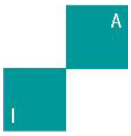
Inteligencia Artificial 20(59) (2020), 123-127
doi: 10.4114/intartf.vol20iss59pp123-127



INTELIGENCIA ARTIFICIAL

http://journal.iberamia.org/

Evaluating the impact of curriculum learning on the training process for an intelligent agent in a video game

Abstract We want to measure the impact of the curriculum learning technique on a reinforcement training setup, several experiments were designed with different training curriculums adapted for the video game chosen as a case study. Then all were executed on a selected game simulation platform, using two reinforcement learning algorithms, and using the mean cumulative reward as a performance measure. Results suggest that curriculum learning has a significant impact on the training process, increasing training times in some cases, and decreasing them up to 40% percent in some other cases.

Resumen Se desea medir el impacto de la técnica de aprendizaje por currículos sobre el tiempo de entrenamiento de un agente inteligente que está aprendiendo a jugar un video juego usando aprendizaje por refuerzo, para esto se diseñaron varios experimentos con diferentes currículos adaptados para el video juego seleccionado como caso de estudio, y se ejecutaron en una plataforma de simulación de juegos seleccionada, usando dos algoritmos de aprendizaje por refuerzo, y midiendo su desempeño usando la recompensa media acumulada. Los resultados sugieren que usar aprendizaje por currículos tiene un impacto significativo sobre el proceso de entrenamiento, en algunos casos alargando los tiempos de entrenamiento, y en otros casos disminuyéndolos en hasta en un 40% por ciento.

Keywords: Curriculum Learning, Reinforcement Learning, Training Curriculum, Mean Cumulative Reward, Proximal Policy Optimization, Video Games, Game AI, Unity Machine Learning Agents, Unity ML-Agents Toolkit, Unity Engine.

Palabras clave: Aprendizaje por Currículos, Aprendizaje por Refuerzo, Currículo de Entrenamiento, Recompensa Media Acumulada, Optimización por Políticas Próximas, Videojuegos, Inteligencia Artificial en Videojuegos, Unity Machine Learning Agents, Unity ML-Agents Toolkit, Motor Unity.

1 Introduction

In this document we present the results of several experiments with curriculum learning applied to a game AI learning process to measure its effects on the learning time, specifically we trained an agent using a reinforcement learning algorithm to play a video game running on a game simulation platform, then we trained another agent under the same conditions but including a training curriculum, which is a set of rules that modify the learning environment at specific times to make it easier to master by the agent at the beginning, then we compared both results. Our initial hypothesis is that in some cases using a training curriculum would allow the agent to learn faster, reducing the training time required. We describe in detail all the main elements of our work, including the choice of the game simulation platform to run the training experiments, the review of the reinforcement learning algorithms used to train the agent, the description of the video game selected as case study, the parameters used to design the training curriculums, and the discussion of the results obtained.

1.1 Problem Statement

According to [6] Justesen et al. (2017) there are several challenges that are still open in the domain of artificial intelligence applied to video games, also known as Game AI, some of them are listed below:

ISSN: 1137-3601 (print), 1988-3064 (on-line)
©IBERAMIA and the authors

- General video game playing: This challenge refers to creating general intelligent agents that can play not only a single game, but an arbitrary amount of known and unknown games. According to [21] Legg et al. (2007) being able to solve a single problem does not make you intelligent, to learn general intelligent behavior you need to train on not just a single task, but in many different tasks. [22] Schaul et al. (2011) suggest that video games are ideal environments for Artificial General Intelligence (AGI), in part because there are multiple video games that share common interface and reward conventions.
- Computational resources: Usually training an agent using deep neural networks for learning how to play an open-world game like Grand Theft Auto V [23] Rockstar Games (2020) requires huge amounts of computational power. This issue becomes even more noticeable if we want to train several agents. According to [6] Justesen et al. (2017) it is not yet feasible to train deep networks in real-time to allow agents to adapt instantly to changes in the game or adapt it to a particular playing style, which could be useful in the design of new types of games.
- Dealing with extremely large decision spaces: For well-known board games such as Chess the average branching factor is around 30, but for video games like Grand Theft Auto V [23] Rockstar Games (2020) or StarCraft the branching factor is several orders of magnitudes larger. How can we scale deep reinforcement learning to handle such levels of complexity is an important open challenge.

These challenges are problems that have not a straightforward solution, rather requiring a combination of multiple techniques to mitigate their impact, we believe curriculum learning could be one of those techniques since it could allow the agents to learn in less time, which in turn could mean less computational resources required. In the case of general video game playing, we believe curriculum learning could help to overcome this challenge is most scenarios, if agents are trained on easier 2D games first, allowing the agents to learn how to explore levels and collect items that can provide help later to defeat potential enemies and overcome future obstacles, and then if they are be trained on harder and complex 3D games, using all the experience gained when learning simpler games. Setting the training process this way could make the agent learn faster than a training process in which games are provided to the agent in a random order of complexity. Using less time for training provides additional side benefits including less money required for cloud servers to run the training, less energy consumption, less carbon footprint, and in case of commercial video games, faster product release times.

1.2 Motivation

Training intelligent agents to play video games have several applications in the real world since a video game can be seen as a low-cost and low-risk playground for learning complex tasks. In most cases, direct agent interaction with the real world is either expensive or not feasible, since the real world is far too complex for the agent to perceive and understand, so it makes sense to simulate the interaction in a virtual learning environment which receives input and returns feedback on a decision made by the agent, then most of the knowledge about the environment can be transferred to a physical agent, usually a robot, this approach is called Sim-to-Real (Simulation to Real World). In this scenario, having faster learning times when training agents would benefit several real-world applications.

2 Background

2.1 Reinforcement Learning

Machine learning is traditionally divided into three different types of learning [18] Alpaydin. (2010): supervised learning, unsupervised learning, and reinforcement learning. While supervised learning, in which intelligent agents are trained by example, has shown impressive results in a variety of different domains, it requires a large amount of training data that often has to be curated by humans [6] Justesen et al. (2017), a condition that is not always meet on the video games domain, sometimes there is no training data available (e.g. playing an unknown game), or the available training data is insufficient and the process of collecting more data can be very labor-intensive and sometimes infeasible. In these cases, reinforcement learning methods are often applied [6] Justesen et al. (2017). Reinforcement Learning (RL) [25] Bertsekas et al. (1996) [19] Sutton et al. (2018), is an attempt of formalizing the idea of learning based on rewards and penalties [20] Wolfshaar. (2017), in which an agent interacts with an environment and its goal is to learn a behavior policy through this interaction to maximize future rewards. How the environment reacts to a certain action is defined by a model that usually does not know, the agent can be in one of a set of states ($s \in S$) and can take one of many actions ($a \in A$) to change from one state to another. The decision of which state is chosen is decided by transition probabilities between states (P), once an action is taken the environments return a reward ($r \in R$) as feedback. The model defines the reward function and the transition probabilities [32] Weng (2018).

2.2 Perception-Action-Learning

A video game can easily be modeled as an environment in an RL setting, wherein agents have a finite set of actions that can be taken at each step and their sequence of moves determines their success [6] Justesen et al. (2017). At any given moment the agent is in a certain state, from that state it can take one of a set of actions. The value of a given state refers to how ultimately rewarding it is to be in that state. Taking an action in a state can bring an agent to a new state, provide a reward, or both, this is called the perception-action-learning loop [27] Arulkumaran et al. (2017). At time t, the agent receives state *st* from the environment. The agent uses its policy to choose an action at. Once the action is executed, the environment transitions a step, providing the next state *st+1* as well as feedback in the form of a reward *rt+1*. The agent uses knowledge of state transitions, of the form *(st, at, st+1, rt+1)*, in order to learn and improve its policy [27] Arulkumaran et al. (2017). The total cumulative reward is what all RL agents try to maximize over time. Depending on the game mechanic, reward signals can be sent frequently, for instance, every time an agent performs actions like killing an enemy, reaching a checkpoint or getting a health box, or it can be very sparse if the agent is rewarded only until finding something valuable inside an extensive terrain or a complex maze.

2.3 Curriculum Learning

According to [1] Bengio et al. (2009) humans learn better when concepts and examples are taught in a meaningful order, using previously learned concepts to ease the learning of more abstract, complex concepts. Our current educational model heavily relies on this to increase the speed at which learning can occur, by using curriculums that use the "starting small" strategy. [1] Bengio et al. (2009) state that choosing which examples to present and defining a specific order for showing them, based on criteria of incremental difficulty, can increase the learning speed. An easy way to demonstrate this strategy is thinking about the way arithmetic, algebra, and calculus is taught in a typical education system. Arithmetic is taught before algebra. Likewise, algebra is taught before calculus. The skills and knowledge learned in the earlier subjects provide scaffolding for later lessons. The same principle can be applied to machine learning, where training on easier tasks can provide scaffolding for harder tasks in the future according to [4] Juliani (2017). [2] Elman et al. (1993) state that the "starting small" strategy could make possible for humans to learn what might otherwise prove to be unlearnable, however, according to [3] Harris (1991), some problems can best be learned if the whole data set is available for the neural network from the beginning, otherwise they often fail to learn the correct generalization and remain stuck in a local minimum. [1] Bengio et al. (2009) formalize the use of curriculums as training strategy in the context of machine learning by calling it Curriculum Learning, stating that this strategy can have two effects: in convex criteria, problems can increase the speed of convergence of the training process to a minimum, in case of non-convex criteria can increase the quality of the local minima obtained. [1] Bengio et al. (2009) hypothesize that a well-chosen curriculum strategy can act as a continuation method, which is a general strategy for global optimization of nonconvex functions, especially useful on deep learning methods that attempt to learn feature hierarchies, where higher levels are formed by the composition of lower-level features, training these deep architectures involve potentially intractable non-convex optimization problems. [5] Allgower (2003) indicates how continuation methods address complex optimization problems by smoothing the original function, turning into a different problem that is easier to optimize. By gradually reducing the amount of smoothing, it is possible to consider a sequence of optimization problems that converge to the optimization problem of interest. [28] Narvekar et al. (2020) present Quick Chess as an example of how a curriculum can be used to teach how to play a game using the "starting small" strategy. Quick Chess is a game designed to introduce children to the full game of chess, by using a sequence of progressively more difficult subgames. The first subgame is played on a 5x5 board with pawns only, which is useful for children to learn how pawns move, get promoted, and take other pieces. In the second subgame the King piece is added, which introduces a new objective: Keep the king alive. In each successive subgame, new elements are introduced (such as new pieces, a larger board, or different configurations) that require learning new skills and building upon the knowledge learned in previous subgames, the final subgame is the full game of chess [28] Narvekar et al. (2020).

3 Game Simulation Platform

3.1 Review of Game Simulation Platforms

The first step of our work was to choose an adequate game simulation platform that allows us to run all the game AI experiments to test our ideas. The chosen platform needs to support experiments with reinforcement learning, should be highly configurable, easy to use and it should not be constrained to a specific type of video game. According to [6] Justesen et al. (2017) increasing use of deep learning methods in video games is due to the practice of comparing game-playing algorithm results on public available game simulation platforms, in which algorithms

are ranked on their ability to score points or win in games. In a typical deep reinforcement learning model running on a game simulation platform, input is taken from the game environment and meaningful features are extracted automatically, the RL agent produces actions based on these features, making the environment transition to the next state [26] Shao et al. (2019). Several game simulation platforms listed by [16] Juliani et al. (2020), [26] Shao et al. (2019) and [6] Justesen et al. (2017) are described below:

- Arcade Learning Environment (ALE) [7] Bellemare et al. (2013) is a free, open-source software framework for interfacing with hundreds of games for Atari 2600, a second-generation home video game console originally released in 1977 and sold for over a decade [8] Montfort et al. (2009). ALE allows domain-independent AI algorithms evaluation providing an interface to game environments that present research challenges for reinforcement learning, model learning, model-based planning, imitation learning, transfer learning, and intrinsic motivation.
- Retro Learning Environment (RLE) [9] Bhonker et al. (2017) is an open-source software environment that allows intelligent agents to be trained to play games for Super Nintendo Entertainment System (SNES), Nintendo Ultra 64 (N64), Game Boy, Sega Genesis, Sega Saturn, Dreamcast, and PlayStation. RLE supports working with multi-agent reinforcement learning (MARL) [10] Buşoniu et al. (2010) tasks, particularly useful to train and evaluate agents that compete against each other, rather than against a pre-configured in-game AI.
- ViZDoom [11] Kempka et al. (2016) is an AI research platform for visual reinforcement learning based on the classical first-person shooter (FPS) video game Doom, a semi-realistic 3D world that can be observed from a first-person perspective. VizDoom allows developing agents to play Doom using the screen buffer as input, agents that have to perceive, interpret and learn the 3D world in order to make tactical and strategic decisions such as where to go and how to act.
- DeepMind Lab [12] Beattie et al (2016) is a first-person 3D game platform designed for research and development of general artificial intelligence and machine learning systems, which can be used to study how pixels-to-actions autonomous intelligent agents do learn complex tasks in large, partially observed and visually diverse worlds.
- Project Malmo [13] Johnson et al. (2016) is an AI experimentation platform built on top of the popular video game Minecraft, a platform designed to support fundamental research in artificial general intelligence (AGI) and related areas including robotics, computer vision, reinforcement learning, planning, and multi-agent systems, by providing a rich, structured and dynamic 3D environment with complex dynamics.
- TorchCraft [14] Synnaeve et al. (2016) is a library that enables deep learning research on real-time strategy games such as StarCraft: Brood War, a popular real-time strategy (RTS) video game published in 1998 by Blizzard Entertainment. RTS games have been a domain of interest for the planning and decision-making research communities [15] Silva et al. (2017), since they aim to simulate the control of multiple units in a military setting at different scales and levels of complexity, normally in a fixed-size 2D map.
- Unity [16] Juliani et al. (2020) is a 3D real-time cross-platform video game development platform, featuring high-quality rendering and physics simulation, created by Unity Technologies. Unity is not restricted to any specific genre of gameplay or simulation, this flexibility enables the creation of tasks ranging from simple 2D grid world problems to complex 3D strategy games, physics-based puzzles, or multi-agent competitive games possible. Unity provides an open-source framework called Unity Machine Learning Agents Toolkit (Unity ML-Agents Toolkit) [17] Juliani (2017), [16] Juliani et al. (2020), [29] Juliani et al. (2020), a game AI framework that enables 2D, 3D, VR/AR games and simulations to serve as learning environments for training intelligent agents. Agents can be trained using reinforcement learning, curriculum learning, imitation learning, neuroevolution, and other machine learning methods [24] Mattar et al. (2020).

3.2 Game Platform Taxonomy

After listing the game simulation platforms, we needed a way to support the selection of one of them to run our experiments, so we decided to rely on a taxonomy proposed by [18] Juliani et al. (2020) that classifies game simulation platforms based on its flexibility to run different game environments. Figure 1 show several platforms classified using this taxonomy.

Single Env	Env Suite	Domain-Specific Platform	General Platform
Cart Pole	ALE	MuJoCo	Unity & ML-Agents
Mountain Car	DMLab-30	DeepMind Lab	
Obstacle Tower	Hard Eight	Project Malmo	
Pitfall!	AI2Thor	VizDoom	
CoinRun	OpenAI Retro	GVGAI	
Ant	DMControl	PyBullet	
	ProcGen		

Figure 1. Taxonomy of game platforms based on their flexibility [16] Juliani et al. (2020).

Each taxonomy category is described below:

- Single Environment: Platforms that act as a black box from an agent's perspective, and usually run only a specific game.
- Environment Suite: Consist of several learning environments packaged together to benchmark the performance of a game AI technique along with different games.
- Domain-Specific Platform: Allows the creation of a set of tasks within a specific domain such as first-person navigation, car racing, or human locomotion.
- General Platform: Platforms that allow creating custom learning environments with arbitrarily complex visuals, physical and social interactions, and configurable tasks.

According to [16] Juliani et al. (2020) Unity is the only platform that has the complexity, flexibility, and computational properties expected from a general game simulation platform for game AI. This platform is the only one that provides out-of-the-box support for both reinforcement learning and curriculum learning, and it is not constrained to a specific game or learning environment. Taking in consideration all these reasons, we decided to choose Unity and its Unity ML-Agents Toolkit [16] Juliani et al. (2020) as the simulation platform to run our experiments.

4 Reinforcement Learning Algorithms

The Unity ML-Agents Toolkit [16] Juliani et al. (2020) provides an implementation based on TensorFlow of two state-of-the-art reinforcement learning algorithms that we found appropriate to train our agents: Proximal Policy Optimization (PPO) [30] Schulman et al. (2017) and Soft Actor-Critic (SAC) [31] Haarnoja et al. (2018). PPO is an on-policy algorithm that has been shown to be more general-purpose and stable than many other reinforcement learning algorithms. On the other hand, SAC is off-policy, which means it can learn from any past experiences, which are collected and placed in an experience replay buffer to be randomly drawn during training. Proximal Policy Optimization (PPO) aims to take the biggest possible improvement step on a policy to improve performance, without stepping so far that could cause a performance collapse. Proximal Policy Optimization (PPO) trains a stochastic policy in an on-policy way, which means that it explores by sampling actions according to the latest version of its stochastic policy. The amount of randomness in action selection depends on both initial conditions and the training procedure. Over the course of training the policy typically becomes progressively less random, as the update rule encourages it to explore rewards that it has already found. Soft Actor-Critic (SAC) optimizes a stochastic policy in an off-policy way, using entropy regularization. The policy is trained to maximize a trade-off between expected return and entropy, a measure of randomness in the policy, which has a close connection to the exploration-exploitation trade-off: increasing entropy results in more exploration, which can accelerate learning later on, but it can also prevent the policy from prematurely converging to a bad local optimum.

5 Case Study: Toy Soccer Game

5.1 Learning Environment

The next step in our work is the selection of a video game to run our experiments. The Unity ML-Agents Toolkit [40] Juliani et al. (2020) includes a set of 15 preconfigured 3D game environments specially designed to serve as challenges for game AI researchers, and test novel machine learning techniques, especially reinforcement learning ones. After inspecting all the learning environments, we selected one that fulfilled all our requirements: Soccer Twos. This environment is a toy soccer game that has 2 agents of opposite teams, blue and red, that compete against each other in a 30x15 meters rectangular field called pitch, which has one goal at each end. The objective of each agent is to push a ball inside the opponent's goal to win the soccer match. The pitch is surrounded by a wall to prevent the ball from leaving the field. At the beginning of the match, the ball starts at the center of the pitch, and both agents are placed 2 meters away from the ball, then the agents can move at a maximum speed of 2 meters per second, and they can push the ball in any direction but they cannot throw it into the air. Since the environment is a simplification of the real soccer game, there are elements that are not present, for example, there are no other players, including the goalkeeper, there are no team manager and no referee. Also, there are soccer rules that do not apply, including getting a yellow or red card for fouls done by the players, or special ways of restarting the game like penalty kick or corner kick. We constrained the environment to 15.000 iterations, also called frames or simulation steps, if no agent is able to score a goal before the iteration limit is reached, a draw is declared and the match finishes. A modern six-core CPU usually executes between 60 to 120 frames per second, this means a match could last from 125 to 250 seconds if no agent scores a goal. The scenario includes a trained reinforcement learning model that can

be used by both agents, in our case we used this model on the red agent only. All our training experiments were run only on the blue agent, serving the red agent only as the opponent to defeat. We configured the environment to be able to run up to 10 soccer matches simultaneously, in order to run our experiments faster, taking advantage of the Simultaneous Single-Agent scenario provided by the Unity ML-Agents Toolkit [40] Juliani et al. (2020). A screenshot of Soccer Twos environment is shown in Figure 2.

Figure 2. Screenshot of learning environment Soccer Twos [29] Juliani et al. (2020).

5.2 Observation Space

Each agent, represented in the environment as a cuboid with little decorations resembling eyes and a mouth, can observe its surroundings using collision rays that end in a collision detection sphere, each ray is shot in a particular direction to detect the distance to other objects. The agent uses 3 rays shot backward separated by 45 degrees, and 11 rays shot forward distributed across 120 degrees. Each ray can detect 6 types of objects: The soccer ball, the walls enclosing the soccer field, agents from the same team (blue color), agents from the opposite team (red color), its own goal (blue color), the opponent's goal (red color). By each type of object, the x, y, and z coordinates plus the distance to the object are detected. In total, the agent has an Observation Space vector of 336 variables per iteration: `14 rays * 6 types of objects * [ (x, y, z) + distance to object ] = 336 variables` Since the game can run from 60 to 120 frames per second, each variable is being updated 60 to 120 times per second. This is the way the agent can perceive its surroundings.

5.3 Action Space

The Action Space vector is a vector composed of 3 branched actions the agent can use to explore and interact with the environment:

- Frontal motion: If this value is greater than 0 the agent moves forward, if the value is less than zero it moves backward, and if the value is equal to zero no frontal motion is done.
- Lateral motion: If this value is greater than 0 the agent moves to its right, if the value is less than zero it moves to its left, and if the value is equal to zero no lateral motion is done.
- Rotation: If this value is greater than 0 the agent rotates around its Y-axis clockwise, if the value is less than zero it rotates around its Y-axis counterclockwise, if the value is equal to zero no rotation is done.

The maximum lateral or frontal speed is constrained to 2 meters per second.

5.4 Reward Signals

Reward signals are the rules to reward and punish the agent for its actions to guide learning, they are the way to measure its performance in the game. The maximum reward value in a match is 1, and the maximum punishment value is -1. The reward rules applied to our agent are:

- *-1*: When the opponent agent scores a goal.
- *(1 - accumulated_time_penalty)*: When the blue agents score a goal.
- No reward or punishment value is given to either agent if the match time runs out

The value *accumulated_time_penalty* is incremented by *(1/MaxSteps)* every iteration and is reset to 0 at the beginning of a new match. This penalty value is set this way to encourage our agent to score goals faster. *MaxSteps* is equal to 15.000 simulation steps. The main method of measuring the agent's performance over multiple soccer matches is the mean cumulative reward, which can tell us on average if the agent is scoring goals or not. A mean value close to 1 means that our agent is winning almost every single match, outperforming its opponent, close to -1 means is almost losing every single time, close to 0 means that in most cases the matches are ending in a draw, no agent is scoring goals which mean that both have a low game performance. If the mean value is close to 0.5 it means that both agents are being able to score goals at similar rates. We do not have information about the trained model that the red agent is using, we do not know which reinforcement learning technique was used, or which learning environment configuration and hyperparameters were set, but it is safe to assume that it was trained using one of the state-of-the-art reinforcement learning algorithms included in the toolkit, using an optimal set of hyperparameters, since the learning environment chosen was designed to serve as a challenge to test new ML algorithms and techniques. We wanted how the mean cumulative reward behaves if we use the same trained model on both blue and red agents and make them compete against to each other, it turned out that the mean cumulative reward stays close to 0.5 over time, meaning both agents perform well and score goals at similar rates, each one having a 50% chance of winning the game. In this context it is safe to assume that if one of our experiments ends up with a mean cumulative reward value tending to 0.5, the training was successful, and the agent is performing at least as well as the red agent which is using the trained model. Since the blue agent has an opponent that is already trained to play in an optimal way, it is expected that at the beginning of the experiments the mean cumulative reward value starts at zero, meaning the blue agent is being outperformed by the red agent. What we wanted to evaluate in this work is if using the curriculum learning technique on the training process the mean cumulative reward value gets closer to 0.5 faster than without a curriculum, in other words, if this technique makes the agent to learn faster.

5.5 Environment Parameters

To run experiments with curriculum learning using the Unity ML-Agents Toolkit [16] Juliani et al. (2020), additionally to trainer_config.yaml configuration file, we must provide a curriculum.yaml file that contains the parameters we want to vary over time to control how difficult is for our agent to learn how to play, the idea is to vary those parameters in a way that allows the agent to learn faster. These variations are discrete and are divided into lessons. The difficulty our agent has when learning at the beginning of the training is linked directly to the performance of the opponent agent, the easier the opponent is, the easier is for our agent to learn, so it makes sense to design training curriculums that alter the opponent's performance to give our agent an initial advantage over it.
We adapted the case study to support the variation of 2 environment parameters that alter the opponent's behavior, and one that alters the reward signals:

- *opponent_exist*: At the beginning of the training the blue agent needs to learn how to move inside the pitch and how to rotate and move towards the ball, but if the red agent is present, this learning process is interrupted, because the red agent will score a goal in a very short time, almost immediately, since it is already trained on how to do it efficiently. In this context, we want to be able to remove the opponent from the field while our agent is mastering how to move efficiently towards the ball and to push it towards the opponent's goal. After the agent learned how to play alone in the field, we want to restore the opponent back, so our agent can start learning how to defeat it.
- *opponent_speed*: We believe modifying the movement speed of the red agent will give the blue agent a clear advantage over its opponent, the slower the red agent is, more opportunities the blue agent will have to learn how to score goals since it will be able to move faster towards the ball than the red agent. As soon as our agent learns how to defeat a slow version of its opponent, we want to increase the opponent's speed so it can become a harder player to overcome.
- *ball_touch_reward*: We want to give an additional reward to our agent every single time it touches the ball, because we wonder if this will incentivize our agent to move toward the ball faster, but this approach has an important risk to consider: This can teach the agent how to seek the ball very quickly but not necessarily how to score goals. This option has an additional problem: The mean cumulative reward values will be altered by the additional reward, this will invalidate any possible performance comparison we want to make against another agent's cumulative reward values.

In the case of parameter *opponent_exist* a value of 1 means the opponent does exist, and a value of 0 means the opponent does not exist. For *opponent_speed* parameter, a value of 0 means the opponent cannot move. The maximum movement speed of all agents is 2 meters per second, so a greater value for the *opponent_speed* parameter will be rounded down to 2. For each curriculum, we want to experiment with, a curriculum.yaml file must be

provided, containing the definition of which of those parameters are going to vary over time, and the times when these variations occur, times called lesson thresholds.

6 Experimental Results

6.1 Experimental Setup

We used Google Cloud Platform to execute all or experiments, specifically we used Google Compute Engine (GCE), an Infrastructure as a Service (IaaS) solution used by Google's applications including Gmail and YouTube. GCE enables its users to launch Virtual Machines (VMs) on demand, that can be accessed using Secure Shell (SSH). We activated 2 virtual machines running Fedora 10, a popular Linux distribution. Each virtual machine had a virtual 8-core Intel CPU, 16 GB of RAM and 10 GB of Hard Drive. We created a bash file called SetupVirtualMachine.sh containing a set of terminal commands to setup the machines to run the experiments, setup that includes the installation of all required third-party libraries, installation of Unity and the ML-Agents Toolkit, activation of the Unity License and cloning of the repository containing the learning environment. After running all these commands, the virtual machine is ready to run our experiments. File can be found here: https://github.com/rsaenzi/master-thesis.

6.2 Control Experiments

Each experiment for this work consisted of 100 million soccer matches, in the first two experiments we ran, we used Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) algorithms without setting a curriculum, so these two can be used as control experiments. Scatter plots showing the mean cumulative reward over the 100 million matches for each algorithm are shown in Figure 3, a gaussian filter was applied to reduce noise, we drew a green dotted line over the 0.5 reward value to help visualize where the cumulative reward should be after running all the soccer matches so the training can be considered successful.

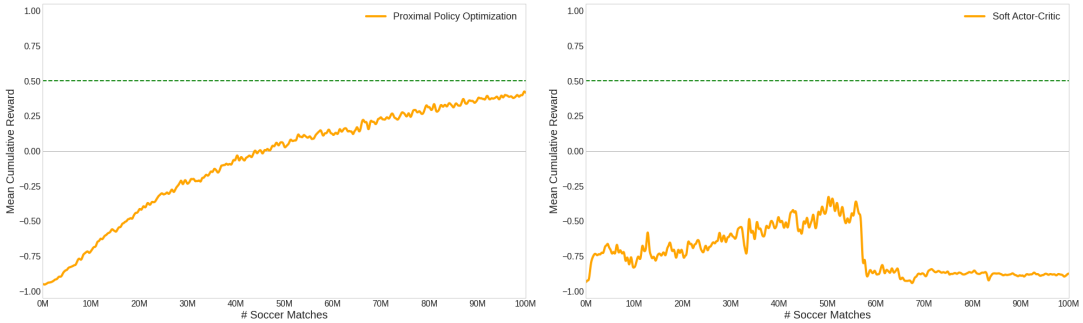


Figure 3. Mean cumulative reward for Proximal Policy Optimization and Soft Actor-Critic.

In the experiment with Proximal Policy Optimization (PPO), we got a result similar to our initial hypothesis about how the mean cumulative reward values should vary over time in a successful training session. On the other hand, with Soft Actor-Critic (SAC) we got results that indicate a poor agent performance because the values tend to -1.

6.3 Preliminary Experiments

We wanted to try different combinations of environment parameters and thresholds, initially, we designed 2 curriculums named A and B, the first one using the *opponent_exist* and *opponent_speed* parameters, the second using only *ball_touch_reward*. With curriculum A we wanted to test if our agents learn faster by removing the opponent during the first 10% of the 100 million soccer matches, after that the opponent would be present but with limited movement speed, first having no speed at all, then incrementing its speed linearly in increments of 0.25 meters per second every 10% of the total matches until reaching its full speed at 90% of the training.

```
# Curriculum A
thresholds: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
parameters:
  opponent_speed: [0.0, 0.0, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0]
  opponent_exist: [0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

In figure 4 the mean cumulative reward results for curriculum A are plotted in blue, alongside with the PPO and SAC control experiments for comparison, the vertical dotted lines in blue represent the curriculum thresholds, where one or more environment parameters were changed, the dotted line is wider where the curriculum has no effect anymore, in this case after 90% of the matches.

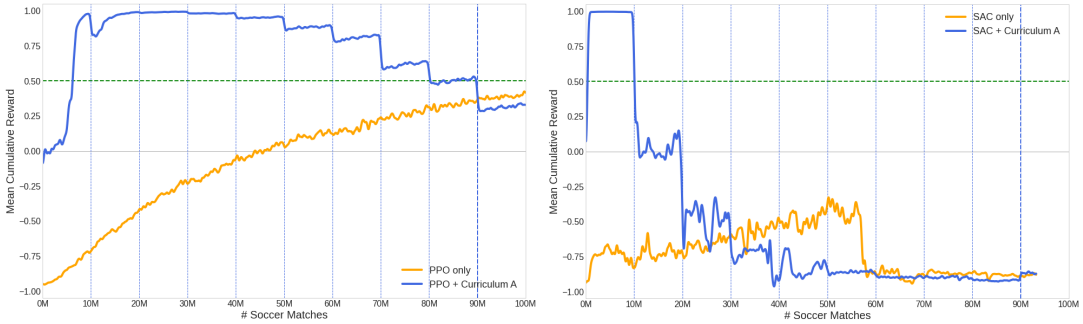


Figure 4. Mean cumulative reward for PPO control experiment vs PPO and Curriculum A.

In the scatter plot for Proximal Policy Optimization only vs Proximal Policy Optimization and Curriculum A we can notice that our agent performs well while the opponent is not present or its movement speed is reduced, but starting from 50% of the matches we can notice performance drops every time the opponent increments its movement speed, and after 90% the agents is slightly over-performed by the control case. When we train the agent using PPO only, we get a mean cumulative reward around 0.45 at the end of the training, but using a curriculum we get around 0.35, this means the curriculum is hurting the performance. In the case of Soft Actor-Critic only vs Soft Actor-Critic and Curriculum A is evident that the curriculum has a positive impact only on the first 10% of the training, this means the agent learns very fast how to score goals if the opponent is not present, but after 20% the agent performs the same and even worse than the control case. In the end, from 90% of the training, there is no noticeable difference in the agent's performance when using the curriculum vs the control experiment, both perform very badly. For curriculum B, additional to the default reward signal established previously, we wanted to give our agent a bonus reward for touching the ball, a reward that is reduced over time. In the first 10% of the 100 million soccer matches, we give a reward of 0.3, then we reduce the reward in 0.1 every 10% of the total matches until having no additional reward at 30% of the training. Figure 5 show the mean cumulative reward results for curriculum B compared to the control experiments.

```
# Curriculum B
thresholds: [0.1, 0.2, 0.3]
parameters:
  ball_touch_reward: [0.3, 0.2, 0.1, 0.0]
```

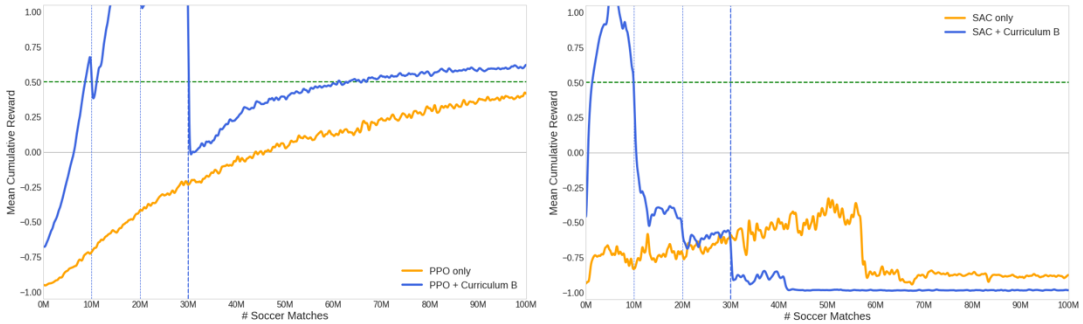


Figure 5. Mean cumulative reward for PPO control experiment vs PPO and Curriculum B.

In the experiment using curriculum B with Soft Actor-Critic, we get similar results using the curriculum A: a peak of good performance in the first 10% of the soccer matches, but then we get a bad performance overall. On the other hand, for Proximal Policy Optimization we get better performance all the time, not only when curriculum B is active, but even after 30% of the training when it is not. The blue line crosses the 0.5 cumulative reward limit around 60% of training, totally outperforming the agent that was trained in the control experiment, its line does not cross the 0.5 limit, however, it gets very close to it. The importance of this experiment is that, in some cases like this one, we can

get the same or even training results in fewer soccer matches, only 60 million matches are required to have a good performance if we use curriculum B, but around 120 million will be required to get the same results using PPO alone. Is important to emphasize that the cumulative reward values before the 30% of the training are being altered by the additional reward we are giving to the agent when touching the ball, so in the range of 0% to 30% the cumulative reward cannot be considered a valid benchmark for the agent's performance, so it cannot be used for comparison. At this point we decided not to continue using SAC. After running the preliminary experiments, we realized that each experiment could take from 10 to 30 days to finish, so is not feasible, at least for us, to test all possible combinations of thresholds and values for 1, 2 and 3 environment parameters, the costs are prohibitive. What we did instead was experimenting with a small set of relevant curriculums, trying to infer any useful information from their results. Since we got a good performance in one of the experiments that have 3 lessons, we decided to run 20 more experiments, not only with 3 but also with 4 lessons, each one having different combinations of environment parameters and thresholds, some of them have 5, 6, 1, and 9 lessons. All of them are compared against the PPO control experiment. We assigned a letter to each experiment in a random order, so it has no special meaning, the order we use to present the experiment results is not relevant either.

7 Discussion

After completing all experiments, we proceeded to group all curriculums according to its performance when compared against the Proximal Policy Optimization (PPO) control experiment, so we can analyze them as a cluster in order to identify common patterns:

7.1 Curriculums with Lower Performance

The first group contains all curriculums that underperformed the control experiment, in this group we have curriculums A, A+B, D, E, F, G, W, Z. Figure 6 shows the mean cumulative reward results for all these curriculums.

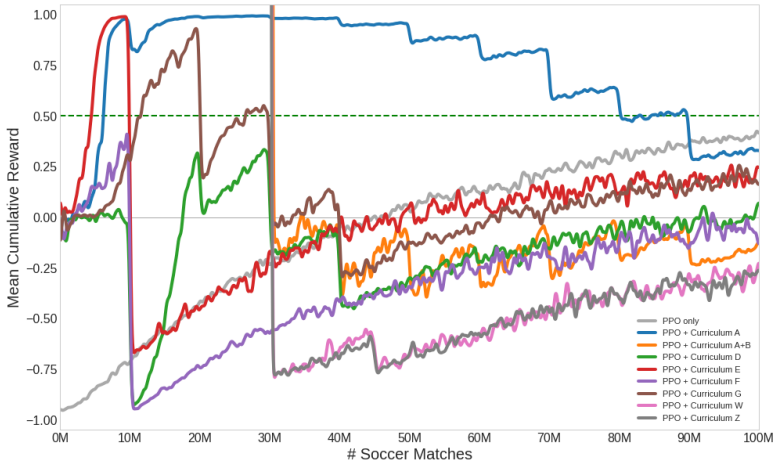


Figure 6. Mean cumulative reward for PPO control experiment vs all curriculum experiments that had lower performance than the control experiment at the end of 100 million soccer matches.

We noticed a few patterns when we analyzed these curriculums:

- Curriculums F and E belong to this group, both have a single threshold at 10% of the training at which the opponent's movement speed is incremented from 0 to 2 meters per second, it seems that this sudden change in the velocity does not provide the agent enough time to learn how to play.
- Curriculums D and G have 4 thresholds, at 10%, 20%, 30% and 40% of the training, in the latest 3 thresholds the opponent's movement speed is incremented linearly, adding 0.5 meters per second. This result suggests that a linear increment in velocity at the end of the training does not provide good performance results.
- Curriculums W and Z use both *ball_touch_reward* and *opponent_speed* environment parameters to encourage the agent to learn, but is not a good strategy because it seems the agent suffers from overfitting, meaning the agent learn how to move towards the ball quickly when the opponent is slow, but not necessarily learn how to score goals efficiently.

- Curriculums A and A+B have a big space between the thresholds, in both cases of 10%, it seems that this scenario is not good for the agent because it plays under the initial favorable environment conditions for too long, making it unable to adapt to new harder environment conditions.

7.2 Curriculums with Same Performance

The second group contains all curriculums that performed almost the same at the end of the training than the control experiment, in this group we have curriculums J, M, Q, V. Figure 7 shows the mean cumulative reward results for all these curriculums.

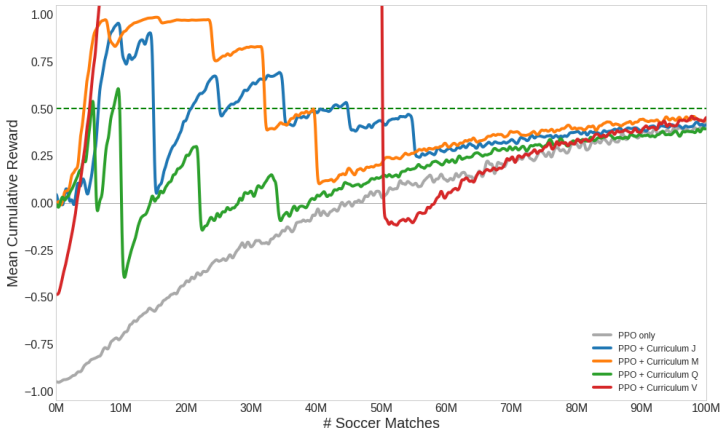


Figure 7. Mean cumulative reward for PPO control experiment vs all curriculum experiments that had the same performance as the control experiment at the end of 100 million soccer matches

In this group most of the curriculums do not have a consistent increase rate in the opponent's movement speed, the distribution of the thresholds does not follow a pattern. The results indicate that this is not a good strategy in terms of the agent's performance. In case of curriculum V, we give additional reward to our agent, starting with 0.5 at the beginning of the training, decreasing the value by 0.1 meters per second every 10% of the training. The bad performance of this curriculum suggests that the agent learns how to reach the ball quickly since the reward is big for this action, but then is unable to learn how to push it towards the opponent's goal.

7.3 Curriculums with Higher Performance

The third group contains all curriculums that performed better at the end of the training than the control experiment, in this group we have curriculums B, C, H, K, L, N, O, P, R, U, X, Y. Figure 8 shows the mean cumulative reward results for all these curriculums.

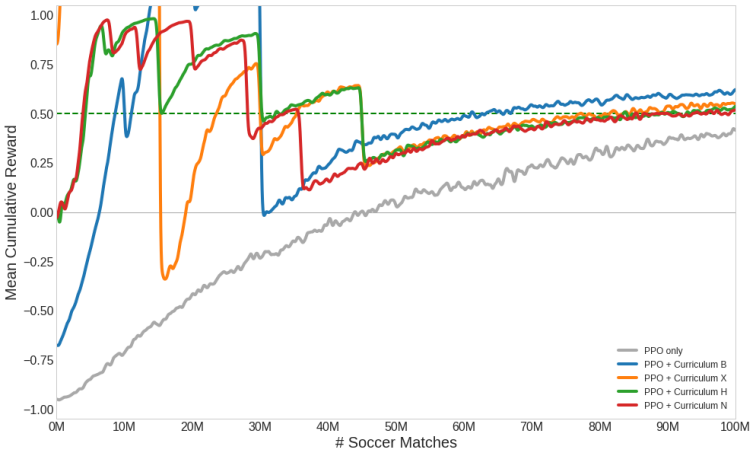


Figure 8. Mean cumulative reward for PPO control experiment vs all curriculum experiments that had higher performance than the control experiment at the end of 100 million soccer matches.

In this group we noticed a few interesting things:

- Curriculums B and U suggest that giving an additional reward to the agent can be beneficial, but only if the reward decreases rapidly during the training, to avoid the agent to learn to rely only on touching the ball to get a reward.
- Several curriculums in this group suggest that it is important for the agent to have a strong advantage over its opponent but only over a short period of time. In most curriculums the opponent is not present for the first 10% of training time, giving the agent the opportunity to learn how to move inside the pitch and how to move the soccer ball around, then the opponent's movement speed is incremented but not in a linear way, rather in a logarithmic way, this prevents the agent to overfit to the initial conditions.

Here we list the times at which the mean cumulative reward reaches the optimal value of 0.5:

- Curriculum B: At 60% of the training time
- Curriculum H: At 82% of the training time
- Curriculum K: At 95% of the training time
- Curriculum N: At 82% of the training time
- Curriculum O: At 90% of the training time
- Curriculum P: At 95% of the training time
- Curriculum U: At 90% of the training time
- Curriculum X: At 75% of the training time
- Curriculum Y: At 95% of the training time

We can notice that the best curriculums are B, X, H and N. Both curriculums H and N require around 18% less training time to get similar results when compared to the control experiment, curriculum X requires around 25% less time, and curriculum B requires around 40% less time. These values are actually higher since the control experiment does not reach the 0.5 value after 100 million experiments but extrapolating its values it is safe to say that the control experiment will reach the 0.5 limit around 120 million matches. Figure 9 shows the mean cumulative reward results only for curriculums B, X, H, N.

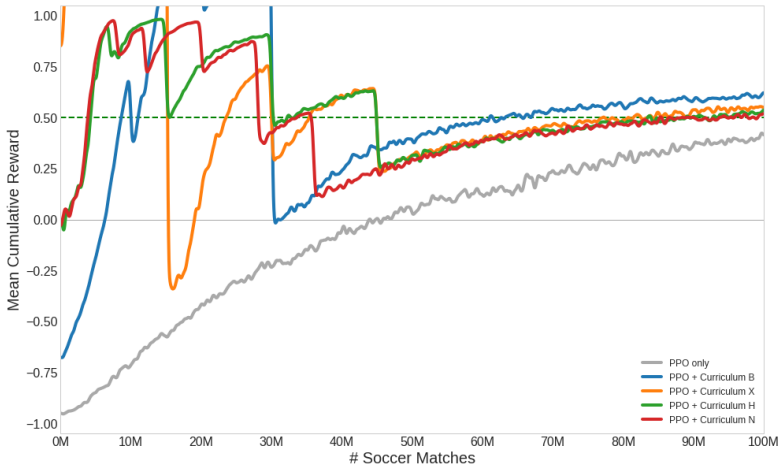


Figure 9. Mean cumulative reward for PPO control experiment vs the best 4 curriculum experiments compared over 100 million soccer matches.

8 Conclusions

Several experiments with curriculum learning were executed to measure its effects over the training process of an agent that is learning to play a video game using reinforcement learning. The initial hypothesis is that, in some cases, using a curriculum would allow the agent to learn faster. Our results indicate that using curriculum learning could have a significant impact on the learning process, in some cases helping the agent to learn faster, overperforming the control experiment, in other cases having a negative impact, making the agent to learn slower. In this work 24 curriculums were designed, each one having a different configuration of learning environment parameters and thresholds. We were able to infer several patterns from the training results that could indicate if a curriculum will improve or hurt the agent's performance, measured by the mean cumulative reward. In 12 experiments we get better performance than the control experiment, the best curriculum could save up to 40% of the training time required to reach an optimal performance level. In this work we used only two algorithms in our experiments, Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC), would be interesting to try more reinforcement learning

algorithms, or even supervised learning ones to test our main hypothesis. In the case of Soft Actor-Critic (SAC) a process of hyperparameter optimization is needed to see if it can work well with the case study we chose. We used a learning environment included in the Unity ML-Agents Toolkit [18] Juliani et al. (2020) as case study: Soccer Twos, a toy soccer video game, but would be interesting to test the curriculum learning technique on a wider variety of video games, including 2D and 3D, first-person and third-person games, of multiple genres such as strategy, adventure, role-playing and puzzle to see if this technique has a bigger impact on a particular genre.

References

[1] Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. Proceedings of the 26th International Conference On Machine Learning, ICML 2009, 41–48. https://dl.acm.org/doi/10.1145/1553374.1553380

[2] Elman, J. L. (1993). Learning and development in neural networks: The importance of starting small. Cognition, 48, 71–99. https://doi.org/10.1016/S0010-0277(02)00106-3

[3] Harris, C. (1991). Parallel distributed processing models and metaphors for language and development. Ph.D. dissertation, University of California, San Diego. https://elibrary.ru/item.asp?id=5839109

[4] Juliani, Arthur. (2017, December 8). Introducing ML-Agents Toolkit v0.2: Curriculum Learning, new environments, and more. https://blogs.unity3d.com/2017/12/08/introducing-ml-agents-v0-2-curriculum-learning-new-environments-and-more/

[5] Allgower, E. L., & Georg, K. (2003). Introduction to numerical continuation methods. In Classics in Applied Mathematics (Vol. 45). Colorado State University. https://doi.org/10.1137/1.9780898719154

[6] Justesen, N., Bontrager, P., Togelius, J., & Risi, S. (2017). Deep Learning for Video Game Playing. IEEE Transactions on Games, 12(1), 1–20. https://doi.org/10.1109/tg.2019.2896986

[7] Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. IJCAI International Joint Conference on Artificial Intelligence, 2013, 4148–4152. https://doi.org/10.1613/jair.3912

[8] Montfort, N., & Bogost, I. (2009). Racing the beam: The Atari video computer system. MIT Press, Cambridge Massachusetts. https://pdfs.semanticscholar.org/2e91/086740f228934e05c3de97f01bc58368d313.pdf

[9] Bhonker, N., Rozenberg, S., & Hubara, I. (2017). Playing SNES in the Retro Learning Environment. https://arxiv.org/pdf/1611.02205.pdf

[10] Buşoniu, L., Babuška, R., & De Schutter, B. (2010). Multi-agent reinforcement learning: An overview. Studies in Computational Intelligence, 310, 183–221. https://doi.org/10.1007/978-3-642-14435-6_7

[11] Kempka, M., Wydmuch, M., Runc, G., Toczek, J., & Jaskowski, W. (2016). ViZDoom: A Doom-based AI research platform for visual reinforcement learning. IEEE Conference on Computational Intelligence and Games, CIG, 0. https://doi.org/10.1109/CIG.2016.7860433

[12] Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., … Petersen, S. (2016). DeepMind Lab. https://arxiv.org/pdf/1612.03801.pdf

[13] Johnson, M., Hofmann, K., Hutton, T., & Bignell, D. (2016). The malmo platform for artificial intelligence experimentation. Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16), 2016-January, 4246–4247. http://stella.sourceforge.net/

[14] Synnaeve, G., Nardelli, N., Auvolat, A., Chintala, S., Lacroix, T., Lin, Z., Richoux, F., & Usunier, N. (2016). TorchCraft: a Library for Machine Learning Research on Real-Time Strategy Games. https://arxiv.org/pdf/1611.00625.pdf

[15] Silva, V. do N., & Chaimowicz, L. (2017). MOBA: a New Arena for Game AI. https://arxiv.org/pdf/1705.10443.pdf

[16] Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., & Lange, D. (2020). Unity: A General Platform for Intelligent Agents. https://arxiv.org/pdf/1809.02627.pdf

[17] Juliani, A. (2017). Introducing: Unity Machine Learning Agents Toolkit. https://blogs.unity3d.com/2017/09/19/introducing-unity-machine-learning-agents/

[18] Alpaydin, E. (2010). Introduction to Machine Learning. In Massachusetts Institute of Technology (Second Edition). The MIT Press. https://kkpatel7.files.wordpress.com/2015/04/alppaydin_machinelearning_2010

[19] Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (Second Edition). The MIT Press. http://incompleteideas.net/sutton/book/RLbook2018.pdf

[20] Wolfshaar, J. Van De. (2017). Deep Reinforcement Learning of Video Games [University of Groningen, The Netherlands]. http://fse.studenttheses.ub.rug.nl/15851/1/Artificial_Intelligence_Deep_R_1.pdf

[21] Legg, S., & Hutter, M. (2007). Universal intelligence: A definition of machine intelligence. Minds and Machines, 17(4), 391–444. https://doi.org/10.1007/s11023-007-9079-x

[22] Schaul, T., Togelius, J., & Schmidhuber, J. (2011). Measuring Intelligence through Games. https://arxiv.org/pdf/1109.1314.pdf

[23] Rockstar Games. (2020). https://www.rockstargames.com/

[24] Mattar, M., Shih, J., Berges, V.-P., Elion, C., & Goy, C. (2020). Announcing ML-Agents Unity Package v1.0! Unity Blog. https://blogs.unity3d.com/2020/05/12/announcing-ml-agents-unity-package-v1-0/

[25] Bertsekas, D., & Tsitsiklis, J. (1996). Neuro-Dynamic Programming. In Encyclopedia of Optimization. Springer US. https://doi.org/10.1007/978-0-387-74759-0_440

[26] Shao, K., Tang, Z., Zhu, Y., Li, N., & Zhao, D. (2019). A Survey of Deep Reinforcement Learning in Video Games. https://arxiv.org/pdf/1912.10944.pdf

[27] Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017, August 19). A Brief Survey of Deep Reinforcement Learning. IEEE Signal Processing Magazine. https://doi.org/10.1109/MSP.2017.2743240

[28] Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E., & Stone, P. (2020). Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. https://arxiv.org/pdf/2003.04960.pdf

[29] Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., & Lange, D. (2020). Unity ML-Agents Toolkit. https://github.com/Unity-Technologies/ml-agents

[30] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. https://arxiv.org/pdf/1707.06347.pdf

[31] Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. https://arxiv.org/pdf/1801.01290.pdf

[32] Weng, L. (2018). A (Long) Peek into Reinforcement Learning. Lil Log. https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html