinforcement Learning (MARL) however introduces several new challenges:

Non-Stationary learning parameters, as each agent learns and adapts the environment becomes more dynamic from the perspective of other agents. This requires more stable and robust algorithms to ensure meaningful convergence.

Agents on the same side must learn to cooperate with one another, while also maximizing personal rewards. This competition vs. cooperation requires shaping reward structures that balance individual success with team strategy.

Due to the structure of D&D combat operating around a turn-based system, agents must learn not just what to do, but when to act, considering turn priority, cooldowns and action economy.

Combat Mechanics

At the heart of the D&D combat system is a structured, turn-based loop that dictates the flow of encounters. Each combat round consists of a full cycle through an initiative order, with every entity taking a turn to perform actions, move, or react to the battlefield.

This structure should be maintained by a list of combatants sorted by initiative, calculated at the beginning of combat. The system then cycles through this list, granting each entity a turn. Understanding the baseline cycle of combat and its core features helps to ease the development cycle.

Actions In Combat

A turn for an entity during a round consists of four potential action types:

- Action the primary activity on a turn. For most entities this includes attacking, dashing, disengaging, dodging, hiding, or using an object.
- Bonus Action, which allows certain classes or monsters features to do additional things on their turn.
- Movement, during a turn a creature can move up to its movement speed.
- Reactions are triggered in response to a specific condition outside of the entities turn, allowing entities to react to every changing battle.

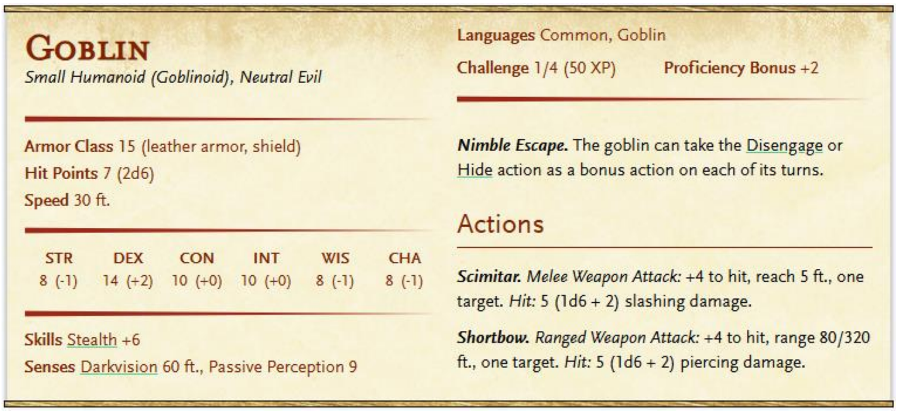
To track these through combat, Booleans were created to flag whether an action type had been used or not, which would reset at the beginning of a creatures turn. While movement is tracked through a for loop to allow movement to be incremental.

Stat Block Representation

Creating a modular system to replicate stat-blocks found in D&D while also allowing this system to be extended to player characters each entity has a set of properties such as:

- Ability Scores (STR, DEX, CON, INT, WIS, CHA).
- Armor Class (AC).
- Hit Points (HP).
- Speed.
- Actions and Traits.

For this project a Goblin was chosen as the prototype monster, for the simplicity of its stat block while also having some tactical depth. To train against this entity a level 1 fighter character was made and then implemented into the same system.


GOBLIN

Small Humanoid (Goblinoid), Neutral Evil

Armor Class 15 (leather armor, shield)
Hit Points 7 (2d6)
Speed 30 ft.

STR	DEX	CON	INT	WIS	CHA
8 (-1)	14 (+2)	10 (+0)	10 (+0)	8 (-1)	8 (-1)

Skills Stealth +6
Senses Darkvision 60 ft., Passive Perception 9
Languages Common, Goblin
Challenge 1/4 (50 XP) Proficiency Bonus +2

Nimble Escape. The goblin can take the Disengage or Hide action as a bonus action on each of its turns.

Actions

Scimitar. *Melee Weapon Attack:* +4 to hit, reach 5 ft., one target. *Hit:* 5 (1d6 + 2) slashing damage.

Shortbow. *Ranged Weapon Attack:* +4 to hit, range 80/320 ft., one target. *Hit:* 5 (1d6 + 2) piercing damage.

Figure 2 - Goblin Stat Block

Movement and Positioning

D&D battle maps consist of two styles, square-grid and hex-grid, while benefits exist for the hex style, square based grids are far more prevalent which led to the choice of that being used in this simulation. The grid is aligned with D&D's 5-foot-square model. The Unity environment overlays a 2D grid onto the terrain, each entity moves in discrete tiles. Movement uses the A* pathfinding algorithm to determine optimal routes to targets or objectives, this implementation also considers terrain and entity locations. Entities are also programmed with an ability to move one square in a specific direction, this is thanks to commands being executed one at a time, which also allows entities to move, attack and then move again.

The combination of these two systems allows for tactical flexibility including, moving towards an enemy if they are out of range, repositioning for flanking or escape, along with disengaging. Entity movement is determined by speed and action usage, for example dashing doubles an entities movement but consumes their action for a turn.

Attack Mechanics

Combat resolution follows D&D 5e rules:

1. Declare target.
2. Roll To-Hit: Roll a D20(20-sided dice), add attack modifier.
3. Compare result to target's AC, if equal or greater, the attack hits.
4. Roll weapon dice and add modifiers.
5. Target loses HP equal to the amount rolled.

While in traditional D&D players and monsters would be considered downed if their HP reaches zero, for simplicity of the system, we shall just consider this a kill. Weapon stat's as well as modifiers are stored in external JSON files, this means that multiple entities can access the same information.

While additional mechanics for combat exist, including conditions, special abilities and spells, due to the complexity of implementing these, they have been omitted from the current version of the application. However, the modular implementation would allow for expansion of these rules at a later stage.

The combat systems' modular and rule fidelity ensure that the simulation stays grounded in authentic D&D mechanics, while also allowing AI to engage meaningfully with strategy, positioning, and turn-based decision making.

Project Management

Effective project management was essential in guiding the development of this AI-driven battle simulator. Early in the process, a detailed Gantt chart was created to map out the major milestone and development phases. The project was then broken down into manageable stages:

1. Research and Ideation
2. Environment Setup and Unity Prototyping
3. AI Model Integration and Testing
4. Combat System Implementation
5. Evaluation and Final Report

Within each stage were internal deadlines, to ensure steady progress, with buffer time added for unexpected challenged. The Gantt chart was updated periodically to reflect priorities and changing scope in the project. See Figure 10 - Updated Gantt Chart

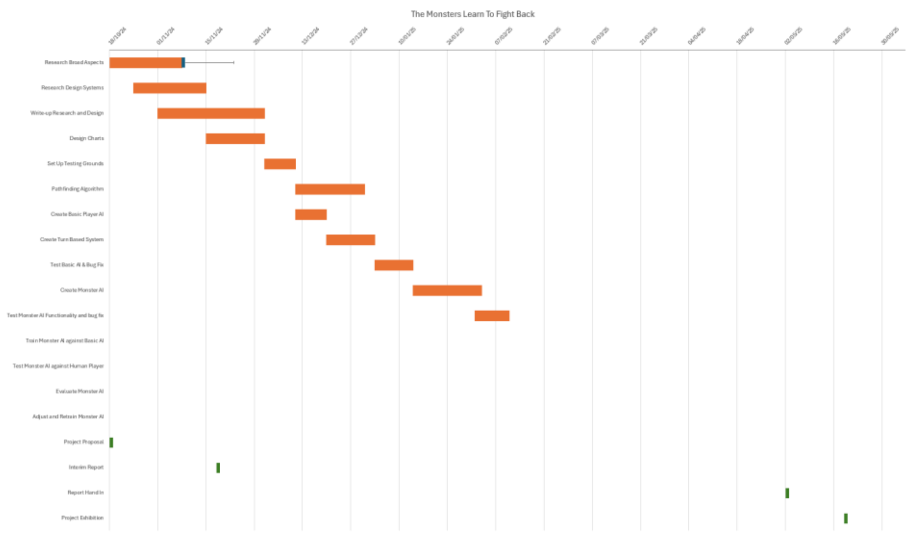


Figure 3 - Original Gantt Chart

Requirement Analysis

While the initial project had goals to create a reinforcement learning-based monster AI, simulating D&D 5e combat encounters through Unity and maintaining modularity for future expansion.

As development progressed, certain requirements evolved, from using pre-written data with DRL to just PPO-based learning. Visual importance was deprioritized in favour of functionality. Manual turn switching, and user interactions were adapted to focus on developing a solid foundational AI.

This adaptive approach to the project ensured that while ambitious, the project was achievable within the development time. Below is a table of the initial requirements table I drafted with current implementation status:

ID	**Requirement**	**Type**	**Status**	**Notes**
FR1	Enable a Goblin to move across a 10x10 grid	Functional	Complete	Implemented using A* pathfinding and Unity tile-based movement
FR2	Allow the Goblin to attack a player	Functional	Complete	Includes hit checks, weapon stats, and damage resolution via JSON
FR3	Implement a turn-based system with initiative	Functional	Complete	Fully operational; initiative rolls use Dexterity modifier
FR4	Integrate reinforcement learning for monster behaviour	Functional	Complete	PPO implemented through ML-Agents and Python (Goblin.py)
FR5	Implement Disengage and retreat behavior	Functional	Partial	Behaviour observable via reward shaping, not hard-coded
FR6	Display entities and terrain in a visual grid	Non-Functional	Complete	Unity capsules represent entities on a clear grid
FR7	Train AI without pre-labelled datasets	Non-Functional	Complete	Entirely trained using RL and custom reward signals
FR8	Log reward trends, episode data, and performance	Functional	Complete	Python-side metadata tracking + TensorBoard visualization
FR9	Implement a UI showing entity actions and turn control	Functional	Not Implemented	UI elements scoped out in favor of core mechanics and training
FR10	Support multi-agent environments	Functional	Not Implemented	Architecture supports it; not tested in current build
FR11	Ensure modular, reusable code across systems	Non-Functional	Complete	Namespaced, class-inherited, modular structure

FR12	Maintain compliance with legal and ethical guidelines	Non-Functional	Complete	Avoided external datasets; respected denied permissions
FR13	Design system for future scalability and expansion	Non-Functional	Complete	Structure allows for new monsters, skills, AI models

Table 1 - Updated table of requirements

Risk Management

Several risks were identified and mitigated during the project lifecycle, initially scope features such as full multi-agent dynamics, environmental complexity and battle designing systems were removed to ensure core functionality, reducing the scope creep which was getting out of control. Modular development allowed parallel development on AI training and combat logic, reducing bottlenecks. Backup systems and versioned saves helped mitigate the loss of data or codebase corruption.

Documentation and Version Control

Development was documented with progress notes, iteration logs and version control. While GitHub was not used due the nature of the project, however Unity Version Control (UVC) was used alongside locally stored versions of the application as backups allowed for quick recovery from issues and clearer tracking of changes over time.

Implementation

System Architecture Overview

Implementation began with building the battle simulator in Unity, integrating ML-Agents, and creating a modular combat system on D&D 5e rules. The core structure was divided into functional components to ensure modularity and ease of debugging.

These components include Core Systems (GameEngine, GridManager, TurnOrder, Movement, etc.), Entity Systems (Entity, Player, Monster), AI Systems (ML-Agent subclass, PPO Python script), Resources (Weapons, Stat-blocks), etc. Each script was developed iteratively, starting with the grid system and movement, before slowly layering in more advanced classes, and subclasses.

The use of namespaces allowed classes to be grouped in folders under a singular System Name, importing it in another class allowed for entire Systems or Singular classes to be easily removed and added to different sections of the project.

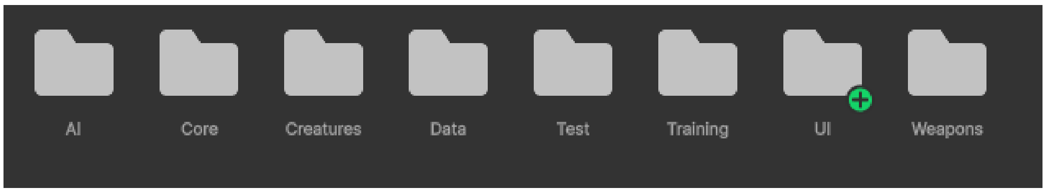
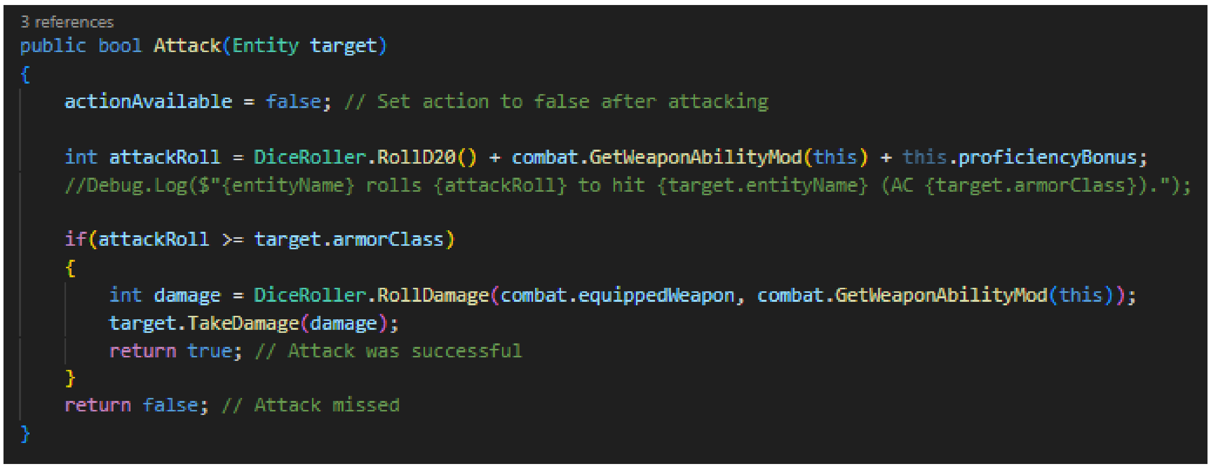


Figure 4 - Grouped Systems in their namespace folders

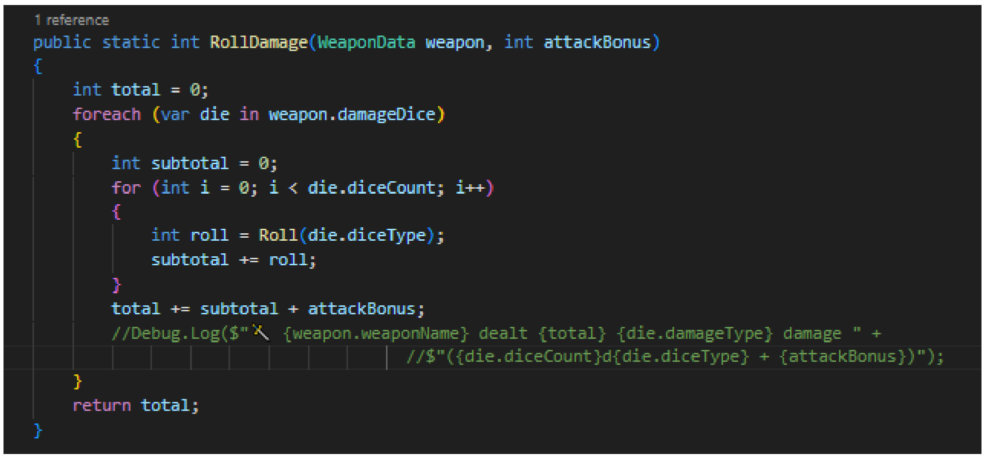
Combat System Coding

Combat logic followed the D&D 5e structure outlined earlier. Actions were implemented as methods called during the active entities turn.



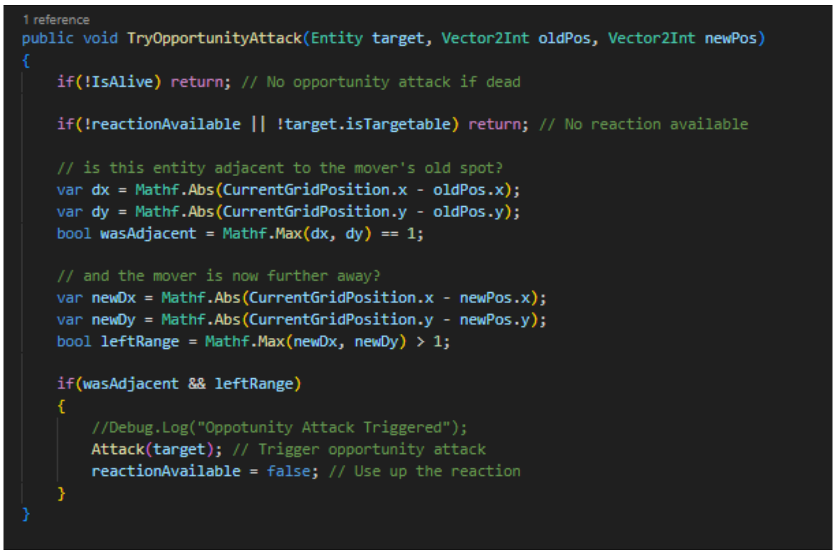
```
3 references
public bool Attack(Entity target)
{
    actionAvailable = false; // Set action to false after attacking

    int attackRoll = DiceRoller.RollD20() + combat.GetWeaponAbilityMod(this) + this.proficiencyBonus;
    //Debug.Log($"{entityName} rolls {attackRoll} to hit {target.entityName} (AC {target.armorClass}).");

    if(attackRoll >= target.armorClass)
    {
        int damage = DiceRoller.RollDamage(combat.equippedWeapon, combat.GetWeaponAbilityMod(this));
        target.TakeDamage(damage);
        return true; // Attack was successful
    }
    return false; // Attack missed
}
```

Figure 5 - Attack function in entity



```
1 reference
public static int RollDamage(WeaponData weapon, int attackBonus)
{
    int total = 0;
    foreach (var die in weapon.damageDice)
    {
        int subtotal = 0;
        for (int i = 0; i < die.diceCount; i++)
        {
            int roll = Roll(die.diceType);
            subtotal += roll;
        }
        total += subtotal + attackBonus;
        //Debug.Log($"⛏ {weapon.weaponName} dealt {total} {die.damageType} damage " +
                                    //$"({die.diceCount}d{die.diceType} + {attackBonus})");
    }
    return total;
}
```

Figure 6 - RollDamage in DiceRoller



```
1 reference
public void TryOpportunityAttack(Entity target, Vector2Int oldPos, Vector2Int newPos)
{
    if(!IsAlive) return; // No opportunity attack if dead

    if(!reactionAvailable || !target.isTargetable) return; // No reaction available

    // is this entity adjacent to the mover's old spot?
    var dx = Mathf.Abs(CurrentGridPosition.x - oldPos.x);
    var dy = Mathf.Abs(CurrentGridPosition.y - oldPos.y);
    bool wasAdjacent = Mathf.Max(dx, dy) == 1;

    // and the mover is now further away?
    var newDx = Mathf.Abs(CurrentGridPosition.x - newPos.x);
    var newDy = Mathf.Abs(CurrentGridPosition.y - newPos.y);
    bool leftRange = Mathf.Max(newDx, newDy) > 1;

    if(wasAdjacent && leftRange)
    {
        //Debug.Log("Oppotunity Attack Triggered");
        Attack(target); // Trigger opportunity attack
        reactionAvailable = false; // Use up the reaction
    }
}
```

Figure 7 - TryOpportunityAttack in Entity

The DiceRoller class handled all dice rolling across the entire application, from initiative rolls to damage dealt. While the base Entity class, which Player and Monster inherited from, held information regarding how to deal with attacks and how they interact with the entities in combat.

Unity and ML-Agents

Unity served as both the game engine and the training environment. Entities were represented visually as coloured capsules on a 10x10 grid, which was rendered as part of an initialization process that ran on start-up.

Unity ML-Agents was used to connect C# game logic with a Python-based training environment. Initially the idea was tested with a script called GoblinAgent, which was written entirely in C# leveraging the power of ML-Agents to test the practicality of the application. Once proven a python script called Goblin.py was developed, which allowed for greater tuning and far more intricate PPO Neural Network (NN) design. This new file is launched through command line, connecting to Unity through gRPC.

This class exposed movement and action decisions as discrete actions. Observations included positions of entities, health values, relative distance and binary flags for action availability. These inputs are normalized and passed to the policy network during training.

Training and Reinforcement Learning Process

Training of the RL network went through three phases, in phase one Goblins were given the ability to move, while the player was a static entity each of these entities would spawn at a random location on the grid. The Goblin was then allowed to move up to 6 steps per turn, with each individual step evaluated independently. The episode would end if:

a. The goblin reached the player (success)
b. The goblin didn't reach the player within a set number of steps (failure)

Rewards were structured to encourage efficient tactical movement and successful completion of the task, while negative rewards were attributed to wasting movement or trying to move to invalid space. At the end of an episode the simulation would reset defining new positions for both the player and goblin and training would continue. Once proven to work with a singular goblin, the simulation was expanded to multiple battlefield instances, allowing for multiple agents to feed into the NN simultaneously.

A file called curriculum was implemented to help hone the goblin's ability to navigate this environment, decreasing the maximum number of steps the goblin had before failure.

Once the goblin was confident in satisfying the objective, the goblin was given the ability to attack and disengage, while the player was also given the ability to attack. Training continued using the model we saved previously from other training to allow the goblin to continue to learn. The conditions for episode ending were updated:

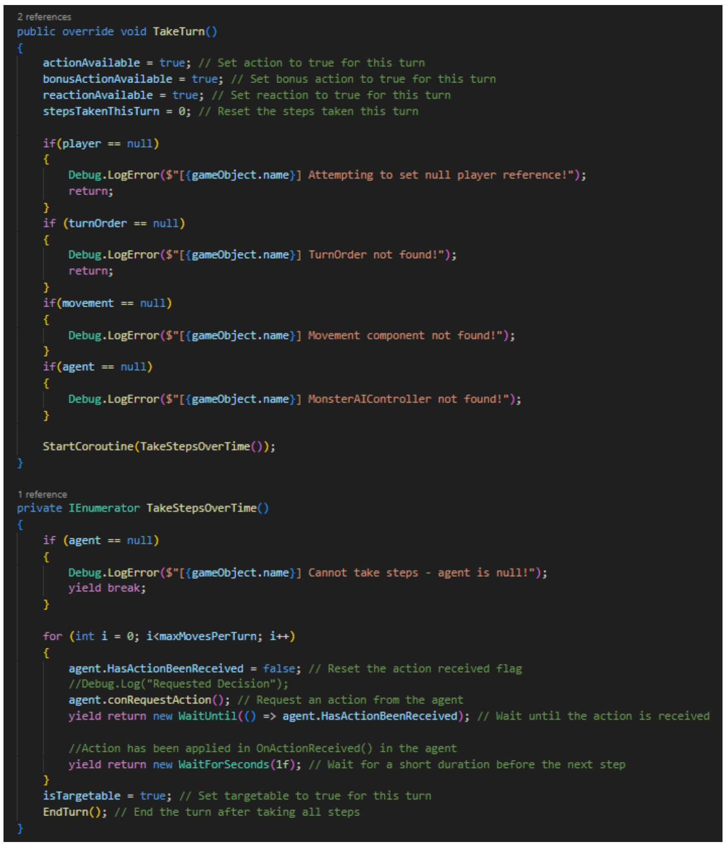
a. The goblin killed the player (success)
b. The goblin died (failure)

The reward structure was also updated to encourage new behaviour, rewards were kept for the goblin moving towards the player, however the penalty for moving away was removed, invalid moves remained, and new rewards were defined, attacking would give a positive reward, while getting attack would apply a negative reward. This was done to teach the goblin that moving to the player, disengaging and then moving away was the best course of action.

Debugging and Iteration

Throughout the implementation, several iterations of this system were required due to unexpected issues, such as issues with the grid and pathfinding, errors with UI implementation.

These problems led to several partial restarts of the project's implementation. Previous systems were kept in labelled directories outside of the application, as many systems that were implemented were reused, reshaped and redesigned to fit the new implementation. In later iterations, modularity and visual debugging aids were heavily implemented to prevent issues arising that could harm the progress of the application.



```
2 references
public override void TakeTurn()
{
    actionAvailable = true; // Set action to true for this turn
    bonusActionAvailable = true; // Set bonus action to true for this turn
    reactionAvailable = true; // Set reaction to true for this turn
    stepsTakenThisTurn = 0; // Reset the steps taken this turn

    if(player == null)
    {
        Debug.LogError($"[{gameObject.name}] Attempting to set null player reference!");
        return;
    }
    if (turnOrder == null)
    {
        Debug.LogError($"[{gameObject.name}] TurnOrder not found!");
        return;
    }
    if(movement == null)
    {
        Debug.LogError($"[{gameObject.name}] Movement component not found!");
    }
    if(agent == null)
    {
        Debug.LogError($"[{gameObject.name}] MonsterAIController not found!");
    }

    StartCoroutine(TakeStepsOverTime());
}

1 reference
private IEnumerator TakeStepsOverTime()
{
    if (agent == null)
    {
        Debug.LogError($"[{gameObject.name}] Cannot take steps - agent is null!");
        yield break;
    }

    for (int i = 0; i<maxMovesPerTurn; i++)
    {
        agent.HasActionBeenReceived = false; // Reset the action received flag
        //Debug.Log("Requested Decision");
        agent.conRequestAction(); // Request an action from the agent
        yield return new WaitUntil(() => agent.HasActionBeenReceived); // Wait until the action is received

        //Action has been applied in OnActionReceived() in the agent
        yield return new WaitForSeconds(1f); // Wait for a short duration before the next step
    }
    isTargetable = true; // Set targetable to true for this turn
    EndTurn(); // End the turn after taking all steps
}
```

Figure 8 - Take Turn in Monster with all visual debugs

Evaluation

The primary goal of the evaluation phase is to determine whether reinforcement learning could be successfully applied to teach a D&D monster (a goblin) how to make strategic combat decisions. This includes assessing the AI's ability to:

- Navigate a grid-based environment towards a target
- Engage a player when in range

- Make use of actions such as attack, disengage, and reposition.
- Improved performance over time based on reward signals.

Evaluation criteria include; successful episodes completed. Ability to work with all systems available.

Current Capabilities

At this stage in development, the simulation is partially functional. Several key systems are operational:

- A full range of actions are usable by the goblin
- The goblin can traverse the battlefield with ease using an A* pathfinding algorithm
- Turn based initiative tracking
- Basic UI feedback showing reward values and successful episodes
- Unity based grid visualization and environment rendering
- All core mechanisms are functional

Training episodes have run successfully for several hours at a time, racking upwards of 8,000 episodes completed.

Issues and Limitations

Despite the successes, the simulation is incomplete in some areas:

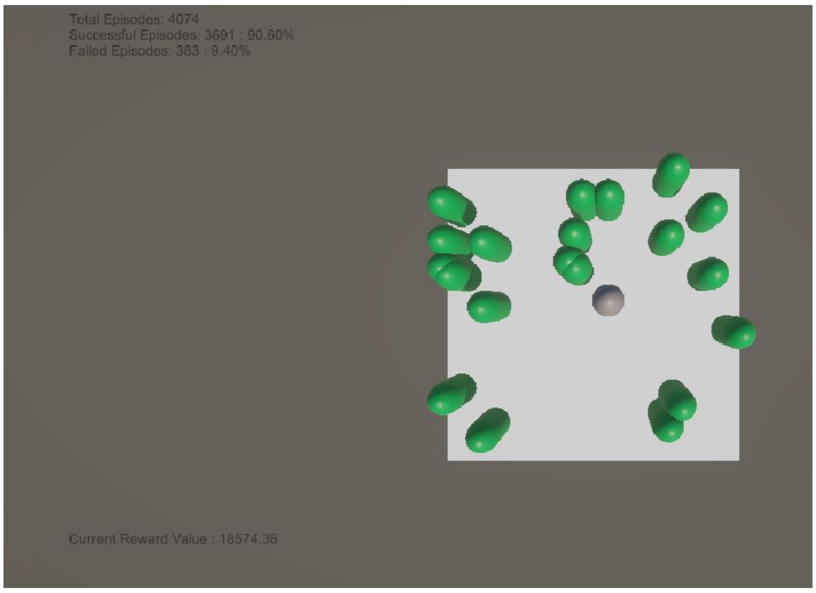
- Multi-agent team fights and flanking system
- Deep strategic NN architecture
- User interface is minimal
- User control of a fully-fledged simulation
- Certain episodes terminate incorrectly due to a state of desynchronization between the python code, Unity and the ML-Agent interface.

Some restarts and codebase revisions were necessary during development to resolve earlier architectural issues and fix desynchronization bugs, one of the most annoying bugs was that my Neural Network wasn't talking to Unity, getting stuck on a environment reset method, I started by debugging by displaying information held by the environment, following this I tried changing the port from 5004 (Unity's base port) to 5005 and connecting through that but this didn't prove fruitful, the next thing I did was create a brand new scene and created a new entity called slime and slowly added scripts to it, I found that it worked perfectly fine with the normal setup and a decision requester component, which automatically calls the python file for a decision. I used a fresh Python file called Basic to help debug, with the idea that something was wrong with the decision requester in my Monster script, I went through the code to see I had called RequestDecision, which would be correct within the agent, but outside of it I needed to call the method conRequestAction.

These setbacks limited the amount of usable training time and delayed the progress towards truly advanced tactical behaviours.

AI Learning Observations

While long-term training results are limited, by the state of the application, early learning behaviour was emergent. The goblin was able to move to the player in the initial movement phase, with over at 90% accuracy.



```
Total Episodes: 4074
Successful Episodes: 3691 : 90.60%
Failed Episodes: 383 : 9.40%

Current Reward Value : 18574.38
```

Figure 9 - Training simulation with over 90% success rate

And while completing the move, attack disengage phase the goblin consistently scored highly at moving to the player attacking and moving away, winning a large majority of the battle simulations.

Though the simulator is not fully realized yet, the implementation to date provides a strong technical foundation for future evaluation. The learning potential demonstrated thus far is promising and aligns with the projects goal of adaptive, strategic monster AI in a D&D simulation environment.

Legal and Ethical Considerations

D&D's OGL

A significant factor in the longevity and success of D&D is the support for community-created content, often referred to as homebrew. This openness has been facilitated by the OGL, the System Reference Document (SRD) (Coast, 2025) and more recently, the adoption of Creative Commons (CC) licensing for certain core mechanics.

These legal frameworks allow creators to legally develop and distribute content that builds on the foundational rules of the game, so long as it doesn't replicate proprietary lore, characters, or settings owned by Wizards of the Coast (WotC). For this project, all systems and mechanics implemented are derived strictly from material covered in the 5e SRD, ensuring compliance with the previously mentioned licensing terms.

The Monsters Know What They're Doing

During the early research phase, the author of The Monsters Know What They're Doing, a widely respected blog and book series, was contacted to request permission to use sections of their writings as training data for the AI. The rationale was that pre-written, rules-based monster strategies could be used to accelerate the learning process and enhance the quality of decision-making models- especially in the early stages of training.

However, the author declined, commenting on the ethical implications of using their work in AI training. In response, the original scope of the AI system was adjusted to exclude any reliance on this data source.

While this removed the chance for an initial dataset available during training, the choice to respect the authors wishes reflects this projects commitment to ethical development. More importantly, it prevented the potential legal and reputational risks associated with using proprietary content without consent.

Ethical Concerns

Copyright and AI Generated Works

Across the broader creative industry, the use of AI-generated content has become a controversial topic. Artists and writers have expressed concern about generative models that reproduce stylistic elements or even direct content from their work without attribution. In the TTRPG space, similar concerns have emerged regarding artwork and worldbuilding content.

Even WotC have faced backlash a couple times. In 2023 after AI-generated art appeared in a D&D publication without their knowledge. The company later apologized and clarified their stance, "Our internal guidelines remain the same with regards to artificial intelligence tools: We require artists, writers, and creatives contributing to the D&D TTRPG to refrain from using AI generative tools to create final D&D products." (Staff, 2023)

These developments highlight the importance of transparency and consent when using AI in creative domains. Despite this project not using GenAI, similar consideration still applies, particularly around creator intent, originality and authorship.

Reflections

This project has pushed me into several areas that were completely new at the outset. Working with unity provided a hands-on experience with real-time game development and scene management, while integrating ML-Agents exposed me to reinforcement learning. I have become more proficient in C#, Python and the practical use of JSON for data storage and retrieval.

Beyond language specific skills, I also gained a much deeper understanding of debugging multi-systems interactions, particularly when AI, pathfinding and visual systems intersect.

This project has shifted significantly from the original conception. Initially envisioned as a reinforcement learning system trained on a tactile dataset, legal and ethical circumstances adaptions were made to make this approach on emergent behaviour through live simulation.

I also underestimated the time cost of Unity's visual and game logic layer, along with Unity's unique development pipeline, which required multiple restarts to address technical errors and scalability concerns. The restarts, while frustrating, ultimately led to better architectural choices and a more robust testable and scalable environment.

There was multiple point throughout this project that tested both my technical resilience and personal motivation. Debugging reinforcement learning behaviours that seemed random or broken, only later to discover a subtle bug, environmental flaw or wrong call in a C# class, was both draining and educational. Additionally, seeing the project fall behind initial expectations was a source of disappointment.

This project has taught me the value of iteration. No design is perfect on the first attempt, and no system works flawlessly without being tested in stages. I have learnt the importance of modular architecture, and simple reproducible environments when developing AI. I have also learned how tightly ethical concerns are with technical design, especially in creative domains where ownership and consent matter almost as much as legal issues. Finally, I learned how to fail productively. Each time I hit a dead end, I came away with a better understanding of Unity, RL, behaviour tuning or system planning.

While I didn't achieve the technical milestone, I set out to reach, I am proud of how far I made it. The fact that an AI-controlled monster can learn based on learned behaviour within a rule-driven environment is meaningful progress. More importantly, I developed a deep understanding of how games, AI and systems thinking come together in practice.

If I were to start all over again, I would place a higher emphasis on building a solid foundation and having a narrow scope for my ambitions and evolve complexity as stability increases. Despite the setbacks, I leave this project with more confidence in my ability to build, adapt, and persevere through the messy, difficult and deeply rewarding process of creating something from the ground up.

Conclusion

The aim of this project was to develop an AI system capable of learning and executing combat strategies within the framework of D&D 5e. By leveraging RL and the Unity game engine, the project set out to create a simulation where a monster, specifically a goblin, could adapt its behaviour in response to the environment it was in, much like a human controlled enemy might do in a live session.

Despite technical setbacks and the complexity of integrating turn-based game logic with AI systems, the project successfully established a foundation. Key systems including movement, pathfinding, action economy and RL integration were developed and tested to the capabilities the system allowed. Although the simulation is not fully functional to the degree that was initially intended. What has been produced demonstrates the potential for Machine Learning (ML) to augment or automate elements of tabletop gameplay.

Alongside this, several significant outcomes were achieved, a modular Unity-based battle simulator that integrates C#, Python and JSON to simulate D&D-style combat. Implementation of a RL model capable of learning movement-based behaviours as well as how to interact with its own action economy. Ethical and legal compliance through conscious avoidance of copyrighted data and considerations of generative AI concerns.

This project still has considerable growth potential. Future iterations of this project could expand on the complexity of the simulation to allow multiple agents systems that incorporates teamwork-based strategies, experimenting with additional monster and player classes, creating more advanced UI for turn tracking, feedback and even user controller or even experimenting with other AI models and techniques to find the truly best fit for this kind of solution. Longer term, the simulation could serve as a combat assistant for DMs, a testbed for encounter design, or even as a standalone tactical game.

This project sits at an interesting intersection of AI, game design/development, and procedural strategy generation. It demonstrates that not only ML can be applied to tabletop systems, but also that doing so requires sensitivity to the cultural, ethical, and structural elements of those systems.

Although the current build is not much more than a prototype, it represents a meaningful step towards dynamic, adaptive game AI within traditionally manual, narrative driven frameworks. The lessons learned here are not only applicable to D&D, but transferable to any project that involves AI navigating complex, rule-based environments.

Thie technical report and project has demonstrated that monsters can, in fact, learn to fight back; and with the right tools, they can do it intelligently.

Bibliography

Ammann, K. (2019). *The Monsters Know What They're Doing*. Saga Press.

Baeldung. (2024). *Genetic Algorithms vs Neural Networks*. https://www.baeldung.com/cs/genetic-algorithms-vs-neural-networks

Coast, W. o. t. (2025). System Reference Document. In (pp. 364).

Kyle. What's the Difference Between DnD 3.5 and 5? (Explained). https://mylarpworld.com/whats-the-difference-between-dnd-3-5-and-5-explained/

Linward, T. (2024). DnD editions differences. https://www.wargamer.com/dnd/editions-differences

MugenMuso. (2021). Making of 5E Monster AI.

Ohnemus, A. (2019). D&D 5E Mass Combat Simulation.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. https://doi.org/10.48550/arXiv.1707.06347

Staff, D. (2023). Updated Statement on AI. In. D&DBeyond: Wizards of the Coast.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, *10*(2), 99-127.

Appendix A: Additional Figures

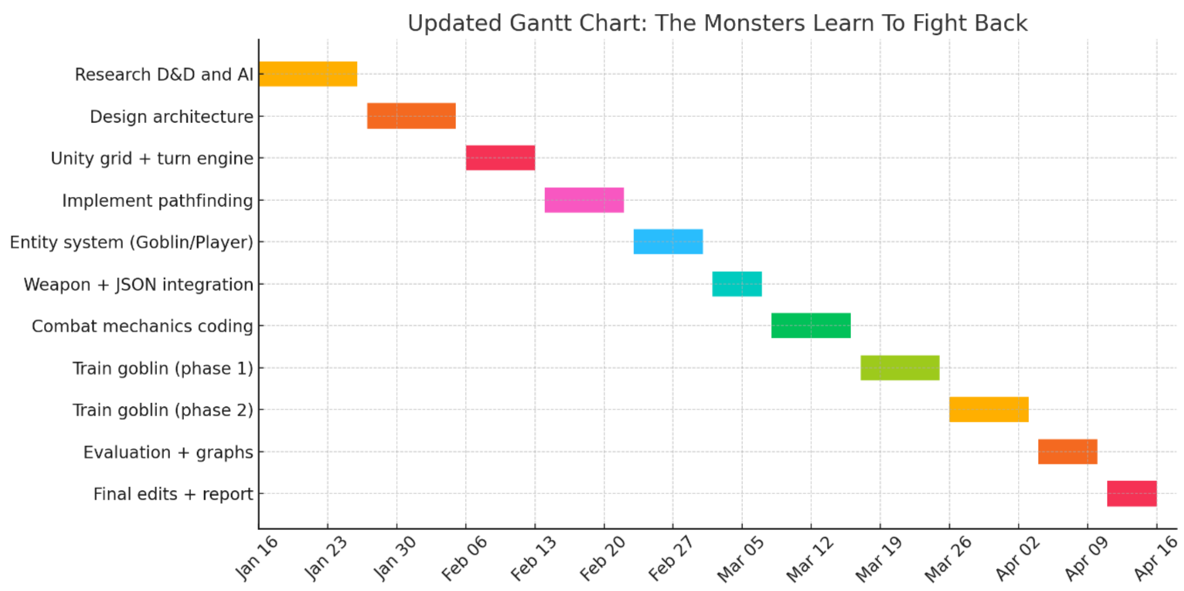


Figure 10 - Updated Gantt Chart

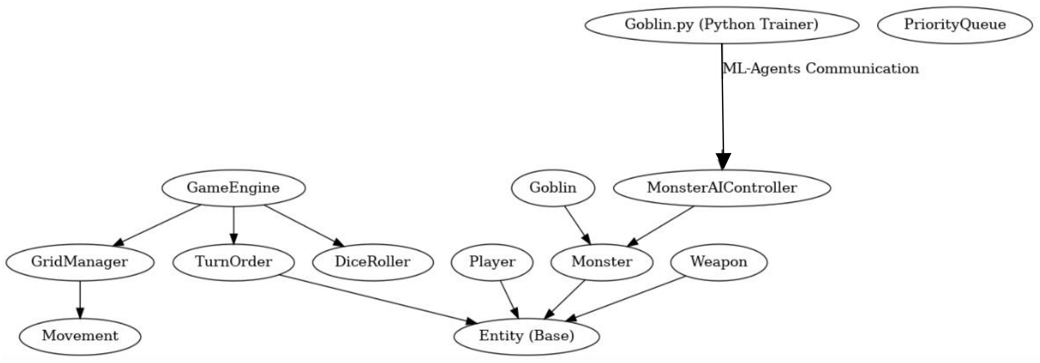
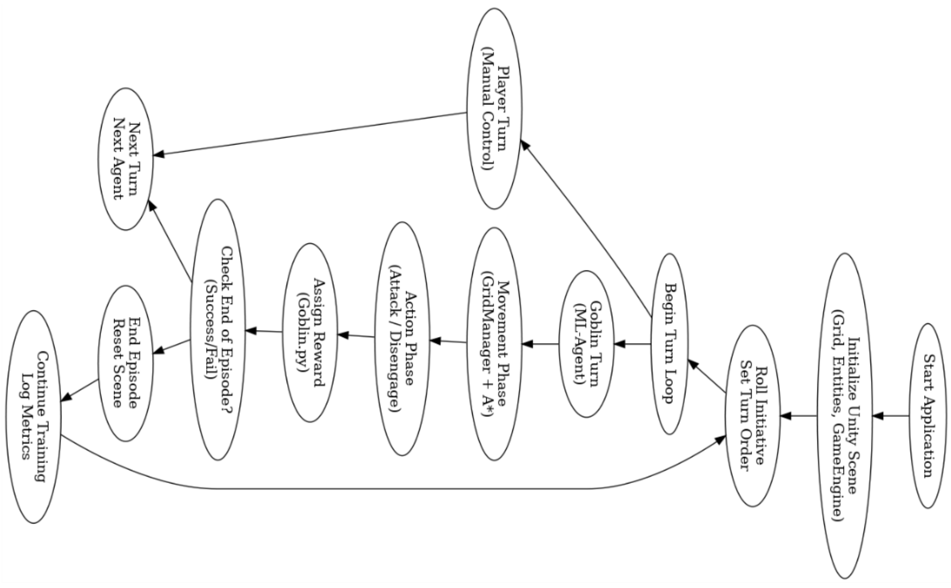


Figure 11 - Class Diagram



Start Application
Initialize Unity Scene (Grid, Entities, GameEngine)
Roll Initiative Set Turn Order
Begin Turn Loop
Player Turn (Manual Control)
Goblin Turn (ML-Agent)
Movement Phase (GridManager + A*)
Action Phase (Attack / Disengage)
Assign Reward (Goblin.py)
Check End of Episode? (Success/Fail)
Next Turn Next Agent
End Episode Reset Scene
Continue Training Log Metrics

Figure 12 - Flow chart of application process order work

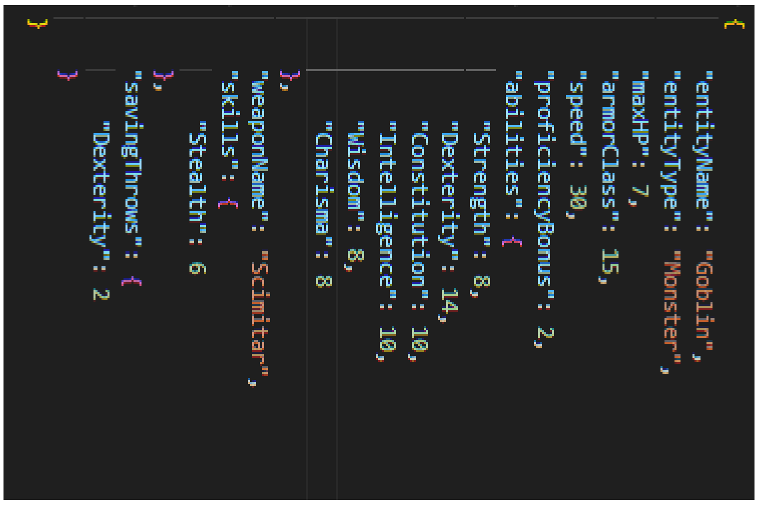
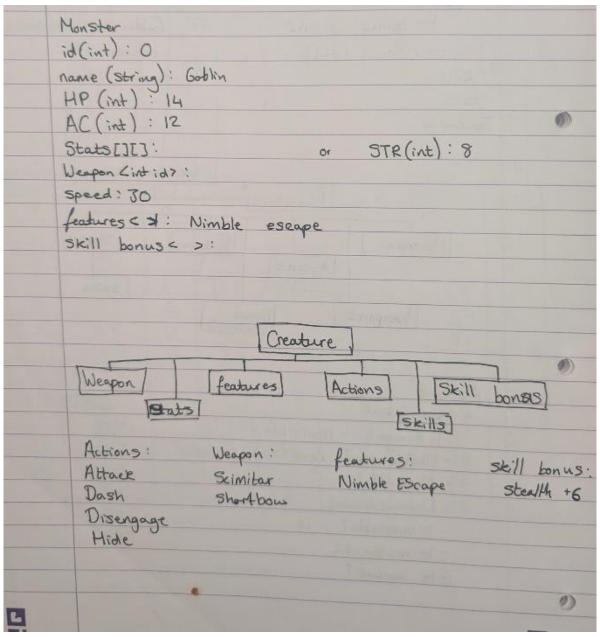

```
{
    "entityName": "Goblin",
    "entityType": "Monster",
    "maxHP": 7,
    "armorClass": 15,
    "speed": 30,
    "proficiencyBonus": 2,
    "abilities": {
        "Strength": 8,
        "Dexterity": 14,
        "Constitution": 10,
        "Intelligence": 10,
        "Wisdom": 8,
        "Charisma": 8
    },
    "weaponName": "Scimitar",
    "skills": {
        "Stealth": 6
    },
    "savingThrows": {
        "Dexterity": 2
    }
}
```

Figure 13 - Goblin JSON file

Appendix B: Tables of Figures and Tables

Figures

Tables



Monster
id(int) : 0
name (string) : Goblin
HP (int) : 14
AC (int) : 12
Stats[][] : or STR(int) : 8
Weapon <int id> :
speed : 30
features < ~~>~~ : Nimble eseape
skill bonus < > :

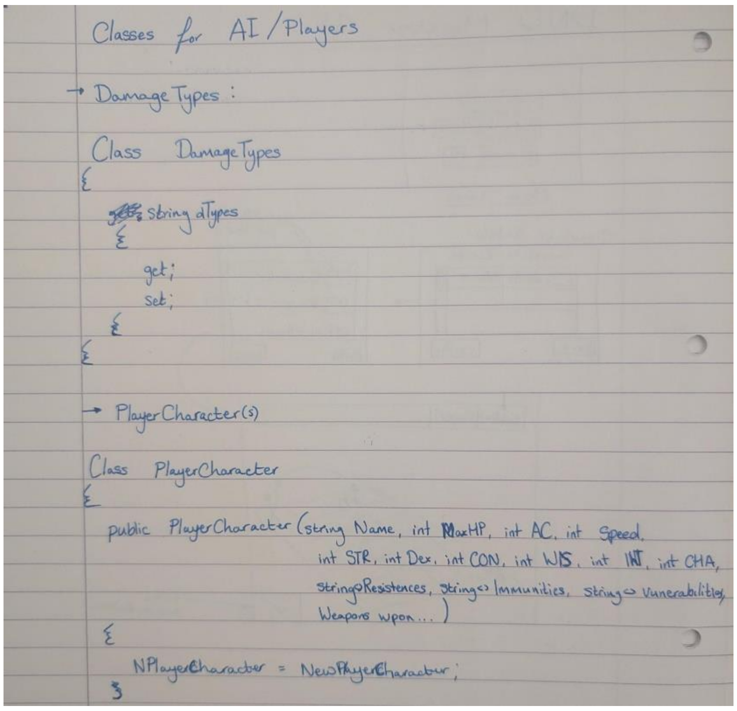
Actions :
Attack
Dash
Disengage
Hide

Weapon :
Scimitar
Shortbow

features :
Nimble EScape

skill bonus :
Stealth +6

Figure 14 - Monster Class Sketch



Classes for AI / Players

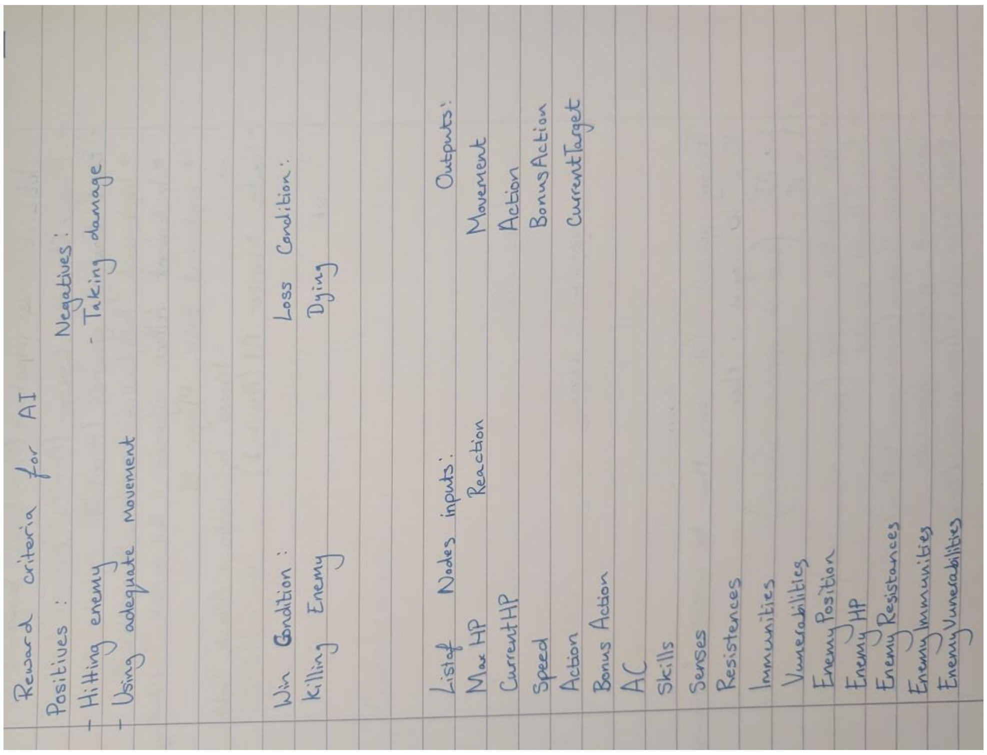
→ Damage Types :

```
Class DamageTypes
{
  ~~gets~~ string dTypes
  {
    get;
    set;
  {
{
```

→ PlayerCharacter(s)

```
Class PlayerCharacter
{
  public PlayerCharacter (string Name, int MaxHP, int AC, int Speed,
                          int STR, int Dex, int CON, int WIS, int INT, int CHA,
                          string<>Resistences, string<> Immunities, string<> Vunerabilities,
                          Weapons wpon... )
  {
    NPlayerCharacter = NewPlayerCharacter;
  }
```

Figure 15 - AI/Player Class Sketch



Reward criteria for AI

Positives:
- Hitting enemy
- Using adequate movement

Negatives:
- Taking damage

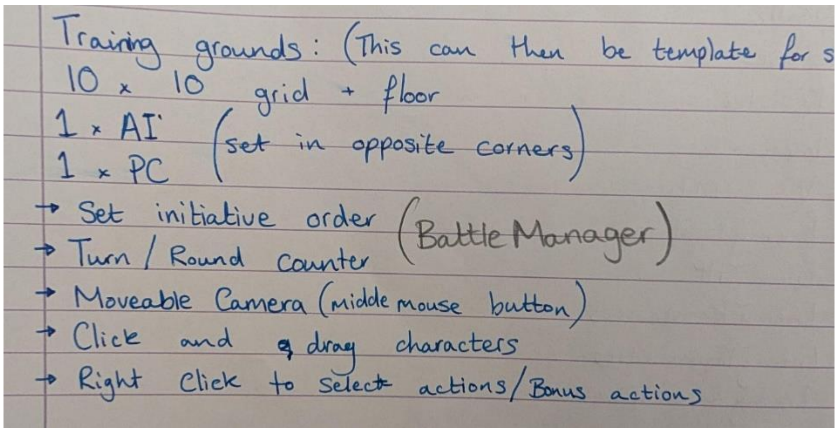
Win Condition:
Killing Enemy

Loss Condition:
Dying

List of Nodes inputs:
Max HP
Reaction
CurrentHP
Speed
Action
Bonus Action
AC
Skills
Senses
Resistences
Immunities
Vunerabilities
Enemy Position
Enemy HP
Enemy Resistances
Enemy Immunities
Enemy Vunerabilities

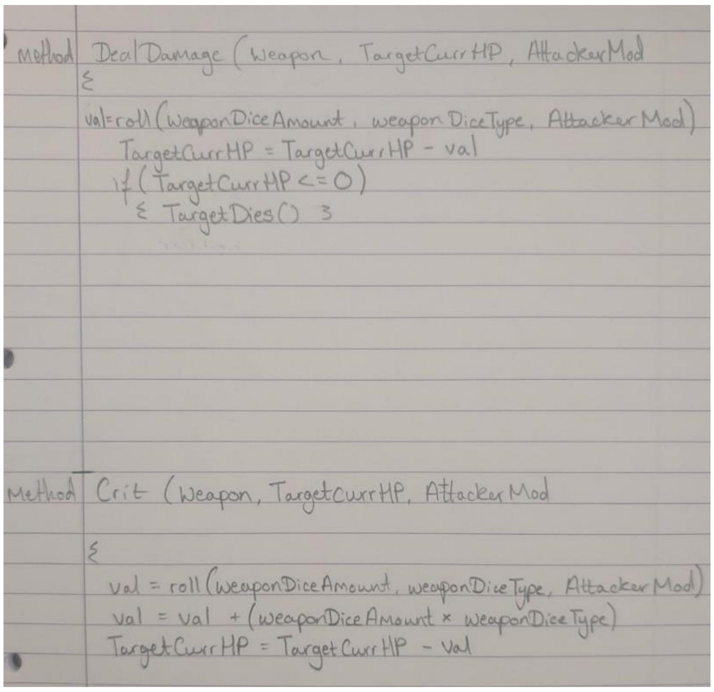
Outputs:
Movement
Action
BonusAction
CurrentTarget

Figure 16 - Reward and Observations for AI Sketch



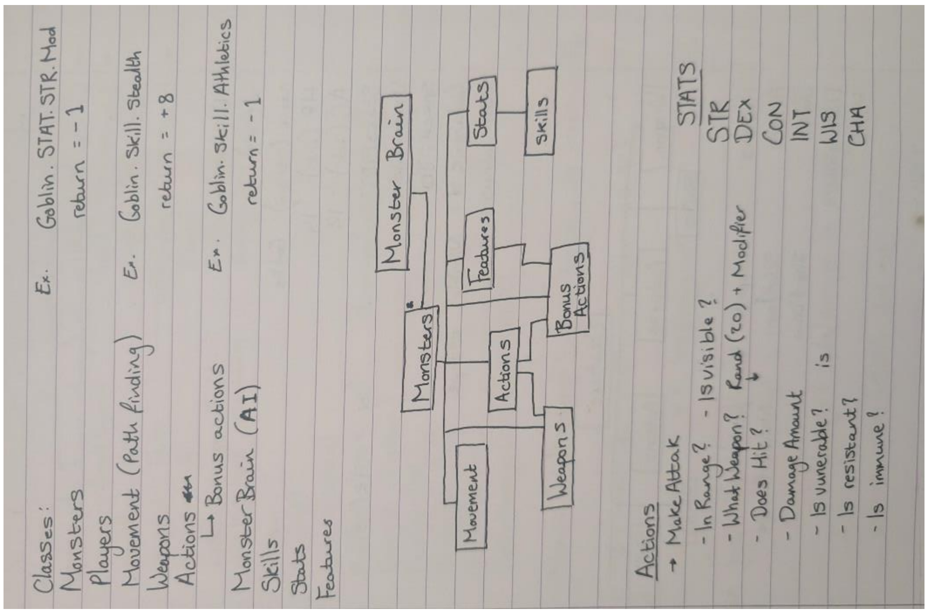
Training grounds: (This can then be template for s
10 x 10 grid + floor
1 x AI
1 x PC (set in opposite corners)
→ Set initiative order (Battle Manager)
→ Turn / Round counter
→ Moveable Camera (middle mouse button)
→ Click and drag characters
→ Right Click to Select actions/Bonus actions

Figure 17 - Training ground scene template sketch



```
Method  DealDamage (Weapon, TargetCurrHP, AttackerMod
        {
        val=roll(WeaponDiceAmount, weapon DiceType, Attacker Mod)
          TargetCurrHP = TargetCurrHP - val
          if (TargetCurrHP <= 0)
            { TargetDies() }

Method  Crit (Weapon, TargetCurrHP, AttackerMod

        {
          val = roll(weaponDiceAmount, weaponDiceType, AttackerMod)
          val = val + (weaponDiceAmount × weaponDiceType)
          TargetCurrHP = TargetCurrHP - Val
```

Figure 18 - Damage method sketch

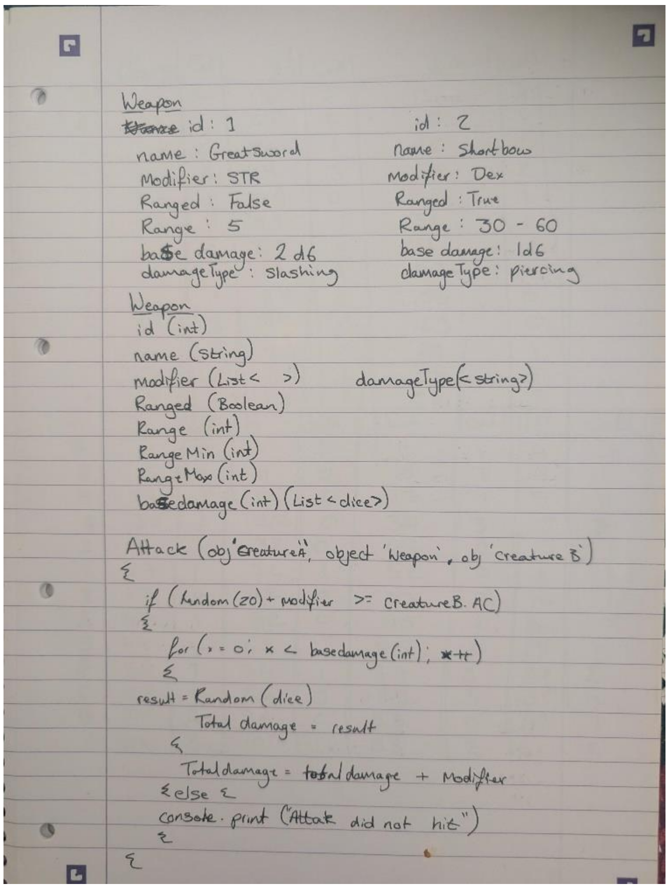


Classes: Ex. Goblin. STAT. STR. Mod
Monsters   return = -1
Players
Movement (Path finding)   Ex. Goblin. Skill. Stealth
Weapons   return = +8
Actions
↳ Bonus actions   Ex. Goblin. Skill. Athletics
Monster Brain (AI)   return = -1
Skills
Stats
Features

Actions
→ Make Attak
- In Range? - Is visible?
- What Weapon? Rand (20) + Modifier
- Does Hit?
- Damage Amount
- Is vunerable? is
- Is resistant?
- Is immune?

STATS
STR
DEX
CON
INT
WIS
CHA

Figure 19 - Class Diagram Sketch



```
Weapon
~~[illegible]~~ id : 1                      id : 2
  name : Greatsword                  name : Shortbow
  Modifier : STR                     Modifier : Dex
  Ranged : False                     Ranged : True
  Range : 5                          Range : 30 - 60
  base damage : 2 d6                 base damage : 1d6
  damageType : Slashing              damageType : piercing

 Weapon
  id (int)
  name (string)
  modifier (List<  >)          damageType(<string>)
  Ranged (Boolean)
  Range (int)
  Range Min (int)
  RangeMax (int)
  basedamage (int) (List<dice>)

Attack (obj 'CreatureA', object 'Weapon', obj 'Creature B')
{
  if (Random (20) + modifier >= CreatureB. AC)
  {
     for (x = 0; x < basedamage (int); x++)
     {
 result = Random (dice)
        Total damage = result
     {
       Total damage = total damage + Modifier
    } else {
    console. print ("Attack did not hit")
    }
}
```

Figure 20 - Weapon Class idea sketch