Bootcamp Method for Training General Purpose AI Agents

Vincent Lombardi
School of EECS
Washington State University
Pullman, WA USA
vincent.lombardi@wsu.edu

Lawrence Holder
School of EECS
Washington State University
Pullman, WA USA
holder@wsu.edu

***Abstract*— General purpose agents have long been an ultimate goal of AI research. One promising approach to this goal is to first train an agent to use a variety of skills, called a skillnet agent, and then allow the agent to learn how to choose the appropriate skill instead of having to choose the appropriate low-level action. We propose a method for training skillnet agents called Bootcamp that helps agents efficiently learn basic skills in an environment. We found that Bootcamp agents outperform skillnet agents trained randomly on various tasks defined in the ViZDoom simulated environment. We also found that skillnet agents outperform more conventional reinforcement-based learning approaches such as DQNs in ViZDoom.**

Keywords—ViZDoom, Reinforcement Learning, Machine Learning, Regular Research Paper

I. Introduction

Our objective is to make a general-purpose agent that can solve or at least be easily trained on multiple different tasks quickly and effectively. We specifically focused on Reinforcement Learning (RL) agents. RL agents are AI agents that learn tasks by performing them over and over while receiving rewards based on their performance. In general, these agents are inflexible which means that they often have to be retrained from scratch for each new task. If we can shorten this process by creating a general-purpose agent, this will make RL agents significantly more useful.

Our main focus was on general video game play (GVGP) as it allows us to test a wide variety of environments. We further refined this to focus on the ViZDoom [1] environment where we created a variety of tasks. ViZDoom was originally designed to support AI agents through visual interaction, i.e., the simulator provides a current image of the agent's view, and then the agent returns an action. However, we have modified the interface to provide a feature vector, or sensor vector, that provides additional information about the environment. This shift from image-based to sensor-vector-based interfaces allows the agent to receive more information about the environment and allows our methods to apply to a larger number of environments.

In addition to the skillnet and Bootcamp approaches to the ViZDoom tasks, we also approached these tasks with both a DQN [2] and A3C [3] based approach. We found that skillnet agents outperform DQN's by a large margin. A skillnet agent is an agent that uses pretrained skills to accomplish a complex task rather than just using base actions. We further improved this by taking a Bootcamp or structured approach to the agents training. We found that a skillnet agent trained via a Bootcamp approach outperformed a skillnet agent trained randomly. Furthermore, we found that our Bootcamp agent can outperform similar skillnet agents on more complex but related tasks.

This research was sponsored in parts by the Defense Advanced Research Projects Agency (DARPA) and the Army Research Office (ARO) and was accomplished under Cooperative Agreement Number W911NF-20-2-0004. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the DARPA or ARO, or the U.S. Government.

II. Related Work

A. Multi-Task Agents

Google Deepmind was able to create a generalized agent (GATO) [4]. This agent used natural language processing techniques to tokenize an environment. The advantage of this approach is that it can process tasks with variable input and output sizes. The downside to this method is that it needs training data from other RL agents which might be hard to get. As a result, this method is not as adaptable in novel situations. Google Pathways is similar to GATO in that it is a way to set up a neural network to remember multiple different tasks [5]. Pathways has different parts of its network assigned to different skills and therefore it does not always use its whole network. Our approach does not break the network down by skill and therefore the whole network is used every time. Unlike GATO our agent uses no natural language processing and therefore does not require the input to be tokenized.

General purpose agents often have issues learning new tasks and remembering old ones. Du et al. [6] propose a solution for this in the form of dreaming. Dreaming is where periodically the agent is restored to a previous state so that it can attempt an old environment with a new set of skills. Similarly Golden et al. [7] propose interweaving previous tasks in an offline learning period. Zhang et al. [8] address remembering old tasks during new task training by maintaining stability in network layers.

A few studies have also mixed older tasks into the training process of new tasks to improve performance. Silver et al. [9] propose remembering old tasks in lifelong learning and transferring important parts to new tasks. Saxena et al. [10] propose interweaving similar older tasks into newer training tasks to speed up learning.

B. Hierarchical Agents

Skillnets are explored more in depth in [11] where they were used in ViZDoom. This agent used image-based learning and was DQN based. This agent only had a navigation and combat skill, so it did not use our Bootcamp approach. However, it does

979-8-3503-2759-5/23/$31.00 ©2023 IEEE
DOI 10.1109/CSCE60160.2023.00007

show the validity of our approach to using skill agents. This agent also was not trained for as long as our agent.

Hierarchical agents or agents with different levels of skills are addressed in [12]. Similar ideas were explored as early as 1999 in [13]. Skills are somewhat similar to combo actions introduced in [14]. These are where the agent essentially selects an objective, and a few actions are carried out. Our skills are more complex, but an agent has to continuously choose to use them. Togelius and Yannakakis [15] present a paradigm for general game play. Specifically, it talks about separating tasks from games. The paper also touches on generating both levels and entire games with AI.

Song et al. [16] created a similar hierarchical agent that played Doom called Starnet. This agent used imaged based systems to accomplish a wide variety of AI generated maps that ranged in difficulty. Starnet used a two-level hierarchical agent. They dealt with the sparse reward problem by rewarding agents for attacking, using tools, collecting resources, and even looking at enemies. Their agent like many others was image based and therefore differed drastically in implementation from our own agent. Starnet also is constructed with a distinct set of goals or options in mind which helps alleviate the delayed reward problem.

The idea of a skillnet or network that can utilize multiple skills was explored by Zhang et al. [17]. In general, skillnets are agents that can accomplish multiple different tasks using smaller skill networks. Their skillnet focused on natural language processing and had access to multiple different skills. Not all of these skills are accessed at any one time.

There are other examples of hierarchical learning in environments other than ViZDoom/Atari. Tessler et al. [18] describe an approach to playing Minecraft with a Hierarchical Deep Reinforcement Learning Network H-DRLN. Specifically, this agent used reusable skills to facilitate lifelong learning. This agent is very similar to ours in that it uses a pretrained navigation skill that is a Deep Skill Network (DSN). A DSN is an RL agent that has been trained to use a skill.

C. Curriculum Learning

Our technique is also similar to curriculum learning in [19] with a few differences. Curriculum learning is where you introduce an agent to new harder tasks as it trains. In this way it learns basic skills first. Mainly we do not scale our Bootcamp agent's difficulty for individual tasks. Instead, we start with easy tasks first and then run the agent on harder tasks once it has learned how to do the easier ones. All these techniques help deal with sparse reward environments such as ViZDoom.

An example of curriculum learning is Rho et al. [20] where an agent was trained how to box. Researchers started by making the agent stand, then walk, and finally, punch. One major difference with our approach is that the agent does not necessarily learn one skill before the others. In the boxing environment tasks are built off each other so it was hard for the agent to forget previous tasks or actions.

Kodaka and Saitoh [21] talk about using curriculum learning in a top-down shooting game where an agent was trained on increasingly difficult tasks. The increase in difficulty was mainly in enemy aggression so there were no skills to be lost here unlike in our experiments. Zhang et al. [22] focused on curriculum learning in Starcraft II. The difficulty measure here was the in-game difficulty were at higher levels the in game AI was able to cheat. The agent also avoids forgetting skills at higher levels since higher levels still required skills from earlier levels.

D. Actor Critic

Advanced Actor Critic is a form of deep reinforcement learning that is quite common in both previous work and our own approach. It consists of two neural networks an actor and a critic. The actor picks and preforms actions while the critic learns to judge said actions. The two most common forms of this are Synchronous Advanced Actor Critic (A2C) [2] [23] and Asynchronous Advanced Actor Critic (A3C) [2] [3]. A2C creates one set of these networks while A3C creates multiple which periodically check in with a global policy to improve the quality and speed of learning.

Proximal Policy Optimization or PPO [24] is similar to A2C, but it uses a different method for calculating the gradient. PPO uses a simplified version of TRPO that uses a penalty instead of a constraint to help compute the KL divergence. KL Divergence is how both agents measure how much the policy has changed. We used a single environment version of this as both a control and a modified Bootcamp agent.

III. Bootcamp Approach

In order to improve the use of skills, we introduce the Bootcamp agent, where the agent is trained on alternating objectives so that it learns different tasks. We then put it in a test environment where it must combine multiple skills to complete the tasks. Our specific idea is to structure how we alternate objectives. As this is an A3C agent we have different instances start on different tasks teaching different skills. These agents then play a few rounds of their starting task and then move on to the next tasks and repeat the process in a loop. This way the agent will not overspecialize at any one objective.

We also provide the Bootcamp agent with minimum objectives for each task, and the agent will retrain if it does not meet these objectives. Essentially if the Bootcamp agent does not complete a certain number of objectives of each type in the last quarter of training, it has to restart. This is a form of quality control that ensures that the skill has been taught. An objective is picking up an item or defeating an enemy, but it is not necessarily the same as winning. For example, if the task is to defeat multiple enemies, each enemy defeated counts regardless of if the agent wins. Similarly, if the objective is to pick up ammo, but the agent picks up health or defeats an enemy, these will also count even if the agent does not end up picking up the ammo.

We used three types of agents in this experiment as shown in Table I. A Bootcamp agent is a skillnet and used our experimental approach. A Control-Skill is an agent which is a skillnet that did not use our experimental approach. Finally, we used a base agent or Control-Base agent which did not use skills. The skillnet agents usually had a choice between objectives

related to completing the tasks while the Control-Base agents had access to the direct actions themselves.

TABLE I. AGENT FEATURES

Agents	Order	Asynchronous	Quality Control	Skill
A3C Boot	Y	Y	Y	Y
A3C Control-Skill		Y		Y
A3C Control-Base		Y		
PPO Boot	Y		Y	Y
PPO Control-Skill				Y
DQN Control-Base				

We found that skillnets can have their skills atrophy. If a skillnet is trained in such a way that all skills are useful, it will be able to adapt to each task. If a skill has atrophied, then the agent will likely never regain use of it. As a result, we have to train the agents on multiple different tasks either at the same time or in quick succession. This is why we iterate over objectives in small batches instead of playing large numbers of episodes of each task before moving on to the others.

For the A3C agent we did use a different number of tasks compared to the number of asynchronous agents or threads. We did this because the agent still might focus more on one task at a certain time by having a greater number of agents focus on the task. The hypothesis here was that this would help the agent learn more complex tasks along with easier tasks which it would otherwise focus on. Otherwise, the agent would focus on the easiest tasks at the expense of the harder ones. For example, if one task only required the agent to use one skill it might favor that over a task that required multiple. Note that the agent does change this focus as every single asynchronous agent will cycle through all of the tasks during training.

The Control-Skill agent variants use the same number of threads or asynchronous agents but lack the order and quality checks. The Control-Base agent variants do not have skills, order, or quality control and may have a different input . The reason that the Control-Base agents have different inputs is that they might either need more information to carry out actions such as the locations of obstacles that a navigation skill would avoid. They also do not need input for when a certain skill is usable as they have no pretrained skills. The Control-Skill and Bootcamp agents should have the same inputs and outputs.

For PPO we used only a single environment which meant that we were not able to have the agent do multiple tasks at the same time. We had to use a modified Bootcamp approach that still had the order and the quality control checks in place as shown in Table I.

IV. DOMAIN

We used six tasks total with the first three serving as the Bootcamp tasks. The purpose of these first three was to teach the agent how to use each skill or carry out each objective. The last three were meant to test how the agent would adapt its skills to a new environment. This meant that tasks 1-3 were not as difficult as the tasks 4-6 which were meant to test how the agent could apply its skills. Table II summarizes the six tasks used to evaluate our agents. Task 1 teaches combat and involves killing all enemies onscreen and has 2 enemies, 0-1 health packs, and 0-1 ammo packs. Task 2 focuses on getting health and involves collecting all health packs. Task 2 has 0-1 enemies, 2 health packs, and 0-1 ammo packs. Task 3 is like task 2 except it focuses on getting ammo instead of health, so the objective is to pick up all of the ammo packs. Task 3 has 0-1 enemies, 0-1 health packs, and 2 ammo packs.

Task 4 involves 3 enemies, 3 health packs, and 3 ammo packs. Furthermore, the agents only start with enough ammo to kill one enemy. In this task ammo packs also give more bullets. The idea here is to see if the agent can apply its skills to a more difficult environment. More enemies means that the agent must use its combat skill more and likely will have to heal more often. The lack of starting ammo also means that the agent must use its ammo collecting skill to win.

Task 5 is very similar except the ammo packs now give four shots instead of three and enemies respawn. The fact that enemies respawn makes task 5 much more difficult, because the agent could run out of ammo before killing all the enemies which should not be possible in the other tasks. More ammo was given per ammo pack to make up for the fact that the agent starts with the same amount of ammo as it does in task 4 even though it will likely face more enemies.

The map for tasks 1-5 is shown in Fig. 1. All five tasks use the same room layout but have randomly placed items such as heath, ammo, and obstacles. All 5 tasks have specifically 1 obstacle alongside the normal walls that are present in the environment. They also have random enemy and player starting positions. To configure this for task 1 from Table II you would need to set it so that two enemies are guaranteed to appear. You would then make at most one health pack and one ammo pack spawn. You also would change the victory conditions to match the stated objective shown in Table II.

Task 6 is a modified version of the standard Deadly Corridor task that comes with ViZDoom. This task involves running down a hallway lined with enemies to get an armor pickup. There are three types of enemies of varying levels of strength. In the base environment these enemies have a strict layout. There are also invisible walls between the enemies and the player that only allow bullets through. Enemies drop weapons and ammo when killed but these items are not supposed to be accessible though it is possible for the agent to pick up the weapons through the wall. Our modified version randomly determines which enemies spawn and what type they are. The enemies at the end of the corridor will always spawn as otherwise the environment would become too easy. We also removed the invisible walls to allow the agent to pick up ammo. The enemies are still locked in place even though the invisible walls are gone as their speed has been set to 0. Technically it is also possible for the agent to pick up weapons but there is no skill for this. In task 6 the health skill will pick up armor packs instead of health packs as there are no health packs in the environment. The reasoning behind task 6's inclusion was to use a map that was different from the other tasks. As a result, the skills behave a bit differently and we wanted to test the agent's adaptability.

Task 6's map is shown in Fig. 2 and comes with ViZDoom itself. You must modify the script so that enemies are only guaranteed to spawn at the end of the hallway and so that enemy

type is randomized. Fig. 2 specifically shows the original version of the task where enemies are placed directly on the map. In our version a script runs at the start of the game to randomly decide which enemies get to spawn.

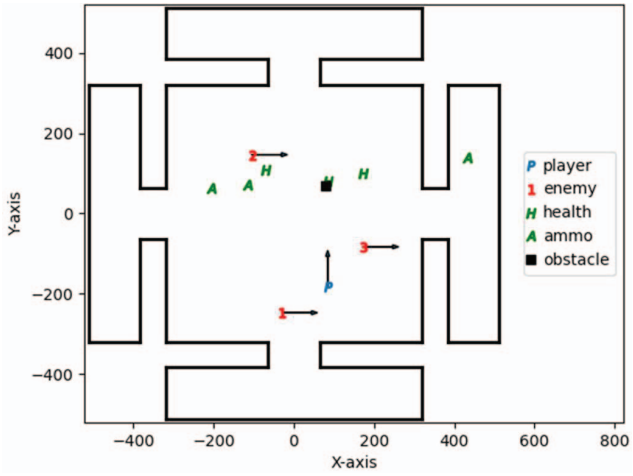

Fig. 1: General map for tasks 1-5.

TABLE II. DESCRIPTION OF TASKS

Task	Enemies	Ammo (starting)	Health	Actions Per Game	Condition
1	2	0-1 (20)	0-1	1500	Defeat enemies
2	0-1	2 (20)	0-1	1500	Collect ammo
3	0-1	0-1 (20)	2	1500	Collect health
4	3	3 (5)	3	2000	Defeat enemies
5	3 respawning	3 (5)	3	2000	Defeat enemies
6	2-6	0-6 (26)	1 armor	1500	Get armor pickup

The DQN's must be trained on easier versions of tasks 1-5. They are given much more ammo to start with, specifically 2000 shots in task 1. There are no obstacles in the environment and all enemies spawn in the middle room. The reason we had to do this is that the DQN was unable to solve the original versions of these tasks when given a sensor vector. Even when given the wall information the DQN agent would not make forward progress. Learning to evaluate if it was going to hit a wall and hit an enemy appeared to be too much for the DQN. For task 4 the DQN still received the same amount of ammo as the other agents, but the enemy's aggression was reduced to account for the fact that they now spawned in firing range of the player. Task 6 was the same for all agents however we used a different Control-Base agent that was purposely trained only on task 6 as the agents originally trained on tasks 1-3 did very poorly. Furthermore, due to a difference in input size we would have had import only part of the original agent which combined with its poor performance on other tasks led us to simply use a new agent.

We also used an A3C and a PPO version of the Control-Base agent which had a similar reward function and setup as the DQN. For task 6 we trained our Control-Base agents from scratch, as task 6 required a different input size and the tasks 1-5 counterparts were not very effective.

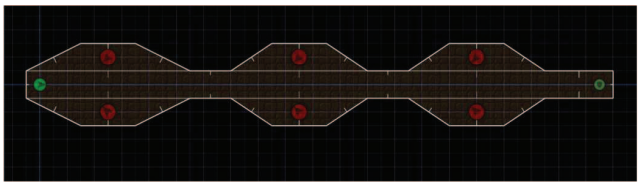

Fig. 2: Map for task 6: The green arrow icon is the player, red arrow icons are enemies, and the green shield icon is the goal. The faded horizontal between the enemies and the main hallway are invisible barriers that block the player from crossing but allow bullets to pass through.

V. SKILLS

We have four skills present in tasks 1-5: combat, navigation, ammo collection, health pack collection. These skills correspond to different objectives but in the case of combat both the navigation and combat skills are required to complete the task.

The combat skill checks if an agent can hit an enemy, turn and hit an enemy, or move left or right one step to hit an enemy. It also checks if obstacles/walls are blocking the agent's line of sight towards an enemy. The combat skill specifically does not fire diagonally through doors as it was determined to be unreliable with our implementation. The skill also checks if the agent has enough ammo to fire but not if it has enough ammo to defeat its opponent.

The navigation skill is an A3C agent that was trained on the same map that tasks 1-5 use. It works by getting a set of coordinates and moving towards its objective. The agent will use this skill to navigate to adjacent rooms so sometimes an intermediate room needs to be picked if the final target is not in an adjacent room. This decision is made by the navigation skill and is implemented via a simple check to see if the final target room is an adjacent room. If it is not, the navigation skill will move to the central room which is adjacent to all other rooms. For example, if the agent was in the east room and needed to go to the north room the navigation skill would be used to navigate to the center room first before being told to go to the north room. The ammo and health skills are very similar to the navigation skill except they are given the coordinates for ammo packs and health packs instead of the center of each room respectively.

The navigation agent also does have some more scripted behavior that helps it go through doors without getting stuck, but it is mainly an RL agent. It gets rewarded for getting closer to its objective and penalized for running into walls. In the case of task 1-5 the navigation agent will also receive a reward for getting into the hallway sections.

To train the A3C navigation skills we simply dropped the agent in the same map as the one we used for our tasks and gave it a point to move towards. We had it play around 2000 games of moving between rooms or moving towards a randomly selected point on the map to simulate items. Our agent was trained to also dodge obstacles which we added in later via curriculum training, but this is not required. The reward function is simple as we gave it a small reward for getting closer to its

goal and staying in the maps cross section so it could easily get through doors. For the ammo and health skills we did not include the cross-section part as the agent would always be told to move towards a point in its current room. To add a new skill, you just load in the middle part of the network and use a new input and output layer.

Task 6 did not have a navigation skill as it would have been the same as the health skill. The combat skill like before turned to face enemies and fired upon them. The combat skill did not move left or right in task 6 as the agent had a finer control over its turning radius so moving left and right would be redundant. The ammo skill pointed the agent at an ammo pack and moved the agent towards it. The health skill pointed at the armor pack at the end of the hallway and drove the agent towards it. Since health and armor are similar, we chose simply to recycle the health skill for this instead of making a new skill or changing the pickup type. None of the task 6 skills were RL based so there was no training. This was due to the relative simplicity of navigation and shooting in the environment compared to the other tasks.

VI. Evaluation

A. Training Methodology

As shown in Table I three types of agents were used in this experiment, a Bootcamp agent, a Control-Skill agent, and a Control-Base agent. The agents are trained on a set of episodes sampled from tasks 1-3 (see Table II). Both the Control-Base and Control-Skill agents are trained from a random sample of the three training tasks until convergence. The Bootcamp agent is trained with a more structured curriculum from these tasks. Specifically, the Bootcamp agent does the tasks in groups of four. It does four combat games, then four ammo packs, and finally four health before looping back to combat. As we are using A3C, we stagger this so that each of the individual agents start 3 games ahead of the previous. If you have four threads, one will start at combat game 1, the second starts at combat game 4, the third starts at ammo task 3, and the final thread starts at health game 2. Training episodes were generated at random with different seeds while the testing episodes were generated randomly using the same seed. The agent was trained on tasks 1-3 at the same time so that it could learn to use all of its skills at the same time.

The Bootcamp agent is trained for 2000 episodes and must eliminate 250 enemies in the last 500 training episodes to be considered successful. The Bootcamp agent also must collect 150 ammo packs and health packs in the last 500 episodes. These criteria were determined with some trial and error as well as realistically determining roughly how many objectives the agent could potentially complete. We train the agent for 2000 episodes as we found that past 2000 the agent improved less. The reason that there are more required enemies is that eliminating enemies is more complicated than picking up items. The control agents do not have this requirement as it would give the control agents to much implicit information about the environment . Part of the Bootcamp approach is giving agents a quality control check to see if it makes a difference over the control. These agents will be tested for 1000 episodes that are randomly sampled from the training tasks with learning disabled. Their raw scores will be averaged over all episodes.

Tasks 4-6 were our test tasks. Agents are trained and tested on task 4-6 for 1000 different episodes. Agents started with the weights from tasks 1-3 and were trained on tasks 4-6 individually. A fresh agent was trained from the task 1-3 weights for each test task. The performance is averaged over 1000 episodes along with the agents' raw score and performance. Task 5 is the same as task 4 except the agents can get more points for eliminating more enemies as the enemies respawn.

For tasks 4-6 we also measure the transfer learning metrics jumpstart and asymptotic performance. Jumpstart is how well the initial performance on a task can be improved by transfer learning. Asymptotic performance is how much the final performance on the task can be improved by transfer learning. For jumpstart we include only the first 200 episodes and for asymptotic performance we used the last 200.

The skillnet agents use a neural network to select which objective it wants to carry out and then a skill executes it. In our case this selection component was an A3C agent. Our health, ammo, and basic navigation skills were implemented with A3C as well as some more scripted actions. The combat skill was implemented purely with scripted actions and simply calculates if an enemy can be hit from a current location. The combat skill also has the limited ability to turn the agent and move from side to side.

For input the agent gets health, ammo, current angle, and its current coordinates. It also gets the targeted enemy's current coordinates, relative distance, relative angle, and health. Targeted enemies are determined by whichever enemy closest or can be easily attacked. The agent gets the targeted ammo and health pack's current coordinates and distance. It gets the number of enemies and ammo packs remaining. It also is notified if there is an ammo pack in range if the agent is less than 200 meters from the ammo pack.

B. Reward Function

A skillnet agent receives -1 point per turn. It receives +1 for damaging enemies, and -2 for using the combat skill incorrectly. The agent also receives -2 for using the ammo or health skills if those skills are not useful at the time. An example of this is using the combat skill when no enemies are present. It receives +0.5 if using the ammo or health skills in their respective tasks. It receives +10 for completing the task in under 750 actions. 100 points are given for kills with +20 for under 100 actions and +10 if it is over 100 actions but under 400. The agent gets +10 points for ammo and health. It gets +50 for getting an item with the correct skill. It also gets +15 for getting an item during its assigned task. The agent also gets punished with -1 points for taking damage. The performance of the agent is somewhat sensitive to the choices of these parameters, so some tuning of the reward function is necessary to maximize agent performance.

The DQNs differ in that they get no rewards or penalties for using the correct skills as they have none. The agents also get more points for completing the tasks. The reason the agents need to get more points for completing a task is that they are more likely to simply run away from a problem. Tasks 4 and 5 also receive different input as they do not get the +0.5 points for using correct skills. They also lack the +15 point reward for getting

items during their respective tasks. The A3C, A2C, and PPO Control-Base agents have a similar reward function to the DQN but with some different values. A3C, A2C, and PPO get rewarded less for completing the tasks as this improved their stability. A3C, A2C, and PPO all get +40 points for inflicting damage to enemies instead of the DQN's +25 points. Otherwise, they get similar points to the skillnet agents for completing objectives. The DQN gets -300 points for dying and +500 points for winning. It also gets +200 points for completing and objective, i.e., defeating an enemy or getting and item. These values were determined through trial and error as well as testing.

Previous attempts at designing the reward function gave a negative point for each action and a large positive reward at the end. After this method failed, we moved to reward the agent for killing enemies to give it partial credit. Next, we added rewards for getting health and ammo packs. For the Bootcamp specifically, we realized that the agent may not learn to use a skill that is used infrequently so we added small rewards for using a skill when it was appropriate. Finally, we added extra points for completing objectives quickly to emphasize the need to complete the task quickly. Overall, the motivation behind the reward function was to teach the agent how to use the skills.

VII. Results

On the base set of tasks, we have seen that the Bootcamp agent outperforms the Control-Skill agent by a small but consistent margin. In general, the Bootcamp agent does 10% better than the Control-Skill agent in number of victories and performance score. This is shown in Table III where the average performance for the Bootcamp agent outperforms the control skill agent. The Bootcamp agent's raw score is significantly better on the same set of tasks than the Control-Skill agent. The DQN Control-Base agent does worse on all three tasks. While the raw score for the DQN appears to be the highest of the three agent types, it is misleading due to the fact that the DQN has a different reward function. This is backed up by the DQN's low number of wins which indicates that despite its high score it cannot effectively complete the tasks. The A3C agent performs similarly to the DQN.

Fig. 3 and 4 show the average training performance for tasks 1-3 and 4 respectively. This shows that the Skill agents overall learned better than the Control Base agents. It also shows the A3C Bootcamp learning the best in task 4. The reason that the graphs start at episode 200 instead of 0 is that it takes about 200 episodes for the average performance to stabilize. Prior to episode 200 the graph looks erratic as outliers have a larger effect.

The PPO Control-Base agent performed significantly better than the other base line agents possibly because the algorithm was better suited to the task and possibly due to be structurally a bit different in that its intermediate layers were smaller. The scores however are deceptive as the PPO Control-Base agent still was playing the same easier versions of all of the tasks that the other Control-Base agents played. PPO's Control-Base agent did not do well at tasks 4-6 and was outclassed by the Bootcamp A3C agent.

TABLE III. Tasks 1-3 Metrics

Average	Wins	Raw	Perf
A3C Boot	665.83	-839.25	0.65
A3C Control-Skill	562.10	-1031.18	0.56
A3C Control-Base	86.63	-1183.91	0.13
PPO Boot	631.63	-787.37	0.63
PPO Control-Skill	611.70	-789.26	0.61
DQN Control-Base	51.83	-7.66	0.09

The gap between the A3C Bootcamp and A3C Control-Skill agents was much larger in task 4 as shown in Table IV. This gap can be explained in part by the fact that the A3C Control-Skill agent did not have the same quality control checks that the Bootcamp agent was given. However, this still does not account for the entire gap, because even if we remove all of the trials based on Control-Skill agents that won less than 500 games, we still end up with 23 agents that average only 280 wins out of 1000 compared to the Bootcamp's 359/1000 average victories. This same pattern can be seen across all metrics, which still shows the gap. The Control-Base agents also did not do well at this task averaging close to 0 wins.

For task 5 the gap between A3C Bootcamp and A3C Control-Skill was significantly smaller in large part due to the increased difficulty. Once again, the A3C Bootcamp agent outperformed the A3C Control-Skill agent in all metrics though by a narrower margin as shown in Table V. The Control-Base agents were unable to solve this task. While their raw scores seem high this is just a result of them dying quickly enough to not accumulate a large number of negative points.

Task 6 experiments are the one time where the A3C Bootcamp agent did not do better than the A3C Control-Skill agents as shown in Table VI. We found that the Control-Skill agent won more games than its counterparts but did poorer or tied with other agents in other metrics. One explanation for this is that task 6 had one less skill and therefore the skill agents were not loaded completely. Adding a fourth skill to task 6 led to a marked drop in performance for all agents. The skill that was cut was the default task which did not have an input like the other skill to tell an agent when it was useful as it was always supposed to be useful. As the agent could not be easily informed of this change this could have led to unexpected results. The A3C Control-Base agent did win a few games but still did not perform as well as its counterparts. This can be explained by the fact that one somewhat effective strategy is to simply charge down the corridor at the goal. This will sometimes let you win and was adopted frequently by Control-Base A3C.

We also compared our algorithm to a PPO based approach. The PPO Bootcamp agent did better than most other agents in tasks 1-3 with the exception of the Bootcamp A3C agent, the PPO skill agents had a higher raw score than the Bootcamp A3C agent but had a lower number of wins and lower average performance. The difference in results here is likely just that PPO was able to find a strategy that improved intermediate rewards at the expense of winning. The PPO Bootcamp agent did worse on tasks 4, 5, and 6 than the A3C Bootcamp agent.

TABLE IV. TASK 4 METRICS

Average	Wins	Raw	Perf	Jump start	Asymp Perf
A3C Boot	358.97	-2002.55	0.46	0.53	0.48
A3C Control-Skill	218.13	-2179.76	0.36	0.42	0.37
A3C Control-Base	0.03	-1004.95	0.03	0.03	0.03
PPO Boot	296.87	-1214.86	0.30	0.31	0.35
PPO Control-Skill	342.10	-1240.99	0.34	0.28	0.36
DQN Control-Base	0.00	-199.57	0.00	0.01	0.01

TABLE V. TASK 5 METRICS

Average	Wins	Raw	Perf	Jump start	Asymp Perf
A3C Boot	144.03	-1975.27	0.22	0.23	0.29
A3C Control-Skill	72.67	-2066.88	0.13	0.14	0.21
A3C Control-Base	0.00	-973.02	0.00	0.00	0.00
PPO Boot	91.43	-1282.39	0.11	0.07	0.09
PPO Control-Skill	93.73	-1289.84	0.10	0.05	0.11
DQN Control-Base	0.00	-196.42	0.00	0.00	0.00

TABLE VI. TASK 6 METRICS

Average	Wins	Raw	Perf	Jump start	Asymp Perf
A3C Boot	122.17	-2070.58	0.37	0.40	0.41
A3C Control-Skill	130.13	-2027.56	0.35	0.40	0.42
A3C Control-Base	39.20	-396.16	0.27	0.28	0.28
PPO Boot	346.40	-873.46	0.50	0.41	0.50
PPO Control-Skill	352.77	-848.57	0.46	0.42	0.50
DQN Control-Base	0.00	-252.15	0.01	0.03	0.04

VIII. DISCUSSION

The Bootcamp approach can be used to quickly help an agent adapt to different tasks. We focused on action-oriented domains where the agent's decisions can affect the environment. Furthermore, for the Bootcamp approach to work, the domain needs to be complex enough to allow for skills. The OpenAI Gym's Cartpole task is an example of an environment that is not complicated enough as the overall objective is too simple to reasonably break down, because the objective is merely to balance the pole. OpenAI Gym's Mountain Car is similar in that there is no real side objective as all the agent does is move left and right to gain speed.

The choice of which information to provide to the agent is typically straightforward. For environments where you control a character on screen you would need to give the agent its basic information such as coordinates, angle, and resources such as ammo and health. You also need to give it the coordinates of objects that it can interact with and identify them by type such as enemies and health. As long as the navigation skill is competent, you do not need to give the agent the coordinates of permanent obstacles. In Atari Asteroids for example you would give the agent the speed and health of each type of asteroid. Environments like Angry Birds are very different, but you would still give the agent basic information about each bird and the location of the pigs as well as the obstacles. Generally speaking, it is better if the input can filter out objectives that are impossible to reach or irrelevant. For example, in ViZDoom if the combat agent thinks it can attack an enemy, we give the detailed information only for that enemy. A skillnet agent gets to choose between attacking or pursuing another objective, while the individual skills figure out how to do this in the most efficient way.

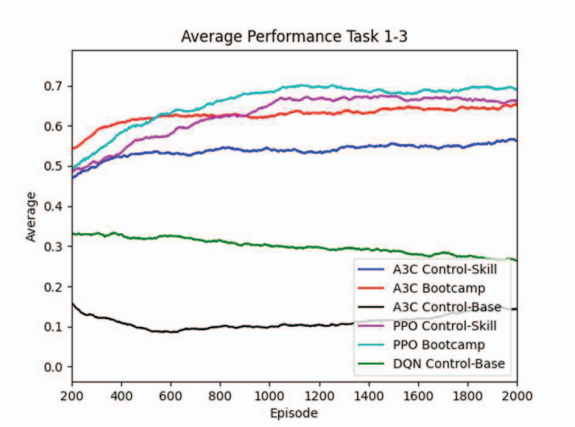

Fig. 3: Tasks 1-3 Training Performance

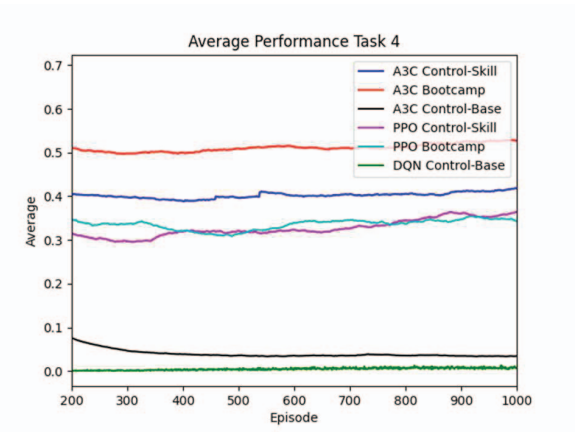

Fig. 4: Task 4 Training Performance

Determining the reward function for a new domain requires some trial an error but is based on how difficult the domain's objectives are and how often the skills are used. The less a skill is used the more the agent should be rewarded for using it. Objectives can usually be determined by when an action involves removing or adding something to the environment. There should not be too many actions that are rewarded while completing an objective. For example, moving towards a position in the environment is not completing an objective but completion occurs when the agent actually reaches the position. Objectives are defined as something a skill is attempting to accomplish, so successfully using a skill leads to a large reward.

IX. CONCLUSION

In this paper we propose the Bootcamp skill-based approach to designing agents and hypothesize that the correct training and use of skills results in agents that are more general purpose, capable of solving different tasks. We implemented this approach using the A3C method to create a generalist ViZDoom agent that can use sensor vectors instead of images. The use of

A3C, ViZDoom, and sensor-vector based inputs are part of the experimental apparatus used to evaluation our hypothesis. Furthermore, we proposed a method of structured training that improves skillnet agent performance in novel environments. We found that skillnet agents outperform DQNs and that Bootcamp agents outperform the average skillnet agent.

One weakness with this approach is that it depends on the skills given to the agent. Skills may not always generalize to other environments, and the agent might have a substantial learning curve. Furthermore, the skillnet is still dependent on having a comparable environment. For example, the Bootcamp agent based on ViZDoom will have no idea what to do if placed in a Cartpole like environment.

Note that specialized agents can outperform the Bootcamp agent in certain scenarios. For task 4 if you trained a specialized skillnet from scratch with the same reward function, the agent will perform similar to the Bootcamp agent especially if it is trained longer. But one of the benefits of the Bootcamp agent approach is to remove this need to train a specialized agent for each new task.

For future work, we would like to apply the approach to image-based environments such as Obstacle Tower [25], ALE, or even more ViZDoom tasks. There are also many policy optimization techniques that could be further explored such as proximal policy optimization (PPO) [24] which could help improve performance.

REFERENCES

[1] M. Kempka, M. Wydmuch, G. Runc, J. Toczek and W. Jaskowaski, "ViZDoom: A Doom-based AI Research Platform," in Proceedings of IEEE Conference of Computational Intelligence in Games 2016, Santorini, 2016.

[2] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," in ICML, New York, 2016.

[3] "A3C," [Online]. Available: https://paperswithcode.com/method/a3c.

[4] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kat, J. T. Springenberg, T. Eccles, J. Bruce, A. Razavi, A. Edwards and N. Heess, "A Generalist Agent," Transactions on Machine Learning Research, 2022.

[5] J. Dean, "Introducing Pathways: A next-generation AI architecture," Google, 28 Oct. 2021. [Online]. Available: https://blog.google/technology/ai/introducing-pathways-next-generation-ai-architecture/.

[6] Y. Du, G. Warnel, A. Gebremedhin, P. Stone and M. E. Taylor, "Lucid Dreaming for Experience Replay: Refreshing Past States with the Current Policy," Neural Computing and Applications, vol. 34, no. 3, 2020.

[7] R. Golden, J. E. Delanois, P. Sanda and M. Bazhenov, "Sleep prevents catastrophic forgetting in spiking neural networks by forming a joint synaptic weight representation," PLOS Computational Biology, vol. 18, pp. 1-31, 18 Nov. 2022.

[8] B. Zhang, Y. Guo, Y. Li, Y. He, H. Wang and Q. Dai, "Memory Recall: A Simple Neural Network Training Framework Against Catastrophic Forgetting," IEEE Transactions on Neural Networks and Learning Systems, vol. 33, no. 5, pp. 2010-2022, 2 Aug. 2022.

[9] D. L. Silver, Q. Yang and L. Li, "Lifelong Machine Learning Systems: Beyond Learning Algorithms," AAAI Spring Symposium - Technical Report, Mar. 2013.

[10] R. Saxena, J. L. Shobe and B. L. McNaughton, "Learning in Deep Neural Networks and Brains With Similarity-Weighted Interleaved Learning," Proceedings of the National Academy of Sciences, vol. 119, pp. 1-11, 27 June. 2022.

[11] N. C. Abildgaard and T. L. Jacobsen, "Solving Complex Problems with Deep Multi-Level Skill Hierarchies," Aalborg University, 2021.

[12] A. Levy , G. Konidaris, R. Platt and K. Saenko, "Learning Multi-Level Hierarchies with Hindsight," in ICLR, 2019.

[13] F. Davesne and C. Barret, "Reactive Navigation of a Mobile Robot Using a Hierarchical Set of Learning Agents," in Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No.99CH36289), 1999.

[14] S. Huang, H. Su, J. Zhu and T. Chen, "Combo-Action: Training Agent for FPS Game with Auxiliary Tasks," in Proceedings of the AAAI Conference on Artificial Intelligence, 2019.

[15] J. Togelius and G. N. Yannakakis, "General General Game AI," in 2016 IEEE Conference on Computational Intelligence and Games (CIG), Santorini, Greece , 2016.

[16] S. Song, J. Weng, H. Su, D. Yan, H. Zou and J. Zhu, "Playing FPS Games With Environment-Aware Hierarchical Reinforcement Learning," in Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, (IJCAI-19), 2019.

[17] F. Zhang , D. Tang, Y. Dai, C. Zhou, S. Wu and S. Shi, "SkillNet-NLU: A Sparsely Activated Model for General-Purpose Natural Language Understanding," arXiv, 7 Mar. 2022.

[18] C. Tessler, S. Givony , T. Zahavy, D. J. Mankowitz and S. Mannor, "A Deep Hierarchical Approach to Lifelong Learning in Minecraft," in AAAI, 2017.

[19] Y. Wu and Y. Tian, "Training Agent for First-Person Shooter Game with Actor-Critic Curriculum Learning," in International Conference on Learning Representations, 2017.

[20] H. Rho, Y. Yu and K. Lee, "Learning to Box: Reinforcement Learning using Heuristic Three-step Curriculum Learning," in 2022 22nd International Conference on Control, Automation and Systems (ICCAS), 2022.

[21] I. Kodaka and F. Saitoh, "A Study on Application of Curriculum Learning in Deep Reinforcement Learning : Action Acquisition in Shooting Game AI as Example," in 2021 IEEE 12th International Workshop on Computational Intelligence and Applications (IWCIA), Hiroshima, Japan, 2021.

[22] D. Zhang, W. Bao, W. Liang, G. Wu and J. Cao, "A Curriculum Learning Based Multi-agent Reinforcement Learning Method for Realtime Strategy Game," in 2022 8th International Conference on Big Data and Information Analytics (BigDIA), 2022.

[23] "A2C," [Online]. Available: https://paperswithcode.com/method/a2c.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, "Proximal Policy Optimization Algorithms," arXiv, 2017.

[25] A. Juliani, A. Khalifa, V.-P. Berges, J. Harper , E. Teng , H. Henry , A. Crespi , J. Togelius and D. Lange, "Obstacle Tower: A Generalization Challenge in Vision, Control, and Planning," in IJCAI, 2019.