GViZDoom

A Benchmark for Generalization of FPS Games in Deep Reinforcement Learning

Tomilin, Tristan

Award date:
2021

Department of Mathematics and Computer Science

GViZDoom: A Benchmark for Generalization of FPS Games in Deep Reinforcement Learning

Master Thesis

Tristan Tomilin

Supervisor:
Dr. Meng Fang

Eindhoven, August 2021

Abstract

In this thesis, we introduce the GViZDoom Benchmark, a suite containing game-like simulation environments with levels of difficulty, designed to research and evaluate generalization in vision-based reinforcement learning (RL). The environments are situated in a semi-realistic 3D world, which the agent navigates and interacts with in the first person perspective. The notion of task difficulty is established in terms of visual modifications to the simulation environment. The benchmark is highly customizable by modifying the parameters and properties of difficulty attributes, and expandable by creating additional scenarios and tasks with novel objectives. We provide detailed experimental protocols for running several experiments using GViZDoom. We first empirically demonstrate that our proposed notion of difficulty, in terms of combining visual modifications of the environment, provides a sensible spectrum of complexity to an agent being trained on a state-of-the-art model-free value-based algorithm. We then use the benchmark to determine how a different number of training tasks impacts the generalizability of our agent. We further establish that GViZDoom is applicable for comparing RL methods of varying efficacy. We finally show, that an agent, trained on multiple low level tasks in parallel, has poor generalizability competence on unseen tasks with increased levels of difficulty. We believe this suite to be beneficial to the RL community, since previous benchmarks, such as Procgen and Arcade Learning Environment, have stimulated research by enabling researchers to shift their focus from designing appropriate environments to algorithmic advancements and novel techniques.

Preface

This master thesis signifies the conclusion of my Data Science in Engineering studies at Eindhoven University of Technology. I executed my graduation project internally within the Data Mining research group in the department of Mathematics and Computer Science, within a time span of nearly seven months over the period of 1st February 2021 to 27th August 2021. I learned a great deal more about reinforcement learning during this period, which even further amplified my fascination in this domain and sparked an interest in research altogether.

I would first of all like to thank my supervisor Dr. Meng Fang for being reachable at all times in providing guidance, feedback, suggestions, and inspiration. I would further like to show gratitude to PhD candidate Tianhong Dai at Imperial College London for assisting with the training and evaluating some of the models for the experiments in this thesis. Last, but not least, I would like to thank all my friends and family for unconditional moral support throughout my studies in the Netherlands.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Generalization is one of the most fundamental challenges in deep reinforcement learning (DRL). Where as humans are able to seamlessly generalize across similar tasks, this competence is predominantly absent in RL agents, who tend to become exceedingly specialized to the environments which they encounter during training. This problem makes DRL agents unreliable for real world applications where robustness is important. Thus, obtaining human-like skills is particularly important, considering the potential applicability of self-learning systems to real-world robotics applications [44]. Video games have progressively been used as DRL benchmarks during recent times. Great progress has been made regarding agent generalizability in the 2D game domain [15, 14, 58, 37, 50]. RL research directed at 3D environments, such as first-person shooter (FPS) games, have been attaining evermore focus in past years [9, 2, 33, 65, 23, 59, 48, 16]. There is yet, however, a lot to accomplish for developing competently generalizable agents, as the 3D setting provides more challenging control problems due to the vast complexity of effectively perceiving, interpreting, and learning the game environment. Arguably, targeting generalization is necessary in order to make progress on artificial general intelligence (AGI), rather than just solving individual problems.

FPS games directly simulate reality as humans perceive it (in a first-person perspective) [66]. Mastering such games requires the agent to possess human-like attributes, as the game sets novel and competitive goals, such as navigation, localization, memory, self-awareness, exploration, and precision [79]. In order to surmount the continually varying objectives and situations, the agent has to make tactical and strategic decisions on which courses of action to take. The overall managing of such skills is much desired for general intelligence, as it enables the AI to thrive in conditions it has not yet encountered.

An RL agent inevitably needs to encounter a variety of situations in order to acquire a concept of universality. Moreover, to properly assess such acquired competence, the trained agent should be tested in an unfamiliar setting. Hence,

one of the key challenges of developing agents to prevail in unencountered situations, is the design of suitable training and evaluation environments, which sufficiently differ from one another, while maintaining some necessary degrees of similarity (e.g., the observation space and action space). RL agents, trained in a set of environments with limited diversity, tend to suffer from overfitting and fail to generalize to unseen testing environments [82, 15, 14]. To resolve this issue, previous works have explored data augmentation techniques to increase the data diversity [22, 43, 83, 45, 46]. These approaches, however, merely tend to locally perturb the observations regardless of the training environments [74], showing limited effectiveness on enhancing the data diversity and the generalization performance. To better cultivate the ability of generalization, one common practice is to employ the agent to learn from multiple tasks in parallel [21, 40, 80], known as multi-task learning [8]. Another option is to train the agent on a set of progressively more challenging tasks, known as curriculum learning, which is is anything but new [51, 5, 78, 55, 36], having had much focus within the broader machine learning context. In this thesis, we design the **GViZDoom** benchmark which enables the use of both training practices.

Proper benchmark functions are crucial for the improvement of RL agents, as they provide suitable means of effortlessly comparing the performance of different methods and techniques. A growing trend in RL generalizability research is procedurally generating new environment content [32] which the agent has not yet encountered. However, we speculate that randomly created environments may not offer the best consistency in terms of accurately comparing agents. We hence propose a benchmark consisting of manually designed 3D environments with a multitude of tasks, adhering to a coherent notion of difficulty across multiple levels to overcome the limitations of previous game-based AI benchmarks, offering a broad and deep challenge. It is vital to note that difficulty is expressed in visual terms within the game, since the agent only receives raw pixels as input. Previous benchmarks [80, 50] primarily define difficulty as an implicit property of the game's mechanics [14, 6], the reasoning capacity required [35], or the relevant competencies necessary to reach an objective [11]. This may be proper for assessing a human player, but we speculate that this does not offer the current state-of-the-art RL agents sufficient leverage for superior generalization, due to the vast difference of how neural networks and human cognition function. Every modification of an environment in our benchmark (e.g., applying noisy textures, or changing the size and shape of enemies) has a visual disparity, and thus directly grants the agent new input to aid in learning generalization.

1.1 Problem Statement

The main goal of this thesis is to design and implement a novel RL benchmark with difficulty levels for researching and evaluating the agents generalization in FPS games. We state the following objectives to achieving this end:

- Design a set of easily configurable and expandable simulation environments,

which require different competencies of mastering FPS games

- Implement a notion of measurable progressive difficulty in terms of visual depiction in the simulation environments
- Determine the appropriate number of parallel training environments in the benchmark to acquire the best means of generalization
- Demonstrate the viability of the benchmark for evaluating DRL methods of different efficacy

1.2 Thesis Structure

We hereby present the structure of this thesis. Chapter 2 outlines the relevant prior research conducted in the domain of generalizable DRL agents, and describes how this thesis complements the previous work. In Chapter 3, we propose the GViZDoom benchmark, comprised of four simulation environments with numerous subtasks of varying complexity. Chapter 4 provides essential theoretical background of DRL methods and techniques for training the agents in our experiments. In Chapter 5, we describe the experiments conducted on the benchmark. We then present and discuss their results. Finally, Chapter 6 concludes our work, states the accomplishments, outlines the deficiencies, and proposes further possible directions for future research.

Chapter 2

Related Work

A great number of RL benchmarks and research platforms tackling generalization in various ways having emerged during recent years, e.g., improving generalizability from the training tasks to inference tasks by parametrizing synthetic scenes [38], or domain randomization [72, 60, 52]. In this chapter we describe various existing platforms, benchmarks, and other RL generalization research related to our work.

2.1 2D Benchmarks and Platforms

2D simulation environments are well suited and have been widely used for researching and benchmarking RL techniques and methods, as they generally provide a modest observation space and dense rewards.

2.1.1 Arcade Learning Environment

The Arcade Learning Environment (ALE) [4] has long served as a gold standard in RL, with the diversity between games being one of its greatest strengths. The benchmark has further been extended [50] to support a form of stochasticity called sticky actions, and combinations of game modes and difficulty levels called flavors. A select subset of games and modes from this extension have been used to evaluate generalization properties of DQN [17]. The game modes, however, are not ranked in terms of complexity, and there is little overlap in terms of characteristics. Further, the formulation of difficulty widely differs between environments, as the games are of very different nature. We design a more abundant set of attributes for each scenario, which determine the game difficulty. Several of these factors are present across multiple environments. Moreover, combining a set of attributes of a game scenario provide a coherent mechanism for determining the difficulty level, as the agent is faced with more complexity.

2.1.2 Sonic

The Sonic benchmark [57] separates levels of the Sonic the Hedgehog™video game into training and holdout sets for measuring generalization in RL. However, due to a limited number of training levels and difficulties in measuring progress, RL agents did not succeed in proper generalization. In our solution, we emphasize on designing an abundant suite of scenarios with subtasks, having a definite success metric for each.

2.1.3 BabyAI

The BabyAI platform [11] comprises an extensible suite of 19 scenarios of increasing difficulty for grounded language learning. The levels are built in MiniGrid [12], a partially observable 2D gridworld environment. They design a set of informally defined competencies, which specify what an agent should be capable of. The scenarios are built by selecting a subset of competencies necessary for each level. The missions generated for a level are thus only solvable by an agent possessing the select competencies. We follow a similar approach in designing our level structure, in which difficulty increases with the competencies required from the agent. However, in our solution, complexity is expressed directly in terms of the input to the agent via visual information from the 3D environment. Moreover, they empirically determine the difficulty of a scenario based on experimental results, whereas we defined it as the number of attributes expressing complexity. We further group tasks of scenarios with the same number of attributes together into levels of difficulty.

2.1.4 Safety Gym

Safety Gym [30] provides a suite of 18 high-dimensional continuous control environments for studying safe exploration and constrained RL. All Safety Gym environments are comprised of comprehensive randomization to prevent agents from overfitting to particular environment layouts. These environments thus enforce a need for generalization, although it is not explicitly addressed within the benchmark. While the environments do have a high level of randomization, the primary objective is to enforce safety and to minimize the constraint cost.

2.1.5 CoinRun

CoinRun [15] is a procedurally generated environment designed as a benchmark for generalization in RL. The authors observe the extent to which agents can overfit to a fixed training set, reporting that the number of training environments required for good generalization is much larger than the number used by prior work on transfer in RL. We run a similar experiment on our benchmark in Section 5.2.2, investigating how a different number of training tasks affect the generalization ability of the agent in a 3D environment. They also demonstrate that deeper convolutional architectures and forms of regularization significantly

improve generalization, as do methods traditionally found in supervised learning, including L2 regularization, dropout, data augmentation and batch normalization.

The paper proposes a metric to quantify overfitting by measuring the percentage of levels solved between training and testing. Not all environments in the GViZDoom benchmark have a clear notion of solvability. Moreover, the scoring metric for evaluating an agent varies according to the objective and nature of the scenario. We therefore heuristically set an upper bound on the score to each scenario, and introduce an indicator of performance, which measures the percentage of success according to the upper bound of each individual task from a scenario. The performance indicators for training and testing across all environments can then likewise be compared to assess generalization and overfitting.

2.1.6 Animal-AI

The Animal-AI Environment [6] utilizes tasks inspired by the extensive literature on animal cognition to evaluate the generalizability of the agent on unseen test tasks. It keeps all the positive elements of standard gaming environments, but is explicitly designed to contain only the necessary ingredients for performing cognitive testing built up from perception, navigation, and animal-like artificial cognition. The environment has a deterministic state transition function based on a simulated physics engine. The authors found that satisfactory performance depends on generalizing well from particular training configurations. They further state that using a single unified environment makes the prospect of generalization substantially more tenable. The experimental results indicate that there are many tasks in the environment, that some animals can solve that were considered too complex for the agents to solve.

2.1.7 Meta-World

Meta-World [80] explores how existing state-of-the-art meta-learning algorithms can quickly learn qualitatively new tasks when meta-trained on a sufficiently broad and structured task distribution of up to 50 unique continuous control environments, which are split up into training and testing tasks. The shared physics and mechanics between the environments give rise to the conceivable expectation of generalization, given that the specificalities of test tasks are novel to the agent.

2.1.8 Procgen

The Procgen Benchmark [14] consists of 16 unique environments designed to measure both sample efficiency and generalization in RL. The benchmark is specifically designed to evaluate generalization, since distinct training and test sets can be generated in each environment. It is also well-suited to evaluate

sample efficiency, since all environments pose diverse and compelling challenges for RL agents. The intrinsic diversity of the environments demands the agents to learn robust policies; overfitting to narrow regions in state space will not suffice, i.e., the generalization ability becomes an integral part of success when agents are faced with constantly changing levels.

2.2 3D Benchmarks and Platforms

As DRL methods have significantly improved during recent years, 3D environments have become more comprehensible to agents, and thus feasible to be employed as research platforms and test-beds. The motivation behind using 3D environments is much higher, as they provide more challenging control problems and direct applicability of self-learning systems to real-world tasks and robotics applications. Moreover, they subtly pave the road towards AGI.

2.2.1 DeepMind Lab

DeepMind Lab [2] is an extensible first-person game platform designed for research and development of AGI and ML systems in a 3D world with rich science fiction visuals and simulated real-world physics. DeepMind Lab facilitates the development of a wide range of creative tasks, navigational challenges, environments, intelligence tests on visual cues, and can be used to study how autonomous RL agents can learn complex tasks in a large, partially observed, and visually diverse worlds. It is built upon Quake III Arena [29] from id Software. Agents interact with the game via an RL API, which has been built on top of the game engine, and provides the agents with complex observations and accepts a variety of actions. We adopt the ViZDoom platform for out benchmark, which is faster (up to 7000 frames per second on modern computers), more lightweight, and highly customizable via a convenient mechanism of user scenarios, enables off-screen rendering and frame skipping, and provides access to the renderer's depth buffer [39].

2.2.2 Project Malmo

The Project Malmo platform [33] is based on the popular block-based 3D computer game Minecraft, where it is possible to task the agent with objectives ranging from navigation and survival to collaboration and problem solving. The open-world nature of Minecraft provides a convenient platform for exploring RL and AI [20], as the environment is open, dynamic, complex, and diverse. The platform thus provides abundant opportunities for creating tasks and scenarios.

2.2.3 Obstacle Tower

The Obstacle Tower benchmark [35] is a high fidelity procedurally generated 3D environment in the third-person perspective, where agents are required to

solve both low-level control and high-level planning problems. The environment consists of up to 100 floors, each of which can contain a puzzle to solve, enemies to defeat, obstacles to evade, or a key to open a locked door. The environment has high visual fidelity, procedurally generated floors, rooms, and visuals, and a physics-driven interaction. Similarly to our benchmark, Obstacle Tower has a progressive notion of difficulty, since the contents and layout of the rooms within each floor becomes more complex at higher floors. However, Obstacle Tower was specifically developed to offer a broad and deep challenge, the solving of which would require major advancements in RL. Our benchmark is designed to provide a wider spectrum of complexity, meaning that lower level tasks are easily solvable by current RL methods, whereas we leave open the possibility to further combine difficulty attributes to create higher level tasks, which provide a greater challenge. Moreover, GViZDoom functions from the first-person perspective, which has more tangible implications to real world problems and applications.

2.2.4 MazeExplorer

MazeExplorer [23] is a customizable 3D benchmark based on the ViZDoom platform for assessing generalization in terms of navigation and exploration. It uses domain randomization, which we have likewise adopted in some aspects of the GViZDoom environments. One of the main differences of our benchmark and MazeExplorer is the usage of procedurally generated content (PCG). Further, MazeExplorer only endorses competencies of navigation and localization, whereas the scenarios of our benchmark are not constrained in this regard. GViZDoom thrives to cultivate and assess all the necessary skills required to master the game of Doom.

2.2.5 CRLMaze

CRLMaze [48] is a benchmark for assessing continual RL techniques in a complex non-stationary 3D object-picking task based on ViZDoom, subject to constant environmental changes. The task consists of learning how to navigate in a complex maze, and pick up a specific type of objects while avoiding another. Similarly to the first iteration of our benchmark, it consists of 4 scenarios. The CRLMaze task is very similar to the *Poison* task in the *Health Gathering* scenario in our benchmark. GViZDoom includes a wider variety of targeted competencies, environments, difficulty levels, and tasks, which are subject to modifications, and therefore provides a more reliable assessment of any RL techniques.

2.3 Procedurally Generated Content

Numerous standardized game environment benchmarks with fixed structures [4, 30, 80] often lack an explicit split between the training and testing phases, when evaluating the generalizability of a model. To overcome this issue, recent works [59, 58, 15, 23, 14] use environments that utilize different approaches

of PCG [32]. The approach of Progressive PCG [37] has been employed to demonstrate that dynamically adapting the level difficulty during training allows the agent to solve more complex levels, rather than training it on the most difficult levels directly, as agents tend to heavily overfit to the specific training set. It has further been shown [10], that adding surprise minimizing information through rewards learned by the density model on states of the training history can improve generalization on PCG game levels that are unseen during training.

PCG enables the algorithmic creation of a near-infinite supply of highly randomized levels and content [62]. Every such level exhibits a unique configuration of underlying factors of variation, such as layout, positions of game entities, asset appearances, or even different rules governing environment transitions. PCG further enables the model to be trained and evaluated on a distinct seed, which defines a unique instance of an environment for each episode, thus guaranteeing an unequivocal split, and being suitable for evaluating systematic generalization in RL. In the GViZDoom Benchmark, however, we default to use a manual approach for level design. We find that unconstrained PCG causes too much randomness in our environments, such that the difficulty of a given level will no longer be properly ascertainable. Moreover, generating a random map layout using PCG may too heavily impact the root essence of a scenario, thus creating a high variance of complexity between different runs, or making the objective outright unsolvable. We do however use domain randomization to the extent of some scenario-specific characteristics and certain task-specific difficulty attributes (e.g., enemy size, agent height, and respawn intervals).

Chapter 3

GViZDoom Benchmark

The initial version of the GViZDoom benchmark consists of four manually designed environments, each with subtasks of progressing levels of difficulty. Every scenario is designed with a particular narrow objective in terms of FPS game complexity, nevertheless establishing novel skill requirements. This set of competencies are commonplace for human players to effectively master FPS games. The general properties of scenarios are described in Table 3.1. Each environment of the benchmark is built upon an original map from ViZDoom [39], a flexible AI research platform for RL from raw visual information, based on the classical FPS video game, Doom [28].

In this chapter, we describe the problem of generalization on different environments, and how the GViZDoom benchmark contributes to solving this conundrum. We further outline the designed scenarios and their subtasks, along with the configurable parameters, provided state spaces, action spaces, and reward functions.

3.1 Environment

We denote an environment as E_i, each of which consists of a number of subtasks $\mathcal{T}_i$. For a given environment, we train a policy π parameterized by θ on a set of predefined training tasks $\mathcal{T}_i^{tr} \subset \mathcal{T}_i$, and evaluate the policy on a set of test tasks $\mathcal{T}_i^{te} \subset \mathcal{T}_i$, such that $\mathcal{T}_i^{tr} \cap \mathcal{T}_i^{te} = \varnothing$. For every environment E_i, we define a performance indicator $score_i$, which aligns with the primary objective of the scenario (see Table 3.1). A higher metric indicates superior performance. The objective is to train the policy π_θ on the set of training tasks $\mathcal{T}_i^{tr}$ on environment E_i, such that the agent then achieves a maximal score averaged across the set of evaluation tasks $\mathcal{T}_i^{te}$, which we measure as the performance indicator P_i:

$$P_i(\pi_\theta) = \max_\theta \left(\frac{1}{|\mathcal{T}_i^{te}|} \sum_{t \in \mathcal{T}_i^{te}} \mathbf{score}_i\Big(t(\pi_\theta)\Big) \right), \tag{3.1}$$

where i is the environment index. In order to assess the overall performance of an agent on multiple scenarios with varying score functions and evaluation metrics, we assign an upper bound $score_{i,max}$ to each metric as the best possible achievable score in every environment E_i. For scenarios with the objective of survival, this bound is set as the number of frames after the episode is automatically terminated, and for the enemy elimination tasks, we heuristically set it as the maximal number of kills the agent could execute within a single episode. For the *Seek and Kill* scenario, this value is $score_{max} = 20$. We then use these upper bounds to normalize each evaluation score $P_{i,norm}$, and find the mean of performance indicators:

$$P_{norm} = \frac{1}{|E|} \sum_{i=0}^{|E|} \frac{P_i}{score_{i,max}}, \tag{3.2}$$

where $|E|$ is the number of evaluated environments. The mean performance indicator will thus be a value in the range of $P = (0, 1]$, where a value close to 1 indicates that the agent has reached near-optimal performance on the evaluated of all used scenarios, and a value close to 0 signifies a complete lack of acquired competence. In our experiments we report the values as percentages for better comprehensibility.

In order to represent policies for multiple tasks using a single model, we ensure that the observation and action spaces contain significant similarities in structure between tasks [80]. Most attributes of a scenario (e.g., observation space, action space, rewards) can be modified with little effort. Moreover, the benchmark is designed to enable the introduction of new scenarios and subtasks in a straightforward manner, granted by the modular architecture.

Table 3.1: Scenario properties

Scenario	Objective (Success Metric)	Action Space	Game Variables	Episode Timeout (Frames)	Enemies	Weapon
Defend the Center	Survival (Frames Alive)	ATTACK, TURN_LEFT, TURN_RIGHT	KILLCOUNT, AMMO2, HEALTH	1100	Yes	Yes
Health Gathering	Survival (Frames Alive)	MOVE_FORWARD, TURN_LEFT, TURN_RIGHT	HEALTH	2100	No	No
Seek and Kill	Enemy Elimination (Kill Count)	ATTACK, SPEED, MOVE_FORWARD, TURN_LEFT, TURN_RIGHT	KILLCOUNT, AMMO2, HEALTH, POSITION_X, POSITION_Y	1200	Yes	Yes
Dodge Projectiles	Survival (Frames Alive)	SPEED, MOVE_LEFT, MOVE_RIGHT	HEALTH	2100	Yes	No

3.1.1 Observation Space

Doom is by far not a complete information game, since at a single point in time, the agent can spatially occupy only one location of the entire environment and observe a portion of its surroundings. More precisely, we use a 16:9 in-game resolution, which grants a 108 degree field of view (FoV). Alternatively, a 90 degree FoV is obtainable by applying a 4:3 resolution [44]. We define the observation space $\mathcal{O}$ of every environment as a rendered pixel image of the environment from a first person perspective. This image is rendered in a 640×480 resolution with 3-channels of 8-bit values in RGB, and is downscaled to a 84×84 pixel grayscale image during preprocessing. Using RGB images as input to the model is computationally more expensive, but crucial in our benchmark, since color conveys vital information to the agent (e.g., making it more feasible to distinguish beneficial items from harmful ones, and separate enemies from textures or decorations).

3.1.2 Action Space

To speed up the training process and reduce unnecessary overhead of redundant operations, we restrict the executable actions to ones which are most essential to a given environment in terms of achieving adequate performance. We use a discrete action space $\mathcal{A}$ which varies across scenarios, but remains fixed among tasks within one. The available actions in each environment are shown in Table 3.1.

3.1.3 Reward Function

The reward function $\mathcal{R}$ of each scenario is comprised of different components, contingent on the characteristics of the environment, and remains identical across subtasks. After every game iteration, each scenario in our benchmark returns a pre-configured list of in-game variables presented in Table 3.1, which are used for calculating the reward. The precise function to this end is described in detail under each scenario in Section 3.4.

3.1.4 Specifications

The environments are of pseudorandom nature, which manifests in the randomized behaviour of enemies, fluctuating damage inflicted by attacks, and the spawning locations of items, enemies, and the agent. All enemies in our scenarios are configured to have 1 health by default, thus it only requires the agent to fire a single gun shot to eliminate them. The weapon[1] and heads-up display (HUD) are visually rendered in the game by default, whereas the crosshair, particles and decals (materials projected onto existing surfaces) are not, if not specified otherwise. Each scenario is automatically terminated after a predetermined

[1]The weapon is only rendered in environments in which the agent is granted a weapon (see Table 3.1)

number of frames. If the objective of a given scenario is survival, then we consider the environment *solved*, if an agent subsists the corresponding number of frames for 10 consecutive episodes. The *Health Gathering* and *Seek and Kill* scenarios have multiple spawn points to endorse a wider variety of experience that the agent could gather. In these environments, the agent is spawned in one such predetermined locations, which is sampled uniformly at the beginning of each new episode.

3.2 Difficulty Levels

We assign a difficulty level $d \in \{0, 1, \ldots, 4\}$ to each task $t \in \mathcal{T}_i$. We further define the set of all tasks of difficulty d in an environment E_i as $\mathcal{T}_{i,d} \subset \mathcal{T}_i$. The level of a task is determined by difficulty attributes in said task. The *default* task of every scenario, is level $d = 0$ with no added complexity. Level 1 tasks include a single modification, whereas higher level tasks are comprised of a combination of such variations. The difficulty attributes thus overlap across levels. A level is in accordance with the number of variations it is comprised of (e.g., a level 3 task has three types of modifications). However, in the initial iteration of the benchmark we consider the task including all scenario specific difficulty attributes to be of level 4. We make this exception to avoid an unnecessary number of levels, because our experiments in the later chapter of this thesis indicate that the performance difference above three or more difficulty attributes is negligible when training or evaluating the Rainbow agent, a current state-of-the-art value-based DRL method. There is no level gap in the benchmark, as the scenarios could be modified in numerous additional ways, nor is the number of tasks per level fixed.

The modified characteristics of a scenario are not targeted on increasing implicit in-game difficulty, as it is traditionally implemented in FPS games, but changing the visual appearance of the environment. We regard this approach of adding complexity essential for evaluating DRL algorithms, since the agent only receives raw pixels as input to its decision making. Hence, the agent is granted the opportunity to learn from the new visual information for superior generalizability in unencountered environments. The difficulty attributes across environments primarily include the following:

- Introducing new enemy and item types
- Rendering enemies and items in a different size, or style
- Applying noisy textures, which increase the challenge of distinguishing relevant enemies or items from the background
- Adding decorations to the environment, which either act as obstacles by hindering the navigation of the agent, or confuse the agent as being potential relevant targets

Increasingly combining these modifications hence forms a progression of difficulty levels, established in a similar fashion to the competencies in the work by Chevalier et al. [11]. To master higher level tasks without previously having encountered them, the agent needs to have learned a policy that is robust across all axes of variation. Screenshots of the default and seven level 1 tasks of every scenario are presented in Table 3.2. All task with their difficulty attributes are presented in Tables A.1, A.2, A.3, and A.4.

To provide an evident recognition of tasks in literature, we adopt an evidently discernible nomenclature. The name of a level 1 task represents the difficulty attribute it incorporates. The names of tasks may thus overlap across scenarios. Naming higher levels tasks, we concatenate the difficulty attributes which the task is comprised of. We consider this method feasible for the initial iteration of the benchmark, as there are not an inconceivable number of levels which would excessively lengthen the names of tasks.

Doom additionally includes a configurable in-game difficulty setting, which determines the speed and aggressiveness of enemies, the damage factor of the player, and further characteristics, which are not relevant to our environments. We set this parameter to a value of 3 from the range of 1-5 for all scenarios.

3.3 Implementation

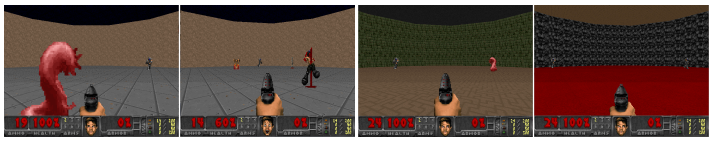
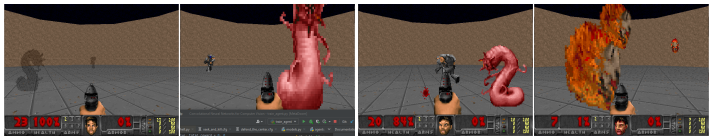
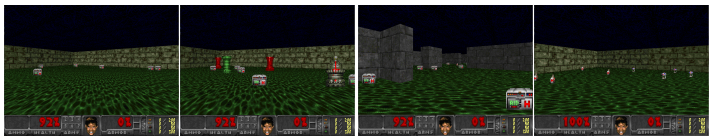
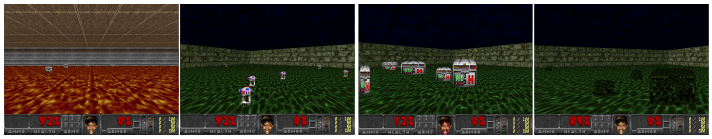
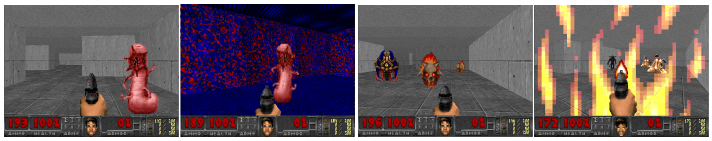
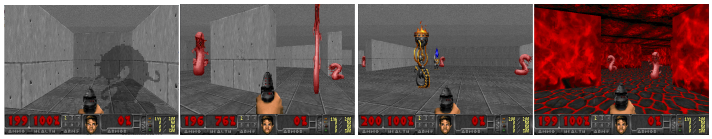
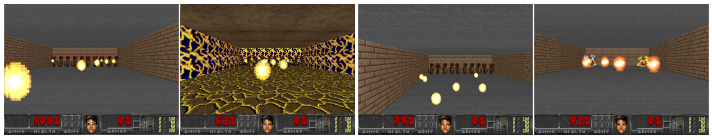
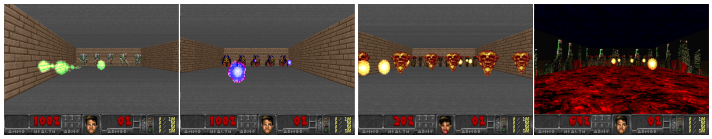
We use the map editing tool *SLADE3* [34] to modify the *Internal WAD*[2] (IWAD) files of the original scenarios from *ViZDoom*. We prepare an independent IWAD for each task of a scenario. To achieve our desired environment variations, we modify two key components of the IWAD. To change textures, place initial starting items in the environment, add decorations, and set agent spawn points, we adjust the *TextMap* file, composed in Universal Doom Map Format (UDMF), a specification for laying out Doom maps in a textual way. SLADE includes a built-in user interface to provide a more comprehensive manner of adjusting such attributes. To customize the in-game properties and subroutines of a scenario (e.g., enemy characteristics and behaviour, spawning intervals, rendering properties, player attributes), we modify the *Scripts* file in the IWAD, composed in Action Script Code (ACS), a miniature programming language, structured much like C/C++.

3.4 Scenarios

In this section we describe the four scenarios in the initial version of the GViZ-Doom benchmark. We outline the core objectives, and provide the motivation behind the design in terms of what relevance each scenario bears in relation to

[2]A WAD file is a game data file used by FPS games running on the original Doom engine. It contains data such as sprites (graphics), level information, and items.

Table 3.2: Screenshots of level 0 and 1 tasks

Scenario	Tasks			
Defend the Center		Default	Gore	Mossy Bricks	Stone Wall
		Fuzzy Enemies	Resized Enemies	Fast Enemies	Flying Enemies
Health Gathering		Default	Obstacles	Supreme	Poison
		Lava	Stimpacks	Resized Kits	Shaded Kits
Seek and Kill		Default	Blue	Mixed Enemies	Invulnerable
		Shadows	Resized Enemies	Obstacles	Red
Dodge Projectiles		Default	Flames	Resized Agent	Mancubus
		Barons	Cacodemons	Flaming Skulls	City

FPS games in general.

Every environment is an extension of an original scenario in the *ViZDoom* [39] platform. The subtasks of each are described in Appendix B. We define the *default* task by outlining the modifications to the corresponding original scenario. All other tasks are derived from the *default* task.

3.4.1 Defend the Center

In this scenario[3], the agent is positioned in the center of a circular room. Enemies are spawned at fixed positions alongside the wall of the area. Once they are eliminated, they respawn at their original location after a fixed time delay. The enemies do not possess a projectile attack and therefore have to make their way within melee range of the agent to inflict damage. The agent is rendered immobile, being equipped with a weapon and a limited amount of ammunition to fend off the approaching enemies.

The core objective of the agent is to maximize its survival, the success of which is measured by how many frames an episode lasts. The episode ends when the health of the agent reaches 0. This scenario assesses the aiming accuracy and enemy detection ability of the agent, which are an integral part of FPS games. The agent ought to ideally make a tactical trade-off in terms of prioritization. Shooting enemies at close range has a higher likelihood of a successful shot, which prevents them from getting close enough to inflict damage. To repel a maximum number of enclosing enemies, the agent should not unnecessarily waste the limited ammunition. The reward r_t of a single iteration is defined as follows:

$$r_t = \mathbb{1}\{k_t\} - \mathbb{1}\{d_t\} - 0.1\mathbb{1}\{m_t < m_{t-1}\} - 0.2\mathbb{1}\{h_t < h_{t-1}\}, \tag{3.3}$$

where k indicates a killed enemy, d indicates death, m is the remaining ammo, h is the remaining health, and t is the time step.

3.4.2 Health Gathering

In this scenario[4], the agent is spawned in an environment with an acid surface, which slowly, but constantly, inflicts damage, and decreases the agent's health. An episode of the game ends when the agent's health reaches 0. Different items, granting a varying amount of health, spawn in random locations at specified time intervals. The default health item is a *MediKit*, which heals the agent for 25 hit points. Certain sub-tasks also include poison vials as a complexity attribute, which inflict damage to the agent instead of providing health.

This scenario relates to FPS games in terms of locating and collecting relevant items in the environment with optimal navigation abilities. The agent's objective

[3]Demo available at https://www.youtube.com/watch?v=hbGCBlBNUik
[4]Demo available at https://www.youtube.com/watch?v=13-rBxaoWFE

is to stay alive a maximal number of frames. The agent should thus identify the items which grant health and navigate around the map to collect these items quickly enough to avoid running out of health, while averting damage inflicting items. The agent is granted a small reward for every frame it manages to survive. The reward r_t of a single iteration is defined as follows:

$$r_t = 0.01 - \mathbb{1}\{d_t\} - \mathbb{1}\{p_t\} + \mathbb{1}\{h_t > h_{t-1}\}, \tag{3.4}$$

where p indicates whether poison was picked up, d indicates death, h is the remaining health, and t is the time step.

3.4.3 Seek and Kill

Seek and Kill is based on the *health_gathering_supreme.wad* scenario[5] from the *ViZDoom* platform. In this scenario, the agent is randomly spawned in one of the 20 predefined locations within a maze-like environment, and equipped with a weapon and unlimited ammunition. A fixed number of enemies are spawned at random locations at the beginning of an episode. Additional enemies are spawned at random unoccupied locations after a specified time interval. The enemies are rendered immobile, forcing them to remain at their fixed locations. This demands the agent to locate them by navigating around the maze.

The goal of the agent in this scenario is to maximize the number of enemies it is able to locate and kill in a limited time span. The agent is expected to learn how to efficiently navigate the environment and identify enemies, which is a prevalent factor in FPS games. In order to encourage traversal of the maze, the agent is granted an additional linearly scaling reward for how far it has relocated relative to its previous position five frames ago. We define the reward function r_t of a single iteration as follows:

$$\begin{aligned} r_t = \mathbb{1}\{k_t\} - \mathbb{1}\{d_t\} - 0.1 \cdot \mathbb{1}\{m_t < m_{t-1}\} - 0.01 \\ -0.3 \cdot \mathbb{1}\{h_t > h_{t-1}\} + 0.005 \cdot \left\| l_t - l_{t-5} \right\|_2^2, \end{aligned} \tag{3.5}$$

where k indicates a killed enemy, d indicates death, m is the remaining ammo, h is the remaining health, l is the location and t is the time step.

3.4.4 Dodge Projectiles

This scenario[6] is a modified version of *defend_the_line.wad* from *ViZDoom*. The agent is positioned in one end of a rectangular room, facing the opposite wall. A number of immobile projectiles firing enemies are spawned at the other side of the area with equal distance from one another. The agent is not equipped with any weapon nor ammunition, and cannot look around. It can only move left or right. To balance the difficulty between tasks of the same difficulty level,

[5] Demo available at https://www.youtube.com/watch?v=6POupU974u4
[6] Demo available at https://www.youtube.com/watch?v=nh4_FW0ebRY

the damage factor of enemies is adjusted accordingly.

The goal of the agent in this scenario is to minimize taking damage and maximize its survival by evading incoming projectiles via lateral movement. The performance of the agent is measured by the number of frames in an episode. The reward r_t of a single iteration is defined as follows:

$$r_t = 0.01 - \mathbb{1}\{h_t < h_{t-1}\}, \tag{3.6}$$

where d indicates death, h is the remaining health, and t is the time step.

Chapter 4

Theoretical Background

In this chapter we describe the fundamental techniques for training a generalizable DRL agent to play Doom. The key idea is to apply image recognition on an agent's field of vision using a convolutional neural network (CNN), extract relevant features from the game frames and determine the next best course of action granting maximal utility. It is worth noting, that a naive reinforcement learning approach is applicable to all scenarios in the Benchmark, as by design it has been made very probable to obtain positive rewards by a random series of actions.

4.1 Deep Q-Networks

The environments in our benchmark are partially observable, since the model's input only contains a portion of the visual information of the entire state x_t, due to the limited view angle. We therefore consider a sequences of actions and observations ($s_t = x_1, a_1, x_2, \ldots, a_{t-1}, x_t$) to build a policy [53]. This is formalized as a finite Markov Decision Process (MDP), in which every sequence is a distinct state. An MPD is a tuple $(S, A, P, \mathcal{R}, \gamma)$, where S is the set of states, A is the set of actions the agent can take at each time step t, γ is a discount factor [7], P is the transition probability of going from state s to s' using action a, $\mathcal{R}$ is the reward function, defining the feedback signal that the agent receives after taking an action and changing the state. The objective is therefore to learn a policy $\pi : s \rightarrow a$ that maximizes the expected discounted cumulative reward over the agent's run [7], by using the complete sequence s_t as the state representation at time t.

Such a policy can be attained using Q-Learning [77], in which the action-value function $Q^{\pi}(s, a)$ is learned iteratively, so as to gradually approximate the expected reward in a model-free fashion. Before the learning begins, the Q-table is initialized to arbitrary fixed values. Then, at each time step t, the agent selects an action a_t, observes a reward r_t, and enters a new state s_{t+1},

after which we update Q. The core of the algorithm is a Bellman equation as a simple value iteration update, using the weighted average of the old value and the new information:

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_t\gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right) \tag{4.1}$$

where r_t is the reward received when moving from the state s_t to the state s_{t+1}, and α is the learning rate ($0 < \alpha \leq 1$).

High-dimensional environments in FPS games, however, make it infeasible to use simple Q-learning due to the curse of dimensionality [7]. Deep-Q Networks (DQN) extend model-free RL algorithms, like Q-Learning, to use Deep Neural Networks as function approximators, implicitly capturing hierarchies in the state representation that make the RL problem scale even to visual input states. The objective now becomes to find a policy that maximizes the expected sum of discounted rewards:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}, \tag{4.2}$$

where $\gamma \in [0, 1]$ is a discount factor that determines the importance of future rewards, and T is the time at which the game terminates. The Q-function of a given policy π is now defined as the expected return from executing an action a in a state s:

$$Q^{\pi}(s, a) = \mathbb{E}\big[R_t | s_t = s, a_t = a\big]. \tag{4.3}$$

It is more feasible to use a function approximator to estimate the action-value function Q. In particular, we use a neural network parameterized by θ to obtain an estimate of the Q-function of the current policy, which is close to the optimal Q-function Q^*, defined as the highest return that can be expected to achieve by following any strategy:

$$Q^*(s, a) = \max_{\pi} \mathbb{E}\big[R_t | s_t = s, a_t = a, \pi\big] = \max_{\pi} Q^{\pi}(s, a). \tag{4.4}$$

The optimal action-value function obeys the Bellman optimality equation, in which given the optimal value $Q^*(s', a')$ of the sequence s' at the following time-step for all possible actions a', the optimal strategy is to select the action a', maximizing the expected value of $r + \gamma Q^*(s', a')$ [53] in the following fashion:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}}\Big[r + \gamma \max_{a'} Q^*(s', a') \big| s, a\Big]. \tag{4.5}$$

The objective thus becomes to estimate the action-value function by using a non-linear approximator, and finding θ such that $Q_\theta(s, a) \approx Q^*(s, a)$ [44]. This network can be trained by minimising a sequence of loss functions $L_t(\theta_t)$ that changes at each time step t,

$$L_t(\theta_t) = \mathbb{E}_{s, a \sim \rho(s,a)}\Big[\big(y_t - Q_{\theta_t}(s, a)\big)^2\Big], \tag{4.6}$$

where $y_t = \mathbb{E}_{s' \sim \mathcal{E}}\Big[r + \gamma \max_{a'} Q^*(s', a') \Big| s, a\Big]$ is the target for time step t, and $\rho(s, a)$ is the behavior distribution [53]. Differentiating the given loss function with respect to θ leads to the following gradient:

$$\nabla_{\theta_t} L_t(\theta_t) = \mathbb{E}_{s,a \sim \rho(s,a); s' \sim \mathcal{E}}\Bigg[\Big(r + \gamma \max_{a'} Q(s', a'; \theta_{t-1}) - Q(s, a; \theta_t)\Big) \nabla_{\theta_t} Q_{\theta_t}(s, a)\Bigg]. \tag{4.7}$$

Rather than computing the accurate expectations in the above gradient, we use a rough approximation for the updates:

$$\nabla_{\theta_t} L_t(\theta_t) \approx \big(y_t - Q_\theta(s, a)\big) \nabla_{\theta_t} Q_{\theta_t}(s, a). \tag{4.8}$$

4.1.1 Experience Replay

Experience replay [47] enables RL agents to memorize past experiences and reuse them for the upcoming situations [81]. Using memory replay has many advantages over the standard Q-learning. First of all, every step of an experience has the potential to be used in several weight updates, which grants superior data efficiency. Further, it is inefficient to learn directly from consecutive samples, due to the strong correlations between them. Randomizing the samples breaks these correlations and thus decreases the variance of the updates. Moreover, experience replay averages the behaviour distribution over many of its previous states, thus avoiding oscillations or divergence in the parameters and smoothing out learning. This is important because unwanted feedback loops may arise and the parameters could get stuck in a poor local minimum due to the tendency of training batches being dominated by samples of maximizing actions [54].

To perform experience replay, the experiences of an agent $e_t = (s_t, a_t, r_t, s_{t+1})$ are stored at every time-step t in a buffer $D_t = \{e_1, \ldots, e_t\}$, which are then pooled across numerous games. During Q-learning, updates are applied on samples of transitions $(s, a, r, s') \sim U(D)$, drawn uniformly at random from the pool of stored samples [53]. To ensure superior data usage, we include a fixed number of most recently acquired transitions to the end of the batch of experience, replacing the randomly sampled ones.

4.1.2 Target Network

In vanilla DQN, the same network is used for making predictions during the feed-forward pass, and performing gradient updates during backpropagation. This approach has the tendency to increase $Q(s_{t+1}, a)$ for every a, given an update that increases $Q(s_t, a_t)$. It may further lead to a rapid increase or decrease of the target y_t, causing severe undesirable oscillations in the policy. In order to improve stability, we clone our regular network Q after every predetermined n updates to obtain a target network $\hat{Q}$, which is then used for generating the learning targets in the following n updates for Q. Using an older set of parameters, this way of generating targets adds a delay between the time an

update is made to Q and the time the targets y_t are affected by the update. This makes oscillations in the policy less likely, because in standard Q-learning, . Generating the targets using an older set of parameters adds a delay between the time an update is made and the time the update affects the targets, making divergence or oscillations much more unlikely [54].

4.1.3 Frame Skipping

In order to accelerate training, we use the frame skipping [4] technique, in which the agent receives input of the screen only every $k+1$ frames, where k represents the parameter for the number of frames skipped for each step. A single action, which the agent has determined as most rewarding, is then repeated over all the following k frames. Too high of a frame skipping rate may, however, inhibit performance, since the agent becomes unable to receive sufficient input about the state, and loses its ability to respond accordingly. The objectives in our designed scenarios in the GViZDoom Benchmark (e.g., accurately firing at enemies, or maneuvering away from multiple incoming projectiles) require swift reaction from the agent, as the state may change to a great extent in a matter of a few frames. If the agent were to repeat a single action for too many times, it may either over-rotate and miss the enemy, or over-navigate and get hit by the incoming projectile. We use a frame skip rate of $k = 4$, since it has been found most suitable in former research on the ViZDoom platform [44, 41].

4.1.4 Algorithm

As basis for training a poor baseline in our experiments, we use DQN with experience replay and a target network. We modify the initial algorithm proposed by Mnih et al. [53] by including a target network for generating the targets (Algorithm 1), as described in Section 4.1.2. The use of experience replay and target networks enables relatively stable learning of Q-values, which has previously led to superhuman performance on several Atari games [27].

The learning task is solved directly using samples from the environment, without explicit estimations of transitions or rewards. The algorithm is off-policy [69], meaning that it ensures reasonable exploration of the state space by basing its actions on a behaviour distribution. More formally, the agent performs a random action with probability ϵ, while also learning the greedy policy $a = argmax_{a'}Q(s, a'; \theta)$, by maximizing its returned rewards with probability $1 - \epsilon$. The value of ϵ decays over time, meaning that the rate of exploration also decreases.

Algorithm 1: Deep Q-learning with Experience Replay and a a Fixed Target Q-Network

Initialize replay memory $\boldsymbol{D}$ to capacity N
Initialize action-value function $\boldsymbol{Q}$ with random weights $\boldsymbol{\theta}$
Initialize target action-value function $\hat{\boldsymbol{Q}}$ with weights $\boldsymbol{\theta}^{-} = \boldsymbol{\theta}$
for *episode = 1, M* **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$;
  for t *= 1, T* **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \text{argmax}_a \, Q(\phi(s_t), a; \boldsymbol{\theta})$
    Perform action a_t, and observe the reward r_t and next image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store the transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\boldsymbol{D}$
    Sample random minibatch of transitions $(\phi_t, a_t, r_t, \phi_{j+1})$ from $\boldsymbol{D}$
    Set $y_t = \begin{cases} r_t & \text{for terminal } \phi_{j+1} \\ r_t + \gamma max_{a'} Q(\phi_{j+1}, a'; \boldsymbol{\theta}), & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_t - Q(\phi_t, a_t; \boldsymbol{\theta}))^2$ according to equation 4.7
    Reset $\hat{\boldsymbol{Q}} = \boldsymbol{Q}$ after every a fixed number of steps.
  end
end

4.1.5 Model Architecture

We use the CNN architecture proposed by Hausknecht and Stone [25], consisting of three convolutional blocks, as depicted in Figure 4.1. This schematic is only applicable for a single frame, since the actual input consists of multiple stacked frames. All of the three RGB feature maps are used, as grayscale images have been found to decrease performance [44]. Using raw Doom frames (640×480 pixel images with a 256 color palette) directly, is computationally too demanding. In order to reduce dimensionality, we first preprocessed the raw frames by down-sampling them to a 84×84 pixel image. The function ϕ from Algorithm 1 applies this preprocessing to the last 4 frames of a previous experience and stacks them afterwards [53]. This is used as input to the neural network and is convoluted three times with a decreasing kernel size, after which a rectifier nonlinearity is applied. We use strided convolutional layers instead of Max-Pooling layers, since they enhance the CNN accuracy and reduce the model size [67]. The resulting activations are flattened to a single dimension, by processing them through a linear 4096-unit fully-connected layer. The final layer of the model provides us with Q-Values after a softmax operation, having a single output for each of the possible actions $a \in \mathcal{A}_\rangle$ for a given environment E_i. This kind of architecture, having only the state representation as the input to the neural network, and a separate output unit for each possible action, has the ability of computing Q-values for all of the possible actions in a single state with only passing through

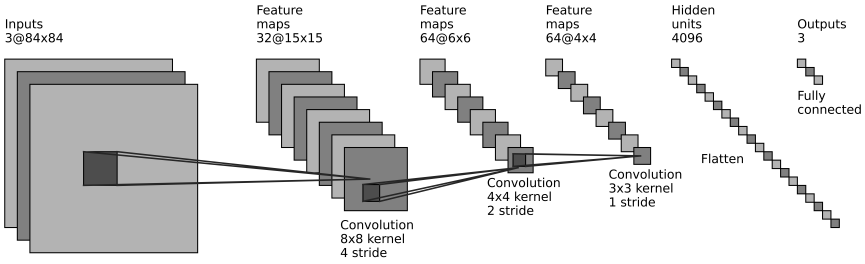


Figure 4.1: DQN model architecture with for an action space of $|\mathcal{A}| = 3$

the network once [53]. We initialize the weights of our model uniformly, using the initializer proposed by Kaiming He [26].

4.1.6 Reward Shaping

In order to facilitate the learning process of the agent, and tackle the problem of delayed rewards and a sparse replay table, we use reward shaping [56], in which we adjust the rewards to provide more appropriate feedback relative to the objective. This is crucial for the learning a favourable policy, as our environments are of very diverse nature. The reward shaping functions of all scenarios are outlined in Section 3.4.

4.2 Rainbow

Rainbow [27] combines several state-of-the-art DRL improvements to the DQN algorithm into a single learning agent. In the following, we describe the six techniques, which comprise Rainbow, and are complementary to one another to a certain extent.

4.2.1 Double Q-Learning

The conventional Q-learning algorithm performs poorly in some stochastic environments due to a large overestimation bias for action values during the maximization step in Equation 4.1 [71]. The reason behind this is that the selected samples are used for both deciding which action is the best in terms of yielding highest expected reward and for estimating that action-value. Thus, to overcome the overestimation problem, Double Q-learning [24] uses two Q-value functions to effectively decouple the selection of the action from its evaluation in the maximization performed for the bootstrap target. In particular, similarly to standard Q-Learning, the value of the greedy policy is still evaluated according to the current values in the online network $\boldsymbol{\theta}$, and the action is selected accordingly. However, we now use a second set of weights $\hat{\boldsymbol{\theta}}$ to evaluate the value of this

policy. This can effectively be combined with DQN [73], using the following loss function:

$$\mathscr{L} = \left(r_{t+1} + \gamma_{t+1} Q_{\hat{\boldsymbol{\theta}}} \left(s_{t+1}, \underset{a'}{\operatorname{argmax}}\, Q_{\boldsymbol{\theta}}(s_{t+1}, a') \right) - Q_{\boldsymbol{\theta}}(s_t, a_t) \right)^2. \tag{4.9}$$

Updating the weights of the target network remains unchanged from the vanilla DQN, making a periodic copy of the online network, as explained in section 4.1.2.

4.2.2 Dueling Networks

The state-action values $Q(s, a)$ can be composed into two fundamental values: a state-value function $\mathscr{V}(s)$, estimating the importance of being in a particular state s, and an action-value function $\mathscr{A}(a)$, estimating the advantage of selecting action a instead of other actions. Since it is often unnecessary to estimate both of these values in many MDPs, we could use a separate stream of fully connected layers in the DQN to obtain an estimation for both $\mathscr{V}(s)$ and $\mathscr{A}(a)$. This type of neural architecture designed for value based RL is called a dueling network [76] (Figure 4.2). The streams share a convolutional encoder, and their outputs are merged by a predefined aggregator. We can thus combine the two streams to generate a single output as follows:

$$Q_\theta(s, a) = v_\eta(f_\xi(s)) + a_\psi(f_\xi(s), a) - \frac{\sum_{a'} a_\psi(f_\xi(s), a')}{|\mathcal{A}|}, \tag{4.10}$$

where $\mathcal{A}$ is the action space; ξ, η, and ψ respectively represent the parameters of the shared convolutional encoder f_ξ, the value stream v_η, and the advantage stream a_ψ; and $\theta = \{\xi, \eta, \psi\}$ is their concatenation. The loss function can be combined in a similar fashion to Equation 4.6. This network architecture helps to generalize across actions and significantly improves performance.

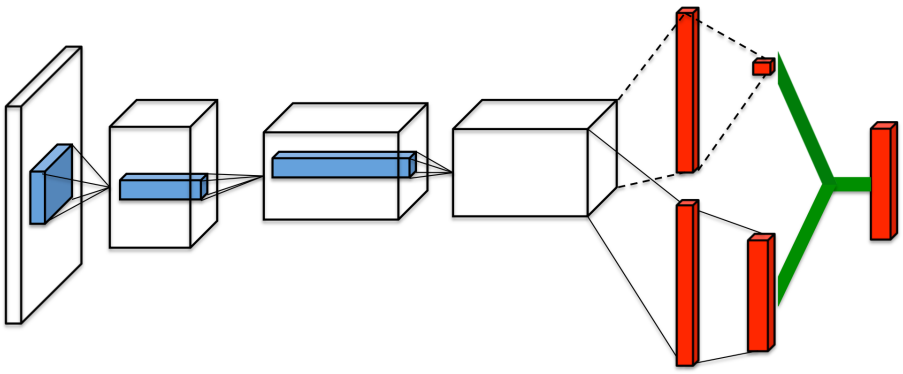


Figure 4.2: Dueling network architecture from the work by Wang et al. [76].

4.2.3 Prioritized Experience Replay

The standard experience replay mechanism enables the agent to remember and reuse past transitions. In particular, experiences are sampled from the replay buffer uniformly. This approach, however, completely disregards the significance of the experiences, as it simply replays transitions at the same frequency as the agent originally encountered them. Experiences, which are rich in terms of providing the agent a lot to learn from them, occur rarely, which makes the chance of them being selected very low. Sampling such transitions more frequently would improve data efficiency, and thus lead to a more desirable policy in lesser training time.

Adding prioritization to the tuples of experience in the replay buffer [64] precisely enables us to focus the more on relevant transitions. This is achieved by taking priority in experiences in which during training there was a great difference between the agent's prediction and the temporal difference (TD) target. In other words, we sample transitions with a probability related to the previously encountered absolute value of the magnitude of the TD error. We thus find the priority of an experience as follows:

$$p_t = |\delta_t| + e, \tag{4.11}$$

where δ_t is the magnitude at time step t, and e is a constant assuring that no experience has 0 probability of being selected. We find the probability of selecting a transition i by normalizing by all priority values in the replay buffer:

$$P(i) = \frac{p_i^a}{\sum_k p_k^a}, \tag{4.12}$$

where $a \in [0, 1]$ is a hyperparameter used to introduce stochasticity in the experience selection. In order to correct for the introduced bias towards high-priority samples, and avert the risk of overfitting, we use importance sampling (IS) weights, which shall adjust the update by reducing the weights of the prevalent samples:

$$w_i = \left(\frac{1}{N \cdot P(i)}\right)^{\beta}, \tag{4.13}$$

where N is the replay buffer size, and β controls how much the IS weights affect learning. The value of this parameter is linearly annealed throughout the duration of training from its initial value to 1. This use useful, because the IS weights are more relevant when the Q-Values begin to converge.

Instead of sorting an array or using a dequeue as with standard experience replay, we initialize the replay buffer as an unsorted *SumTree*, a version of a binary tree, where the value of a parent node is equal to the sum of the values of its children. This grants us $O(log(n))$ efficiency in updating and sampling from the tree. New transitions are inserted into the buffer with maximum priority.

4.2.4 Noisy Nets

A Noisy Net [19] is a type of stochastic neural network in which the bias and weights are repeatedly flustered during training by a parametric noise function. This is generally achieved by adding Gaussian noise to the last fully connected layers of the network, but is applicable to any number of layers. The amount of noise is adjustable during training, which grants control over when and to what degree it is desirable to introduce uncertainty to the model weights. Over time, the network can thus learn to ignore the noisy stream, but will do so at various rates in different parts of the state space, allowing a form of self-annealing exploration conditional to the state.

The limitations of using ϵ-greedy policies for exploration are palpable in environments, where it takes a myriad number of actions before encountering a positive reward. The noisy network is implemented by first replacing the ϵ-greedy policy by a randomized action-value function. Afterwards, the selected layers of the value network are parameterized with noise, in which the parameters are drawn from the parameter distribution of the noisy network after each iteration of training. For experience replay, the given noisy network parameter sample is held fixed across the given batch. Re-sampling is performed before every action, in case an optimization step is taken for every step of action. In more mathematical terms, we utilize a noisy linear layer that combines a deterministic and noisy stream, such that

$$\boldsymbol{y} = (\boldsymbol{b} + \boldsymbol{W}\boldsymbol{x}) + (\boldsymbol{b}_{noisy} \odot \epsilon^b + (\boldsymbol{W}_{noisy} \odot \epsilon^w)\boldsymbol{x}), \tag{4.14}$$

where ϵ^b and ϵ^w are random variables, and $\odot$ denotes the elements-wise product [27].

4.2.5 Multi-Step Learning

Bootstrapping in standard Q-learning is performed by accumulating a single reward, and then using the greedy action at the next step. Alternatively, to propagate newly observed rewards faster to earlier visited states, forward-view multi-step targets can be used [68]. In particular, the truncated n-step return from a given state s_t can be defined as

$$r_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} r_{t+k+1}. \tag{4.15}$$

we can thus define a multi-step variant of DQN by minimizing the alternative loss:

$$\mathscr{L} = \left(r_t^{(n)} + \gamma_t^{(n)} \max_{a'} Q_{\hat{\theta}}(s_{t+n}, a') - Q_\theta(s_t, a_t) \right)^2. \tag{4.16}$$

Learning from a suitably tuned number of multi-step bootstrap targets helps create a shift in the bias-variance trade-off, and often leads to faster learning [70].

4.2.6 Distributional RL

The standard DQN described in section 4.1 uses the Bellman equation to approximate the expected value of future rewards. However, given that the environment is stochastic in nature like the scenarios in the GViZDoom Benchmark, and the future rewards follow a multimodal distribution, selecting actions solely based on expected values may not lead to the optimal outcome. This can be improved using distributional Q-learning [3], in which we use a categorical distribution to approximate the distribution of discounted returns instead of the expected return. In particular, given that $\mathcal{Z}(s_t, a_t)$ is the return obtained from executing action a_t in state s_t following the current policy, then $Q(s_t, a_t) = \mathbb{E}[\mathcal{Z}(s_t, a_t)]$, where $\mathcal{Z}$ represents the distribution of future rewards [49]. We can therefore obtain the distributional version of Bellman equation as follows:

$$\mathcal{Z}(s, a) = r + \gamma \mathcal{Z}(s, a). \tag{4.17}$$

The performance relies heavily on how well the distribution function $\mathcal{Z}$ in defined. More specifically, we define a discrete support vector $\boldsymbol{z}$ with $N_{atoms} \in \mathbb{N}^+ atoms$, defined by $z^i = v_{min} + (i-1)\frac{v_{max}-v_{min}}{N_{atoms}-1}$ for $i \in \{1, \ldots, N_{atoms}\}$ [27]. We form a categorical distribution, placing probability masses on $\boldsymbol{z}$, from which at every time step t, we can find the approximating distribution d_t, with a probability mass of $p^i_\theta(s_t, a_t)$ on every atom i, such that $d_t = (\boldsymbol{z}, \boldsymbol{z}_\theta(s_t, a_t))$. The goal is thus to optimize θ, such that the distribution d_t well approximates the ground truth distribution of returns. Under an optimal policy π^*, for a state s_t and action a_t, the distribution of the returns should therefore exactly match a target distribution. We find this distribution by first taking the distribution for the next state s_{t+1} and the optimal action according to our policy $a^*_{t+1} = \pi^*(s_{t+1})$. We further use the discount to contract it towards zero, and then shift it by the reward. We construct a new support for the target distribution, and minimize the Kullbeck-Leibler (KL) divergence between the target and approximating (d_t) distributions. We represent the θ as a neural network, as in DQN, but with $N_{atoms} \times N_{actions}$ outputs. For each action dimension of the output, we independently apply a softmax activation to ensure that the distribution for each action is appropriately normalized.

4.3 Multi-Task Learning

Since we are using multiple training tasks $\mathcal{T}^{tr}$ to train a single model, we formulate this task into a multi-task learning (MTL) problem. MTL improves generalization by leveraging the domain-specific information contained in the training signals of related tasks [8]. The goal of multi-task RL is to learn a single, task conditioned policy $\pi(a|s, z)$, where z indicates an encoding of the task ID. This policy should maximize the average expected return across all tasks from the task distribution $p(\mathcal{T})$ [80], given by

$$\mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})}\left[\mathbb{E}_\pi\Big[\sum_{t=0}^{T} \gamma^t r_t(s_t, a_t)\Big]\right]. \tag{4.18}$$

We train the model asynchronously, by using a shared experience replay buffer $\mathscr{D}$ across each task $\mathcal{T}_j \triangleq \{\mathscr{S}_j, \mathscr{A}_j . p_j(s_1), p_j(s'|s,a), r_j(s,a)\}$ in the set of training tasks $\mathcal{T}_i^{tr}$ in environment E_i. We use hard parameters sharing with no task-specific output layers in the model. The agent executes the feed-forward of all tasks in parallel by making use of the shared target network $\hat{\theta}$. In our simulation environment, each task is run on a separate instance of the Doom game. Given that N is the memory capacity, we collect experience by storing transitions of each task $\mathscr{D}_{\mathcal{T}_i} = \{(s_{1:N}, a_{1:N}, r_{1:N}, s'_{1:N})\}$ into the shared replay buffer $\mathscr{D} \leftarrow \mathscr{D} \cup \mathscr{D}_{\mathcal{T}_i}$.

4.3.1 Example-Level Prioritization

Backpropagation is performed in a separate thread, which trains the online network θ. We sample a batch of transitions from the shared buffer $\mathscr{D}^b \sim \mathscr{D}$. This is performed uniformly for our DQN baseline, which grants experiences from every task an equal probability of being selected. However, it has been previously observed that imbalances in task difficulty can lead to unnecessary emphasis on easier tasks, thus neglecting and slowing progress on difficult tasks [21]. On the contrary, since Rainbow uses prioritized replay, more difficult tasks with experiences of higher priority will be sampled more often, and thus dominate the weight updates.

4.3.2 Task-Level Prioritization

In traditional multitask learning [8], a model continues to invest the same level of emphasis on easy tasks, even after mastering them. Perfecting such simple tasks is a waste of valuable resources. As a result, challenging tasks, which may require additional learning, learn slowly and perform poorly, compared to easier tasks [31]. We thus use key performance indicators (**KPI**s) as progress signals to dynamically prioritize more difficult tasks by assigning more weight to samples from such tasks when computing the loss [21]. We use the success metrics outlined in Table 3.1 as KPIs for a given scenario. We find the KPI value of a task $\mathcal{T}_j$ as the running mean of the success metric across the last k iterations:

$$KPI_{\mathcal{T}_j} = \frac{1}{k} \sum_{l=0}^{k} score_i(\pi_{t-l}), \tag{4.19}$$

where t is the current time step, i is the index of the environment, and π is our policy at a certain time step. We use this value to linearly scale the weights of samples from a task $\mathcal{T}_j$ as follows:

$$w_j = \frac{\max_{\mathcal{T}_i} KPI_{\mathcal{T}_i}}{KPI_{\mathcal{T}_j}}. \tag{4.20}$$

Hence, the samples from a task with the highest performance KPI_{max} at a given time step t are assigned a weight of $w = 1$, and all other samples of more difficult tasks receive a higher weight.

Chapter 5

Experimental Results and Discussion

In this chapter we outline our experimental setup, and conduct four experiments using the GViZDoom benchmark. First, we assess the level difficulty in our benchmark. Second, we evaluate the impact of the training set size on generalizability. Third, we compare two value-base RL algorithms. Fourth, we evaluate our trained Rainbow agent on tasks of all difficulty levels. Finally, we present and discuss the results of our experiments.

5.1 Experimental Setup

In this section, we describe the general experimental protocols, the software built for the benchmark, the hardware for training the models, the methods and algorithms with their corresponding hyperparameters for training and evaluating the agents.

5.1.1 Protocol

For our experiments, we use every scenario in the benchmark. The tasks of these scenarios with their attributes are described in Tables A.1, A.2, A.3, and A.4. Screenshots of some lower level tasks are provided in Table 3.2. To evaluate the generalization competence of an agent, we default to using lower level tasks for training and higher level ones for inference. To provide means of reproducibility, and to increase the reliability of the experimental results, we control the pseudorandom nature of the environments in a similar fashion as in the work by Cobbe et al.[14], by using a respective predefined seed $s \in S$ for training each model. Every task in our experiments is thus trained using three seeds $S = [1111, 2222, 3333]$. The evaluation score for a task is determined by the corresponding success metric (see Table 3.1) of the scenario from which the task originates.

5.1.2 Algorithms

Since our environments have a very limited discrete action space, we employ two value-based DRL methods in our experiments. As an initial baseline, we use **DQN**, a popular algorithm for RL in high-dimensional environments, such as video games. The hyperparameters used for training the vanilla DQN are brought in Table 5.1. As an improved variant, we use **Rainbow** [27], a combination of state-of-the-art improvements to DQN. We apply the same parameters as used for DQN in Table 5.1. Additional hyperparameters, specific to Rainbow, are presented in Table 5.2. We retain the architecture and most of the hyperparameters from [27], with a few minor changes. First, we use a priority exponent $\omega = 0.6$ for Prioritized Experience Replay (PER). Second, we use a replay buffer size of 100K instead of 1M to lower the algorithm's memory consumption. Third, to initialize the weights in the noisy stream, we set $\sigma_0 = 0.017$, as proposed in the original paper for noisy networks[19]. The value of $\sigma_0 = 0.5$ used in [27] yielded poor performance in our experiments. Lastly, we apply the distributional min and max values according to the reward spectrum of the scenario used for training. We use the Rainbow algorithm for running an experiment by default, if not specified otherwise. The agent is trained using the Huber loss function, the Adam [42] optimizer, and the convolutional architecture outlined in section 4.1.5.

Table 5.1: DQN hyperparameters

Hyperparameter	Value
Replay memory capacity	50K
Target network period	3000
Frames per action	4
Gathering experience	10K
Annealing epsilon	50K
Initial epsilon	1.0
Final epsilon	0.001
Discount factor γ	0.99
Adam learning rate α	10^{-4}
Adam ϵ	10^{-7}
Mini-batch size	32

Table 5.2: Rainbow hyperparameters

Hyperparameter	Value
Multi-step returns n	3
Noisy Nets σ_0	0.017
PER exponent ω	0.6
PER importance β	$0.4 \rightarrow 1.0$
PER minimum priority	0.01
Distributional atoms	51

5.1.3 Hardware

The GPU used for running our experiments is an ASUS Turbo GeForce GTX 1080 Ti, with 11GB RAM, 3584 CUDA cores, and a compute capability of 6.1. We use an Intel Xeon Broadwell-EP 2683v4 CPU with 1024GB RAM, 64 hyperthreads, and a processing speed of 2.1GHz.

5.1.4 Software

GViZDoom[1] is written in the Python programming language. The program is executable via the command line with all the configurable parameters. We use *Keras* [13] with a *TensorFlow* [1] backend for constructing and training the models. The software architecture of value-based DRL methods is illustrated in Figure 5.1. To implement and use custom algorithms in the benchmark, one needs to extend the *Agent* class. Likewise, extending the *Scenario* class enables the creation of a new scenario. To facilitate separating the usage of the CPU and GPU for better optimization, simulation in the environment, and the computation heavy backpropagation are configurable to be executed asynchronously in separate threads. Moreover, it is possible to run multiple threads for independently training the online network. Although ViZDoom is rather lightweight to run [39], experience from the environment can be gathered more efficiently in this manner when using value-based methods. As we formulated in Section 4.3, every task of a scenario is run in a separate thread which holds a Doom game instance, and transitions are collected into a shared replay buffer.

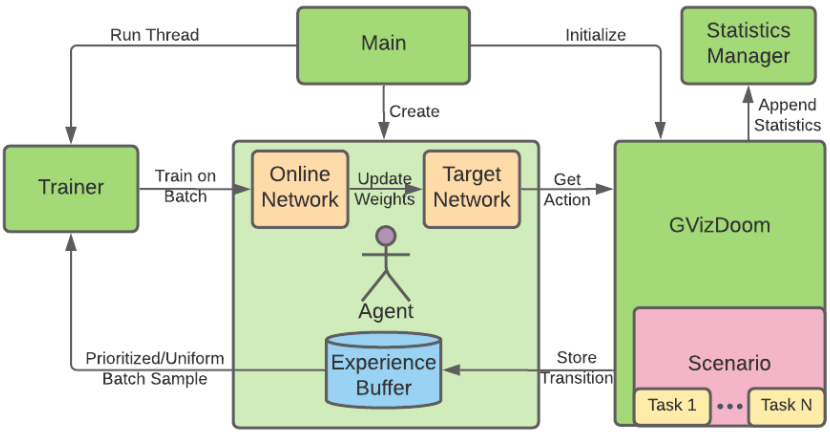


Figure 5.1: Software architecture of value-based DRL methods.

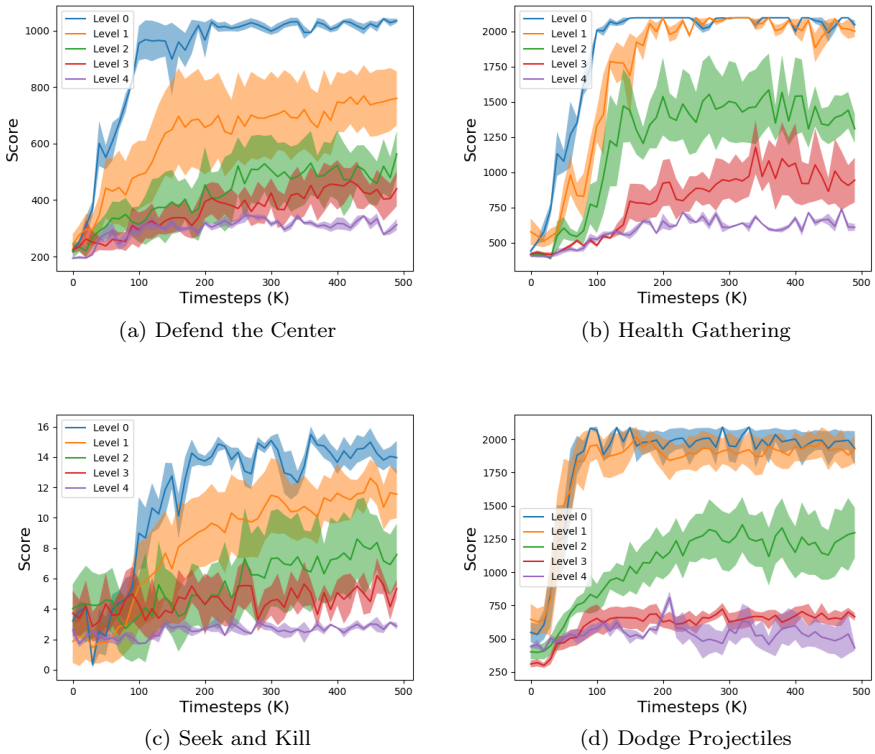
5.2 Generalization Experiments

In this section we use the GViZDoom benchmark to run four experiments. We then present and discuss the results.

[1]The code is made available at https://github.com/TTomilin/GViZDoom

5.2.1 Training Difficulty

We first demonstrate how our difficulty level design (outlined in Section 3.2) impacts the training performance of the Rainbow agent. The model is trained for 500K steps on every individual task of each selected scenario on three seeds. The performance statistics are aggregated and stored after every 5000 iterations. We find the mean score of a task across all seeds, and then average the scores of tasks belonging to the same level. We hypothesize that the agent performs poorly on tasks of higher levels with more difficulty attributes. We additionally wish to observe whether training converges for all levels of every scenario, and how long it takes. Knowing the approximate number of iterations required for convergence, we can propose a lower bound for further experiments with the same training protocol.



(a) Defend the Center

(b) Health Gathering

(c) Seek and Kill

(d) Dodge Projectiles

Figure 5.2: The training performance of the Rainbow Agent on individual tasks of all difficulty levels.

The training curves are plotted on Figures 5.2a, 5.2b, 5.2c, and 5.2d. It can be observed that the training of most models converges before 200K iterations.

Scenario	Level 0	Level 1	Level 2	Level 3	Level 4	Average
Defend the Center	87.48	60.11	40.35	34.46	29.69	50.42
Health Gathering	86.68	78.60	55.64	36.13	27.43	56.90
Seek and Kill	59.69	42.02	28.87	24.11	13.92	33.72
Dodge Projectiles	86.10	84.18	46.79	28.66	28.21	54.79
Average (P_{norm})	79.99	66.23	42.91	30.84	24.81	48.96

Table 5.3: Normalized performance indicators of training the Rainbow agent on all tasks of each level across the full training duration. The values are reported across 3 seeds.

We will hence use this number of training steps for the following experiments. We can further acknowledge that the agent has roughly solved the *default* task on all scenarios with the *Frames Alive* performance metric, since the plotted line is contiguous to the number of frames, when the episode is automatically terminated. It can also be noticed, that barely any progress has been made on the final level 4 task in the course of the entire training process. For more rigorous comparison, we use the formulas in Equations 3.1 and 3.2 to calculate the performance indicator for each level, which we present in Table 5.6. We can observe, that every level of higher difficulty has a progressively lower indicator, which matches our hypothesis. The most noticeable performance decrease (23.3%) appears between levels 1 and 2. Hence, adding a second difficulty attribute has the seemingly highest relative impediment to the Rainbow agent. The lowest performance difference is among levels 3 and 4 (6.0%), signaling that our agent struggles to prevail in a task with three or more complexity attributes, since the level 4 tasks is comprised of all the attributes. From the results of this experiment we can thus conclude that our design of level difficulty indeed presents the agent with increasing complexity as the levels progress.

The training curve of level 1 tasks of the *Defend the Center* scenario on Figure 5.2a appears to have one of the highest variances among the levels of all scenarios. We additionally plot the individual results of tasks of this difficulty level in Figure 5.3 to examine the origin of the high deviation. We can indeed observe that although the tasks belong to the same difficulty level according to our benchmark design, our Rainbow agent does not perform equally well on all of them. This indicates that the specific setup for training or evaluation of selected tasks from a single level can substantially impact the outcome.

5.2.2 Training Set Size

We further wish to determine how the number of training tasks impacts the acquired generalizability abilities of the agent using our benchmark, and what the appropriate number of tasks for training should be. To this end, we compare the performance of our Rainbow agent on unseen test tasks of higher difficulty

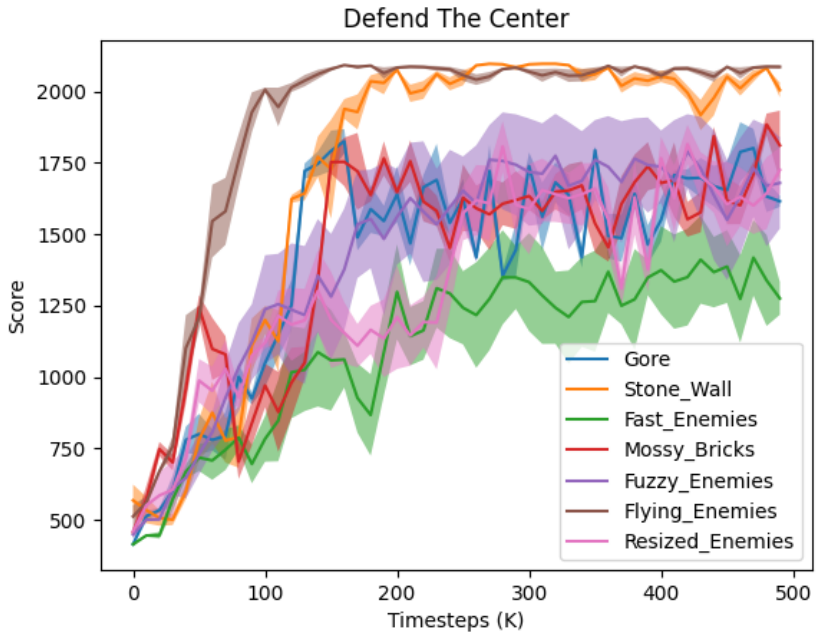


Figure 5.3: Training curves of all level 1 tasks of the Defend the Center scenario. We report the mean and standard deviation across three seeds.

levels, after having trained it on a different number of lower level training tasks. We use three training setups per scenario:

1. Single task of level **0**
2. Four tasks of levels **0** and **1**
3. Eight tasks of levels **0** and **1**

A training set with multiple tasks is trained in an MTL setting, as described in Section 4.3. We train each model for 200K iterations, saving 20 intermediate checkpoints of the model throughout the training process, one after every 10K training iterations. Every such checkpoint is thereafter evaluated for 100 episodes on each of the selected holdout tasks. We use a total of seven test tasks per scenario for evaluation: three tasks of levels $d = 2$ and $d = 3$ each, and the final *complete* task ($d = 4$), including all the scenario specific complexity attributes. We accordingly define a distinct set of difficulty levels $D_{te} = \{2, 3, 4\}$ and tasks $\mathcal{T}_i^{te} = \mathcal{T}_{i,2} \cup \mathcal{T}_{i,3} \cup \mathcal{T}_{i,4}$ for evaluation. It is important to note that we allow the difficulty attributes to overlap across training and test tasks. The performance of a single test task is determined by finding the mean score across 3 seeds, according to the environment specific metric $score_i$. Applying the use

Scenario	1 Task	4 Tasks	8 Tasks
Defend the Center	38.87	41.56	**56.06**
Health Gathering	27.92	**35.75**	32.91
Seek and Kill	27.34	**33.25**	23.92
Dodge Projectiles	35.21	**42.30**	36.42
Average (P_{norm})	32.34	**38.22**	37.33

Table 5.4: Performance indicators of different training set sizes across 3 seeds and pre-selected evaluation tasks of levels 2-4. Since multi training on a higher number of tasks requires more iterations to converge, we only use the evaluation results of the last 5 model checkpoints to ensure a more accurate comparison.

of seeds and the distinction of difficulty levels in our experimental protocol, we complement Equation 3.1 to calculate the evaluation performance in this experiment as follows:

$$P_i(\pi_\theta) = \frac{1}{|D_{te}| \cdot |\mathcal{T}_i^{te}| \cdot |S|} \sum_{d \in D_{te}} \sum_{t \in \mathcal{T}_{i,d}^{te}} \sum_{s \in S} \text{score}_i\Big(t_s(\pi_\theta)\Big). \tag{5.1}$$

Note that the calculated performance indicators using our setup are heavily contingent on the tasks selected for evaluation, and is thus only directly comparable if the experimental setup of scenarios and test tasks matches. We hypothesize that a multi-task training setting with a higher number of tasks reaps a substantially favorable outcome in terms of generalizability, compared to using a single training task.

The evaluation scores of the experiment are plotted on Figure 5.4, and the performance indicators and presented in Table 5.4. We can observe that the Rainbow agent, with a training set size of 4, comprehensively dominates most evaluation tasks of the *Seek and Kill*, *Health Gathering* and *Dodge Projectiles* scenarios, whereas the training protocol with 8 tasks has predominantly prevailed in the *Defend the Center* scenario. We speculate that the latter environment has more variability across the tasks, and thus requires a larger training set size to acquire considerable proficiency. The results of the *complete* tasks do not follow the same trend. We suppose that this is caused by all agents having very poor performance on this task, since it is the most complex undertaking. As we already established in Section 5.2.1, our implementation of the Rainbow is evidently not able to make any progression on the *complete* task even if directly trained on it. Averaging the performance indicators across scenarios, we conclude that a training set of multiple tasks is indeed superior than using a single task (+5.88% and +4.99%). Moreover, the 4 task setup is slightly better (0.89%) than using 8 training tasks.

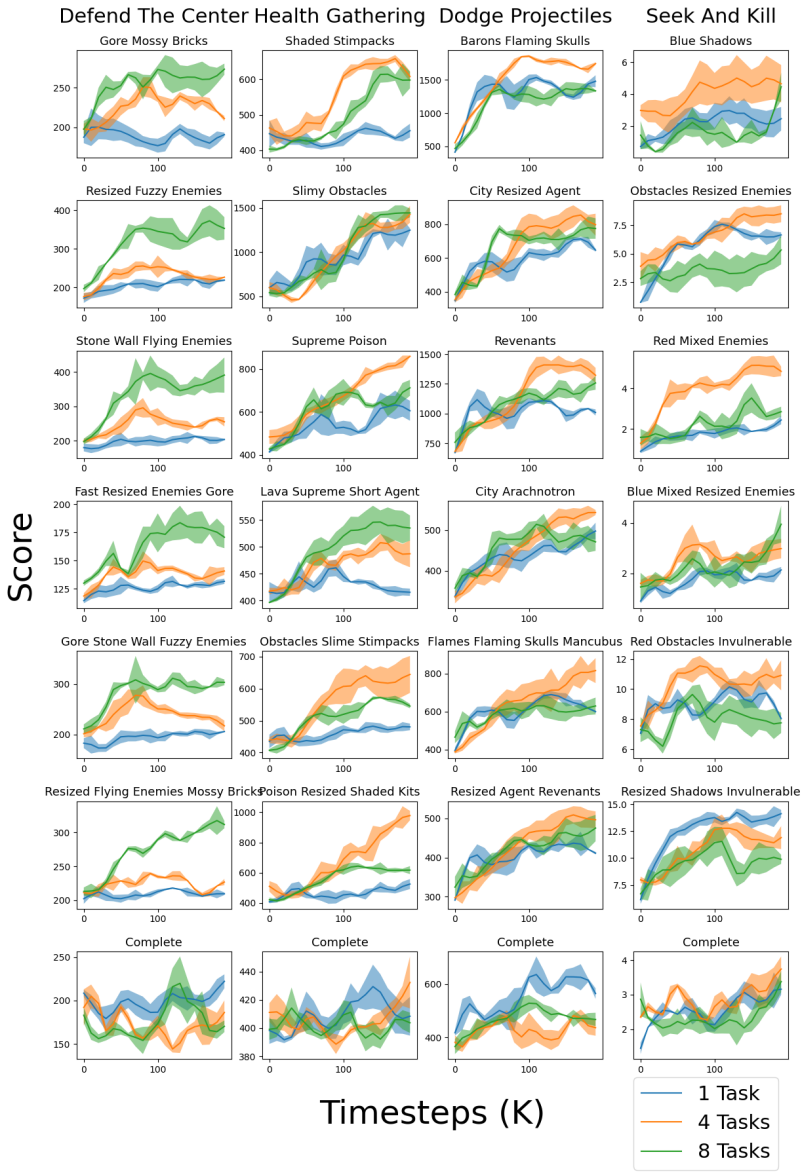


Figure 5.4: Generalization performance in each environment as a function of training set size. The mean and standard deviation is shown across 3 seeds. The tasks are ordered in an ascending manner according to the difficulty level.

Scenario	DQN	Rainbow
Defend the Center	36.90	**56.06**
Health Gathering	19.65	**35.75**
Seek and Kill	16.76	**33.25**
Dodge Projectiles	20.44	**42.30**
Average (P_{norm})	23.44	**41.84**

Table 5.5: Performance indicators of DQN and Rainbow on selected tasks of levels 2-4, after having been trained on all level 0 and 1 tasks on 3 seeds. The tasks are ordered in an ascending manner according to the difficulty level.

5.2.3 Algorithm Comparison

To demonstrate the usefulness of the GViZDoom Benchmark in providing meaningful grounds for comparison between algorithms of varying adequacy, we determine how a generic DRL algorithm, such as DQN, performs on our benchmark in comparison to our implementation of Rainbow, a more recent algorithm, combining several state-of-the-art extensions to DQN.

We train the agent on each environment E_i on four tasks of levels $D_{tr} = \{0, 1\}$, as that training set size indicated the most promising results in the previous experiment in Section 5.2.2. We thus use a training task set of $\mathcal{T}_i^{tr} = \mathcal{T}_{i,0} \cup \mathcal{T}_{i,1}$. Apart from using a single training task set, the rest of the experimental protocol is identical to the previous section. We hypothesize, that Rainbow manages to achieve a significantly higher result compared to DQN.

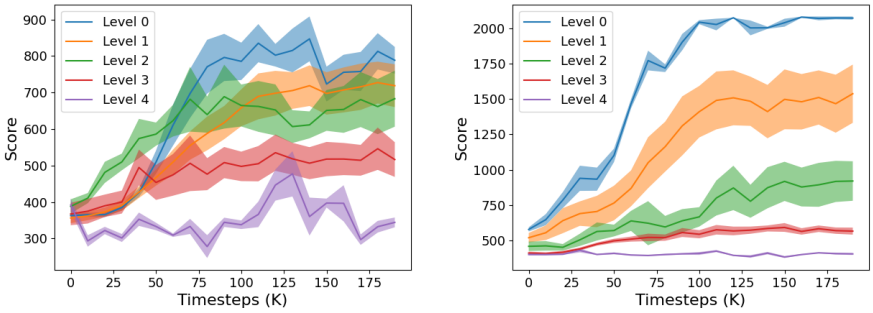
As can be observed from the results in Table 5.5, our implementation of Rainbow performed significantly better than the vanilla DQN on all scenarios. The highest improvement took place in scenario *Dodge Projectiles* (+19.16%), and the lowest in *Health Gathering* (+16.10%). The average improvement among all scenarios is $18.40 \pm 2.68\%$. The standard deviation of evaluation score differences between the methods among scenarios are rather insignificant, indicating that the environments provide a similar scale of complexity.

The evaluation plots on Figure 5.5 indicate that the DQN agent did not manage to learn anything useful from the multi-task training setting, as the evaluation scores of later model checkpoints do not noticeably increase. The agent was hence unable to sufficiently perform on the holdout task set, essentially resulting in similar competence to an agent selecting its actions at random. The difference in performance between the two methods is less noticeable on level 3 tasks, since the Rainbow agent had lower evaluation scores compared to tasks of level 2 difficulty. On the level 4 task *complete* it is no longer surmisable whether the Rainbow agent has better performance.

Figure 5.5: A comparison between DQN and Rainbow. The agent is trained on four level 0 and 1 tasks, and evaluated on tasks of higher levels. We report the mean and standard deviation across three seeds.

5.2.4 Rainbow Evaluation

As a final experiment, we evaluate how our trained Rainbow agent from the previous experiment in Section 5.2.3 performs on tasks of progressive levels of difficulty. We test every trained model checkpoint for 100 episodes on all tasks of each level.



(a) Defend the Center

(b) Health Gathering

(c) Seek and Kill

(d) Dodge Projectiles

Figure 5.6: Rainbow agent evaluated on all tasks of progressive difficulty levels, after being trained on four level 0 and 1 tasks. The mean and standard deviation for all scenarios is shown across 3 seeds.

The performance indicators are brought in table 5.6 and the evaluation curves are presented on Figure 5.6. Similarly to our confirmed hypothesis in Section 5.2.1, the performance of the agent drops as the difficulty increases. The *Health Gathering* scenario has the most distinct separation of performance between difficulty levels. The *Defend the Center* and *Seek and Kill* scenarios have higher variance, converge slower, and have more loosely separated difficulty levels according to the results of our Rainbow agent, than the other two scenarios. Likewise to the training difficulty experiment, it can once again be observed that

Scenario	Level 0	Level 1	Level 2	Level 3	Level 4	Average
Defend the Center	76.38	71.25	66.70	52.27	35.48	60.42
Health Gathering	98.55	71.53	43.13	27.32	18.99	51.91
Seek and Kill	74.66	60.70	45.62	31.80	18.59	46.27
Dodge Projectiles	79.37	63.07	52.85	24.68	22.40	48.47
Average (P_{norm})	82.24	66.64	52.08	34.02	23.87	51.77

Table 5.6: Normalized performance indicators of training the Rainbow agent on all tasks of each level across the full training duration. The values are reported across 3 seeds. Similarly to the results in table 5.4, we only use the evaluation results of the last 5 model checkpoints to ensure a more accurate comparison.

the level 3 and 4 tasks have very little noticeable improvement throughout the evaluation process.

Chapter 6

Conclusion

Training proficient agents, who are able to generalize across environments, currently remains one of the greatest challenges in reinforcement learning. We created the GViZDoom Benchmark to aid the community in grappling with this challenge. We have presented a novel approach of simulation environment design for 3D FPS video games, incorporating tasks of progressing difficulty levels, formed by combining visually observable modifications.

Even though the environments of our benchmark are tailored towards acquiring a narrow competence in the realm of FPS games, we have demonstrated in Section 5.2.1, that our approach of difficulty level design, by tweaking the environment in terms of adding or changing attributes, negatively affects the learning capability. Moreover, combining such slight visual modifications of textures, decorations, surroundings, and enemies further exacerbates performance, and can completely derail an agent from achieving the limited established objectives. Our implementation of Rainbow has nearly solved the *default* task of most environments, and performs nearly as poorly as a random agent on the *complete* task of the highest level. The first iteration of the benchmark hence provides a reasonable spectrum of complexity according to current state-of-the-art value-based DRL methods.

The same phenomena emerges, when employing a trained agent in an environment, which it has not yet encountered. This can be concluded from the relatively poor performance with a decreasing evaluation score of our trained Rainbow agent on held out tasks of higher difficulty levels in Section 5.2.4. Despite being far more elementary in nature than actual full-fledged FPS games, the modified simulation environment hinders the agent to fulfill the assigned goal. Our preliminary results suggest that performing well on a select number of training tasks does not grant the agent having learned any general concepts of the game. Instead of acquiring the relevant competencies, the policy network may simply be memorizing what action to take for a large number of observations that it has continuously been encountering. In order acquire a reliable assessment

of competence, it is therefore vital to evaluate the agent on several holdout tasks with varying unfamiliar characteristics, as our benchmark enables to do.

From the results of the experiment in Section 5.2.2 we can conclude that the number of training tasks has a substantial impact on the acquired competence of the agent. Training on a single task is evidently insufficient to achieve proper means of generalizability. We further surmise that the scenarios in our benchmark require a varying number of training tasks for appropriate performance. The optimal training set size for any GViZDoom environment nevertheless remains unclear from our experiment, and requires further research to ascertain. Moreover, we acknowledge that a different combination of an equal number of training tasks may yield a profoundly different result, i.e., there might not exists an optimal fixed training set size for a scenario.

In Section 5.2.3 we have shown that the benchmark is appropriate for comparing DRL methods of varying adequacy. In our experiment, a more competent technique, such as Rainbow, clearly outperformed a rather rudimentary value-based algorithm DQN by a significant degree on all scenarios of the benchmark. Our experiment further suggests that the scenarios provide a similar scale of difficulty for algorithm comparison.

GViZDoom grants the opportunity to design new scenarios which demand different competencies, the possibility to modify the reward functions, and to create tasks with levels of even higher difficulty. Thus, as the generalization abilities of DRL agents improves over time, and current tasks are rendered too simplistic, the benchmark is simply modifiable to further complicate the scenarios, by introducing new difficulty attributes, or enhancing existing ones by adjusting the relevant parameters. We hope that these extensions not only make the benchmark a useful high-end benchmark of agent competencies, but also position GViZDoom as a tool of academic study and a general customizable environment for presenting novel tasks to RL agents. We expect it to facilitate the design of more efficient and capable algorithms by leveraging these environments in more complex settings.

6.1 Limitations

In this section we address the limitations of our work.

6.1.1 Action Space

The environments in the initial version of the GViZDoom benchmark all have a distinct action space, which is tailored towards the objective and nature of the given scenario. We disregarded all possible irrelevant actions to reduce the model complexity and training time. This, however, negates the possibility to train or evaluate agents in a cross-environment setting. Having a unified action

space between scenarios opens the door to further opportunities in terms of researching generalization. With the expectation of improved DRL methods from future research, a consolidated action space could be implemented in the benchmark. Doing so would require the modification of some scenarios, e.g., physically limiting the movement of the agent in *Defend the Center* and *Dodge Projectiles*, instead of simply restricting the agent from using the relevant actions of movement.

6.1.2 High Variance

Our experimental results have a very high variance. This is most likely caused by the highly random nature of the environment, but more importantly due to a substantial difference in complexity between some of the difficulty attributes. We could observe this from the presented training results of the tasks belonging to the same level of the *Defend the Center* scenario in Figure 5.3. Hence, our proposed difficulty attributes might not be not ideal for providing an even degree of complexity among tasks of a single level. Combining multiple such challenging attributes for constructing higher level tasks may even further exacerbate this performance gap.

6.1.3 Multi-Task Learning

Our experiments only used off-policy model-free DRL methods, which sample past transitions from an experience buffer for learning. Training on more than one task in parallel complicates these matters, as the knowledge acquired from training on one task begins to interfere with that from another. Transferring this knowledge may inversely hurt the target performance, a phenomenon known as negative transfer [75]. Moreover, tasks may learn at different rates, which further hinders learning. We did not use any advanced multi-task learning techniques in our work, as this was not one of the main focus points, and defaulted to simply storing the experiences from all tasks in a single replay buffer. This approach may, however, result in poor sample-efficiency and inadequate utilization of the collected transitions. To mitigate the effect, we used both example-level and task-level prioritization, as described in Sections 4.3.1 and 4.3.2. Nevertheless, this does not guarantee an optimal usage of the replay buffer. Furthermore, the model and parameters were fully shared across the tasks, without any task-specific layers. This may also decrease the potential performance of our agents. A solution to this could be to share less parameters and weights across tasks or train a bigger network.

6.1.4 Value-Based Methods

In our generalization experiments, we only applied value-based methods due to small discrete action spaces in our environments. Nevertheless, there are some general disadvantages compared to policy-based methods. One problem with value-based methods is that they may have substantial oscillations in the

training process, since having a small change in the estimated action values has the potential to trigger a significant change in the choice of the next action. Policy-based methods tend to possess superior convergence properties, as they simply follow the gradient to find the best parameters, leading to smoother updates at each time step. They are also able to learn stochastic policies, which don't require the explicit implementation of an exploration/exploitation trade-off.

6.1.5 Algorithm Evaluation

The GViZDoom benchmark does not imply constraints on using a predetermined protocol in terms of tasks and difficulty levels. Users of the benchmark may thus select different tasks for training and evaluating DRL agents. The downside of this, is that the resulting performance indicators of generalizability from two experiments with varying arrangement are not directly comparable. Hence, to have a meaningful comparison of outcome with results of past research, users are required to either adopt the same setup, or rerun the previous experiment with the new setup. The reason being, that some tasks from the same difficulty level may have a substantial variance of complexity, meaning that simply selecting the same number of different tasks from a desired level might grant considerable advantages or disadvantages in scores.

6.2 Future Work

The approach and experimental results in this thesis leave ample opportunities for further research in various directions. In this section we outline some promising directions for future work, and briefly discuss how more research could improve the results and outcomes of this thesis.

6.2.1 Environment

Our initial version of the GViZDoom benchmark only consists of four scenarios, which cover a narrow subset of the possible competencies that FPS video games require the player to master. A great deal of more such environments could be designed for the benchmark to both cultivate and assess other capabilities of the agent, e.g., selecting the appropriate weapon for a given situation, accomplishing tactical or strategic objectives, obtaining items to access locked areas in the environment, accurately aiming with projectile-firing weapons, cooperating with other agents. We present a few ideas for future scenarios that cover some of such competencies:

- The agent is required to navigate to a room with a locked door, having previously located the skull key or keycard with the correct color for said door. The evaluation metric is the time required to accomplish the objective. The spatial layout of the scenario is randomly generated each episode, and the agent is spawned in an unoccupied randomized location.

- The agent is granted a weapon without any bullets. It needs to locate and collect the correct type of ammunition for the equipped weapon from the ground, and use it to eliminate hostile enemies. The agent is rewarded for picking up the correct bullets, staying alive, and killing enemies. The performance is measured in the number of enemies eliminated.
- The agent is rendered immobile at one end of a room, equipped with a rocket launcher. A number of enemies with increased movements speed are bound to the other end of the environment in substantial distance. The agent needs to anticipate the movement trajectory of the enemies and fire the weapon accordingly in the appropriate direction to eliminate them. Additional enemies continually spawn to a fixed maximum amount at a predetermined time interval. The performance of the agent is measured in the number of eliminated enemies in an episode with limited time.
- The agent needs to navigate to a location in the environment, which is guarded by enemies. The destination is unreachable without first eliminating the enemies. The agent receives a small reward for killing enemies and moving closer to its destination, and is highly rewarded for reaching the end location.

An option to improve the benchmark in terms of variability, is to procedurally generate the subtasks of scenarios, randomizing some or all of the proposed difficulty attributes. We did not use PCG due to the inability of guaranteeing the solvability of a task, and properly assessing the complexity of a level between runs. If these hindrances were to be overcome, PCG would enable expanding the benchmark with far less manual work, and to reduce the tendency of overfitting to a fixed environment.

6.2.2 Training Protocol

In our work, we attempted to get an idea of what an appropriate training set size should be for using the benchmark to yield the best generalization ability. We showed that multiple tasks are required, but did not reach a definite conclusion of the optimal number. To more accurately determine the most suitable number and combination of training tasks, further experiments are required to be conducted.

6.2.3 Agent Performance

Although it is not the main focus of this paper, our experimental results leave substantial room for improvement in terms of performance, since there are more powerful techniques and algorithms that can be utilized in this setting. The particular methods and parameter values chosen for training the agents in the paper might not be optimal after all. There are several directions to take to improve the performance of the agent. Some of which include the following:

- Using a different CNN architecture or augmenting the visual input

- Running a broad grid search for tuning the hyperparameter of our proposed method
- Applying state-of-the-art policy gradient based methods (PPO, TRPO, A3C, SAC), which work with a discrete action space
- Choosing a better training setup by determining the optimal number, difficulty, and combination of training tasks, such that the agent is faced with sufficient challenge in terms of variability while not being overwhelming of excess inconsistency
- Using more advanced MTL techniques for learning multiple tasks in parallel or sequentially
- Tweaking the reward function of scenarios to yield better feedback from the environment, and creating additional intermediary rewards to guide the agent in the correct direction

6.2.4 Curriculum Learning

It is yet to be determined how the benchmark conforms with curriculum learning [5], as it was not in the scope of this thesis. Previous DRL research on the ViZDoom platform [78] has demonstrated that utilizing curriculum training can lead to outstanding results. We thus regard it as a promising method to acquire excellent means of generalizability, especially since the environments in our benchmark have clearly defined difficulty levels. The agent could therefore be sequentially trained on tasks of gradually progressing levels. Moreover, meta-learning techniques (e.g., model-agnostic meta-learning (MAML) [18, 63], or probabilistic embeddings for actor-critic RL (PEARL) [61]) could be incorporated in the training process to further improve efficacy and performance.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

[3] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

[4] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[5] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[6] Benjamin Beyret, José Hernández-Orallo, Lucy Cheke, Marta Halina, Murray Shanahan, and Matthew Crosby. The animal-ai environment: Training and testing animal-like artificial cognition. *arXiv preprint arXiv:1909.07483*, 2019.

[7] Shehroze Bhatti, Alban Desmaison, Ondrej Miksik, Nantas Nardelli, N Siddharth, and Philip HS Torr. Playing doom with slam-augmented deep reinforcement learning. *arXiv preprint arXiv:1612.00380*, 2016.

[8] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[9] Devendra Singh Chaplot, Guillaume Lample, Kanthashree Mysore Sathyendra, and Ruslan Salakhutdinov. Transfer deep reinforcement learning in 3d environments: An empirical study. In *NIPS Deep Reinforcemente Leaning Workshop*, 2016.

[10] Jerry Zikun Chen. Reinforcement learning generalization with surprise minimization. *arXiv preprint arXiv:2004.12399*, 2020.

[11] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. *arXiv preprint arXiv:1810.08272*, 2018.

[12] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[13] François Chollet. Keras. `https://github.com/keras-team/keras`, 2015.

[14] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

[15] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289. PMLR, 2019.

[16] Alessandro Devo, Gabriele Costante, and Paolo Valigi. Deep reinforcement learning for instruction following visual navigation in 3d maze-like environments. *IEEE Robotics and Automation Letters*, 5(2):1175–1182, 2020.

[17] Jesse Farebrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

[18] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[19] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

[20] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *arXiv preprint arXiv:1811.12560*, 2018.

[21] Michelle Guo, Albert Haque, De-An Huang, Serena Yeung, and Li Fei-Fei. Dynamic task prioritization for multitask learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 270–287, 2018.

[22] Nicklas Hansen and Xiaolong Wang. Generalization in reinforcement learning by soft data augmentation. *arXiv preprint arXiv:2011.13389*, 2020.

[23] Luke Harries, Sebastian Lee, Jaroslaw Rzepecki, Katja Hofmann, and Sam Devlin. Mazeexplorer: A customisable 3d benchmark for assessing generalisation in reinforcement learning. In *2019 IEEE Conference on Games (CoG)*, pages 1–4. IEEE, 2019.

[24] Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23:2613–2621, 2010.

[25] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*, 2015.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[27] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[28] id Software. Doom, 1993.

[29] id Software. Quake III Arena, 1999.

[30] D. Amodei J. Achiam, A. Ray. Safety gym. `https://openai.com/blog/safety-gym/`, 2019.

[31] Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *International Conference on Machine Learning*, pages 4940–4950. PMLR, 2021.

[32] Lawrence Johnson, Georgios N Yannakakis, and Julian Togelius. Cellular automata for real-time generation of infinite cave levels. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, pages 1–4, 2010.

[33] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The malmo platform for artificial intelligence experimentation. In *IJCAI*, pages 4246–4247. Citeseer, 2016.

[34] Simon Judd. Slade. `https://github.com/sirjuddington/SLADE`, 2015.

[35] Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, Jonathan Harper, Ervin Teng, Hunter Henry, Adam Crespi, Julian Togelius, and Danny Lange. Obstacle tower: A generalization challenge in vision, control, and planning. *arXiv preprint arXiv:1902.01378*, 2019.

[36] Niels Justesen and Sebastian Risi. Automated curriculum learning by rewarding temporally rare events. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2018.

[37] Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. Illuminating generalization in deep reinforcement learning through procedural level generation. *arXiv preprint arXiv:1806.10729*, 2018.

[38] Amlan Kar, Aayush Prakash, Ming-Yu Liu, Eric Cameracci, Justin Yuan, Matt Rusiniak, David Acuna, Antonio Torralba, and Sanja Fidler. Meta-sim: Learning to generate synthetic datasets. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4551–4560, 2019.

[39] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016.

[40] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491, 2018.

[41] Adil Khan, Jiang Feng, Shaohui Liu, and Muhammad Zubair Asghar. Optimal skipping rates: training agents with fine-grained control using deep reinforcement learning. *Journal of Robotics*, 2019, 2019.

[42] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[43] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

[44] Guillaume Lample and Devendra Singh Chaplot. Playing FPS games with deep reinforcement learning. *arXiv preprint arXiv:1609.05521*, 2016.

[45] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020.

[46] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. *arXiv preprint arXiv:1910.05396*, 2019.

[47] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[48] Vincenzo Lomonaco, Karan Desai, Eugenio Culurciello, and Davide Maltoni. Continual reinforcement learning in 3d non-stationary environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 248–249, 2020.

[49] Nguyen Cong Luong, Dinh Thai Hoang, Shimin Gong, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. Applications of deep reinforcement learning in communications and networking: A survey. *IEEE Communications Surveys & Tutorials*, 21(4):3133–3174, 2019.

[50] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[51] Michael G Madden and Tom Howley. Transfer of experience between reinforcement learning environments with progressive difficulty. *Artificial Intelligence Review*, 21(3):375–398, 2004.

[52] Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J Pal, and Liam Paull. Active domain randomization. In *Conference on Robot Learning*, pages 1162–1176. PMLR, 2020.

[53] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[54] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[55] Sanmit Narvekar. Curriculum learning in reinforcement learning. In *IJCAI*, pages 5195–5196, 2017.

[56] Andrew Y Ng and Michael I Jordan. *Shaping and policy search in reinforcement learning*. PhD thesis, University of California, Berkeley Berkeley, 2003.

[57] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720*, 2018.

[58] D Perez-Liebana, J Liu, A Khalifa, RD Gaina, J Togelius, and SM Lucas. General video game ai: a multi-track framework for evaluating agents. *Games and Content Generation Algorithms*, 2018.

[59] Marco Pleines, Jenia Jitsev, Mike Preuss, and Frank Zimmer. Obstacle tower without human demonstrations: How far a deep feed-forward network goes with reinforcement learning. In *2020 IEEE Conference on Games (CoG)*, pages 447–454. IEEE, 2020.

[60] Aayush Prakash, Shaad Boochoon, Mark Brophy, David Acuna, Eric Cameracci, Gavriel State, Omer Shapira, and Stan Birchfield. Structured domain randomization: Bridging the reality gap by context-aware synthetic data. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7249–7255. IEEE, 2019.

[61] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pages 5331–5340. PMLR, 2019.

[62] Sebastian Risi and Julian Togelius. Increasing generality in machine learning through procedural content generation. *Nature Machine Intelligence*, 2(8):428–436, 2020.

[63] Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel. Promp: Proximal meta-policy search. *arXiv preprint arXiv:1810.06784*, 2018.

[64] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[65] Christopher Schulze and Marcus Schulze. Vizdoom: Drqn with prioritized experience replay, double-q learning and snapshot ensembling. In *Proceedings of SAI Intelligent Systems Conference*, pages 1–17. Springer, 2018.

[66] Shihong Song, Jiayi Weng, Hang Su, Dong Yan, Haosheng Zou, and Jun Zhu. Playing fps games with environment-aware hierarchical reinforcement learning. In *IJCAI*, pages 3475–3482, 2019.

[67] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[68] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[69] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[70] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[71] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the Fourth Connectionist Models Summer School*, pages 255–263. Hillsdale, NJ, 1993.

[72] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[73] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[74] Kaixin Wang, Bingyi Kang, Jie Shao, and Jiashi Feng. Improving generalization in reinforcement learning with mixture regularization. *arXiv preprint arXiv:2010.10814*, 2020.

[75] Zirui Wang, Zihang Dai, Barnabás Póczos, and Jaime Carbonell. Characterizing and avoiding negative transfer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11293–11302, 2019.

[76] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[77] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[78] Yuxin Wu and Yuandong Tian. Training agent for first-person shooter game with actor-critic curriculum learning. 2016.

[79] Marek Wydmuch, Michał Kempka, and Wojciech Jaśkowski. Vizdoom competitions: Playing Doom from pixels. *IEEE Transactions on Games*, 11(3):248–259, 2018.

[80] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pages 1094–1100. PMLR, 2020.

[81] Daochen Zha, Kwei-Herng Lai, Kaixiong Zhou, and Xia Hu. Experience replay optimization. *arXiv preprint arXiv:1906.08387*, 2019.

[82] Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018.

[83] Hanping Zhang and Yuhong Guo. Generalization of reinforcement learning with policy-aware adversarial data augmentation. *arXiv preprint arXiv:2106.15587*, 2021.

Appendix A

Task Attributes

In this chapter, we display which difficulty attributes comprise the scenario tasks.

A.1 Defend the Center

Table A.1: Task difficulty attributes of the Defend the Center scenario

Task	Noisy Textures	Decorations	Enemy Size	Enemy Rendering	Enemy Type	Enemy Speed
Defend the Center						
Level = 0						
Default						
Level = 1						
Gore		X				
Mossy Bricks	X					
Stone Wall	X					
Fuzzy Enemies				X		
Resized Enemies			X			
Fast Enemies						X
Flying Enemies					X	
Level = 2						
Fast Flying Enemies					X	X
Gore + Mossy Bricks	X	X				
Resized Fuzzy Enemies			X	X		
Stone Wall + Flying Enemies	X				X	
Fast Fuzzy Enemies				X		X
Resized Enemies + Gore		X	X			
Level = 3						
Resized Flying Enemies + Mossy Bricks	X		X		X	
Gore + Stone Wall + Fuzzy Enemies	X	X		X		
Fast Resized Enemies + Gore		X	X			X
Level = 4						
Complete	X	X	X	X	X	X

A.2 Health Gathering

Table A.2: Task difficulty attributes of the Health Gathering scenario

Task	Noisy Textures	Decorations	Item Size	Item Rendering	Item Type	Poison Vials	New Map	Agent Height
Health Gathering								
Level = 0								
Default								
Level = 1								
Obstacles		X						
Slime	X							
Lava	X							
Water	X							
Stimpacks					X			
Supreme							X	
Resized Kits			X					
Short Agent								X
Poison						X		
Shaded Kits				X				
Level = 2								
Slime + Obstacles	X	X						
Shaded Stimpacks				X	X			
Supreme + Poison						X	X	
Resized Kits + Lava	X		X					
Short Agent + Water	X							X
Resized Shaded Kits			X	X				
Stimpacks + Poison					X	X		
Level = 3								
Lava + Supreme + Short Agent	X						X	X
Obstacles + Slime + Stimpacks	X	X			X			
Poison + Resized Shaded Kits			X	X		X		
Level = 4								
Complete	X	X	X	X	X	X	X	X

A.3 Seek and Kill

Table A.3: Task difficulty attributes of the Seek and Kill scenario

Task	Noisy Textures	Decorations	Enemy Type	Enemy Size	Enemy Rendering
Seek and Kill					
Level = 0					
Default					
Level = 1					
Blue	X				
Red	X				
Obstacles		X			
Resized Enemies				X	
Shadows					X
Mixed Enemies			X		
Invulnerable			X		
Level = 2					
Blue + Shadows	X				X
Obstacles + Resized Enemies		X		X	
Red + Mixed Enemies	X		X		
Invulnerable + Blue	X		X		
Resized Enemies + Red	X			X	
Shadows + Obstacles		X			X
Level = 3					
Blue + Mixed Resized Enemies	X		X	X	
Red + Obstacles + Invulnerable	X	X	X		
Resized Shadows + Invulnerable			X	X	X
Level = 4					
Complete	X	X	X	X	X

A.4 Dodge Projectiles

Table A.4: Task difficulty attributes of the Dodge Projectiles scenario

Task	Noisy Textures	Decorations	Projectile Type	Homing Missiles	Rapid-Fire	Agent Height
Dodge Projectiles						
Level = 0						
Default						
Level = 1						
City	X					
Flames	X					
Flaming Skulls		X				
Mancubus			X			
Barons			X			
Cacodemons			X			
Resized Agent						X
Level = 2						
Revenants			X	X		
City + Resized Agent	X					X
Arachnotron			X		X	
Barons + Flaming Skulls		X	X			
Cacodemons + Flames	X	X		X		
Mancubus + Resized Agent			X			X
Level = 3						
Flames + Flaming Skulls + Mancubus	X	X	X			
Resized Agent + Revenants				X	X	X
City + Arachnotron	X		X		X	
Level = 4						
Complete	X	X	X	X	X	X

Appendix B

Task Modifications

In this chapter, we describe how all base tasks of scenarios have been created. The modifications of every *default* is presented in relation to the original map in the *ViZDoom* platform, and the variations of higher level tasks are described in comparison to the *default* task. The tasks not mentioned in the following list are combinations of base tasks, i.e., the base tasks, of which the unmentioned higher level task name is comprised, were combined to create it.

B.1 Defend the Center

1. **Default**
   - Original *defend_the_center.wad* from *VizDoom* [39]
2. **Gore**
   - A total of 15 decoration items of 3 different types with in-game id's [26, 28, 29] are constructed at fixed locations
3. **Mossy Bricks**
   - Wall texture *A-MOSBRI*
   - Floor texture *FLOOR4_6*
   - Ceiling texture *CEIL5_1*
4. **Stone Wall**
   - Wall texture *FLAT5_7*
   - Floor texture *FLOOR1_6*
   - Ceiling texture *CEIL3_3*
5. **Fuzzy Enemies**

- The actor property *APROP_RenderStyle* of enemies is set to *STYLE_Fuzzy*

6. **Resized Enemies**
   - The actor properties *APROP_ScaleX* and *APROP_ScaleY* of every enemy are randomized in the range of [0.3, 3.0]
7. **Fast Enemies**
   - The actor property *APROP_Speed* of enemies is set to 20
   - The actor property *APROP_DamageFactor* of the agent is set to 0.3
8. **Flying enemies**
   - New enemy: *LostSoul*

B.2 Health Gathering

1. **Default**
   - Original *health_gathering.wad* from *VizDoom* [39]
2. **Obstacles**
   - A total of 17 obstacles of 3 different types with in-game id's [30, 32, 36] are constructed at fixed locations, blocking the agent's movement
3. **Slime**
   - Wall texture *A-CAMO3*
   - Floor texture *SLIME04*
   - Ceiling texture *GRASS2*
4. **Lava**
   - Wall texture *AQCONC14*
   - Floor texture *LAVA1*
   - Ceiling texture *CEIL1_1*
5. **Water**
   - Wall texture *AQSECT09*
   - Floor texture *FWATER1*
   - Ceiling texture *AQF052*
6. **Stimpacks**
   - New item: *Stimpack*

- Half of the *MediKits* are replaced with *Stimpacks*, which grant 10 health on picking up

7. **Supreme**
   - Original *health_gathering_supreme* scenario from *VizDoom* [39]
   - Removed *Poison*
8. **Resized Kits**
   - The actor properties *APROP_ScaleX* and *APROP_ScaleY* of every health item are randomized in the range of [0.3, 3.0]
9. **Resized Agent**
   - The actor property *APROP_ViewHeight* of the agent is randomized between [0.0, 100.0]
10. **Poison**
    - New item: *Poison*
    - 2 vials of *Poison* are spawned at the start of the scenario
    - A new vial of *Poison* is spawned after every 30 game ticks
    - Poison inflicts 30 damage to the agent
11. **Shaded Kits**
    - The actor property *APROP_RenderStyle* of every health item is set to *STYLE_Shaded*

B.3 Seek and Kill

1. **Default**
   - Original *health_gathering_supreme.wad* from *VizDoom* [39]
   - Wall textures *ICKWALL3*
   - Floor texture *AQF051*
   - Ceiling texture *FLAT19*
   - The surface no longer inflicts damage to the agent
   - Enemies are spawned at random locations around the map
   - Enemies are rendered immobile by setting the actor property *APROP_Speed* to 0
   - The default enemy type is *Demon*
   - The number of initially spawned enemies is set to 20
   - The spawn delay of every subsequent enemy is set to 30 game ticks

- The agent is equipped with a pistol and 200 bullets

2. **Obstacles**
   - A total of 25 elements with in-game id's [44, 45, 46] are constructed at fixed locations of the map, which act as obstacles
3. **Blue**
   - Wall texture *FIREBLU2*
   - Floor texture *FLAT14*
   - Ceiling texture *CEIL4_3*
4. **Red**
   - Wall texture *FIREWALL*
   - Floor texture *CRACKLE4*
   - Ceiling texture *DORED*
5. **Resized Enemies**
   - The actor properties *APROP_ScaleX* and *APROP_ScaleY* of enemies are randomized in the range of [0.3, 3.0]
6. **Shadows**
   - The actor property *APROP_RenderStyle* of enemies is set to *STYLE_Shadow*
7. **Mixed Enemies**
   - Removed enemy: *Demon*
   - Added enemies: *ZombieMan*, *MarineChainsaw*, *DoomImp*, *Cacodemon*, *LostSoul*
   - The number of initially spawned enemies is set to 3 per type
8. **Invulnerable**
   - Removed enemy: *Demon*
   - Added enemies: *Arachnotron*, *Fatso*, *PainElemental*, *Archvile*, *Revenant*
   - The number of initially spawned enemies is set to 3 per type
   - The actor property *APROP_Invulnerable* of the agent is set to *True*

B.4 Dodge Projectiles

1. **Default**
   - Original *defend_the_line.wad* from *VizDoom* [39]
   - The agent is not granted any weapons nor ammunition
   - 10 enemies of type *DoomImp* are spawned at equal distance from one another
   - Enemies are rendered immobile by setting the actor property *APROP_Speed* to 0
   - The actor property *APROP_DamageFactor* of the agent is set to 0.7
2. **Flames**
   - Wall texture *SWATER1*
   - Floor texture *RROCK02*
   - Ceiling texture *CAVERN1*
3. **City**
   - Wall texture *SKY2*
   - Floor texture *ROCKRED1*
   - Ceiling texture *CEIL4_1*
4. **Flaming Skulls**
   - A total of 7 decorations, with an in-game id of 42, are constructed at fixed locations, impeding the agent's line of sight
5. **Resized Agent**
   - The actor property *APROP_ViewHeight* of the agent is randomized between [0.0, 100.0]
6. **Cacodemons**
   - New enemy *Cacodemon*
   - The number of enemies is reduced to 5
   - The actor property *APROP_DamageFactor* of the agent is set to 0.5
7. **Barons**
   - New enemy *Baron*
   - The number of enemies is reduced to 5
   - The actor property *APROP_DamageFactor* of the agent is set to 0.4
8. **Revenants**

- New enemy *Revenant*
- The number of enemies is reduced to 3
- The actor property *APROP_DamageFactor* of the agent is set to 0.3
- The projectiles attacks are *homing*, meaning they trace the agent, making dodging more difficult

9. **Mancubus**
   - New enemy *Mancubus*
   - The number of enemies is reduced to 2
   - The actor property *APROP_DamageFactor* of the agent is set to 0.5
10. **Arachnotron**
    - New enemy *Arachnotron*
    - The number of enemies is reduced to 1
    - The actor property *APROP_DamageFactor* of the agent is set to 0.3