Teemu Ropilo

TEACHING A MACHINE LEARNING AGENT TO SURVIVE IN A 2D TOP-DOWN ENVIRONMENT

Bachelor's thesis
Information technology / Game programming

2019

South-Eastern Finland
University of Applied Sciences

Author (authors)	**Degree**	**Time**
Teemu Ropilo	Bachelor of Engineering	November 2019

Thesis title

Teaching a machine learning agent to survive in a 2D top-down environment — 26 pages

Commissioned by

GameLab

Supervisor

Niina Mässeli

Abstract

Machine learning is not a new thing, but the rapid development of computer architecture has brought the tools in the hands of regular users. At the same time, this has led to advances in machine learning assisted game development, introducing new elements to traditional scripted gameplay behaviour.

The objective of this thesis was to train an autonomous machine learning agent that can survive inside a two-dimensional top-down environment, while performing the tasks required to reach the goals set for it. In addition to training the agent, different learning methods were examined and compared, to find the best method for training the agent.

Before the training environment was created, references were gathered, related to machine learning, especially in the area of game development. The preliminary design for the implementation was then created, which was followed by outlining the documentation. To start developing the training environment, the machine learning tools were configured, and initial tests were made, as a proof of concept. At this point all the necessary elements were present, so from this point forward the process continued with improving on each element. During the training of the machine learning agent, statistics were gathered to examine the efficiency of each learning method. Finally, the statistics were compared alongside the machine learning models, to find on the best learning method for the purposes of this thesis.

The findings showed that imitation learning assisted reinforcement learning was the best learning methods. Even though further tuning of the parameters and environment could have given better results, the obtained results were deemed a success. Reinforcement learning and curriculum learning methods also showed promise, but they were not as efficient. The most significant result of this thesis was the accumulated knowledge on the subject, opening new possibilities in the area of machine learning and game development.

Keywords

machine learning, unity, reinforcement learning, imitation learning, top-down, 2D

TABLE OF CONTENTS

GLOSSARY

AI: Artificial Intelligence

BC: Behavioural Cloning

GAIL: Generative Adversarial Imitation Learning

IL: Imitation Learning

ML: Machine Learning

PPO: Proximal Policy Optimization

RL: Reinforcement Learning

1 INTRODUCTION

AI (artificial intelligence) has been around for quite some time and as such is used for many purposes, including game development. One subset of AI is machine learning, which has been gaining mainstream popularity in the recent years. Even though it might not be evident, it is use in many day-to-day applications of our lives, including games.

Even though game industry has still ways to go in how to best take advantage of machine learning, more and more tools are becoming available for public use, creating new possibilities and ideas. One of these tools is ML-agents toolkit (Unity Machine Learning Agents toolkit). ML-agents toolkit enables everyone to start learning about machine learning and creating their own projects, assisted by a great set of online documentation, created by Unity.

This thesis covers the topic of creating an autonomous machine learning controller agent, by using different learning methods. These methods have their own strengths and weaknesses, which bring up the question, what is best method for the purposes of this thesis? Is there a way to train the agent faster, by using a certain learning method? Which parameters are best for getting working behavior models? These questions will be answered when the logic and overall configuration of the agent is successful, and models are generated for comparison and examination.

The first part of the thesis starts by introducing the topic of machine learning and the different methods used for creating the models. The second part covers the setup and the assets used in creating the training environment and the machine learning agent. The third part covers the implementation and results of the different machine learning methods. The last chapters cover the extra functionalities, which were originally planned for the thesis, and finally the conclusion of the thesis.

1.1 Specification of the implementation

The requirement set for this thesis was to create a working autonomous machine learning agent, capable of surviving in a top down shooter environment, where the goal is to evade hazards, while collecting all the collectables to open the exit door. Different machine learning methods were examined, to find the differences between the methods. Requirement for survival was that the agent was capable to reach the goals set for it. The expected result did not need to be perfect, but the agent behavior needed to show clear progress and understanding of the offered challenges.

To reach the set requirements, an environment was created by using Unity framework in addition to ML-agents toolkit, which also included the necessary tools to verify and examine the results of the machine learning training- The tools also included the means to create visual presentations of the progress of the training. This made it possible to meet the requirements for the thesis, including setting up the test environment and examining the results.

The training environment was a top-down 2D rectangle area, which was filled with hazards and collectables. The environment and its elements, combined with the programming, offered the necessary observations for the training of the machine learning agent.

In addition to the machine learning tools, the ML-agents toolkit included the tools to create statistics from the training phase. The statistics were then used to compare the different learning methods, to examine which method worked best for the implementation of this thesis.

2 MACHINE LEARNING

In machine learning, the learning agent is trained to come up with a set of actions to take, in the current state of the environment. This is also called a policy. The better the policy, the more performant the agent is. This policy is created by feeding the learning brain data that it can then use to connect the correct actions to the changing environment, as shown in figure 1 (What is Machine Learning, 2019). This is same as neurons being connected in a human brain when learning new tasks, meaning that machine learning makes it possible for the machine to evolve and get better, in the programmed tasks (Dzhingarov B. 2019).

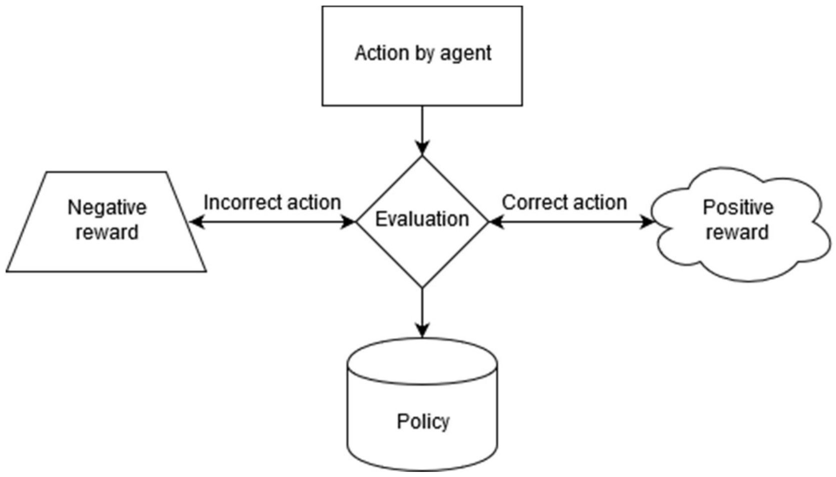


Figure 1 Creating a policy through training

2.1 Machine learning versus traditional AI

Machine learning defines that the machine is learning by itself, compared to the so-called traditional AI, where the machine only follows a prewritten programming to do certain tasks. This means that traditional AI does not evolve, and therefore is limited to accomplishing tasks only in a certain way.

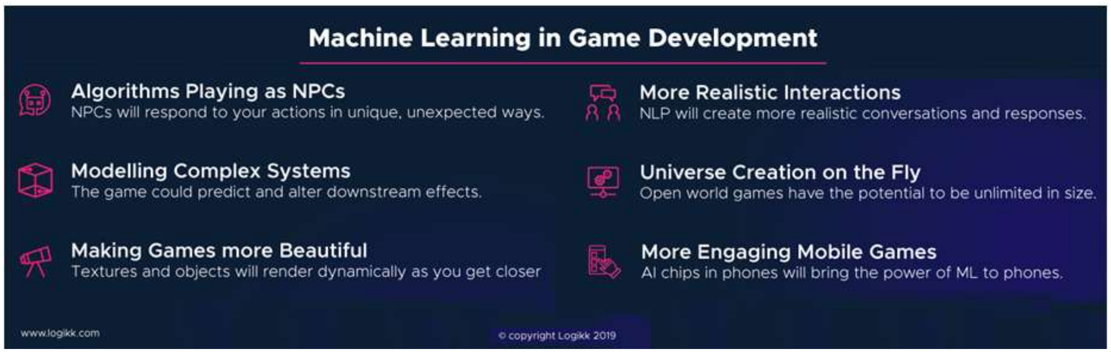
The important thing to understand, when talking about AI and machine learning, is that machine learning is actually a subset of AI, not a separate thing. Machine learning is the part of AI that interprets the data fed to it, either for training the AI to understand the data, or later for finding the patterns in the data (Mueller & Massaron 2016, 20).

2.2 Machine learning in game development

This chapter is based on the WWW-article by Stephenson (2018).

Use of machine learning in game development has been gaining popularity in the past few years. This is mainly due to new GPU architectures making it possible to achieve better and better results, and because of the big-data era which produced significant amounts of data, there exists significant amounts of data to be used for training the machine learning algorithms.

Using machine learning in game development means more unpredictable game world behavior, making environments and actions more dynamic, instead of using pre-scripted actions, in addition to many other ways for using machine learning in game development, as listed in in figure 2. Truly new ways of experience traditional gaming can be invented, when using machine learning.



Machine Learning in Game Development

Algorithms Playing as NPCs
NPCs will respond to your actions in unique, unexpected ways.

More Realistic Interactions
NLP will create more realistic conversations and responses.

Modelling Complex Systems
The game could predict and alter downstream effects.

Universe Creation on the Fly
Open world games have the potential to be unlimited in size.

Making Games more Beautiful
Textures and objects will render dynamically as you get closer

More Engaging Mobile Games
AI chips in phones will bring the power of ML to phones.

www.logikk.com

© copyright Logikk 2019

Figure 2 Ways how machine learning can benefit game development (Stephenson J. 2018)

2.3 Machine learning methods used

In this thesis two methods were used, reinforcement learning, and reinforcement learning combined with imitation learning. These methods were chosen, because they were suitable for the purposes of the learning environment and to the prior level of knowledge on the subject.

Reinforcement learning

In reinforcement learning, the machine learning agent is given data of the state of the environment, while it takes actions that can affect the state. It also has preset goals which it tries to achieve by taking these actions that are related to the state of the environment (Sutton & Barto, 2017, 1). The agent is then given rewards, depending on the actions it takes, and by trial and error, it tries to achieve the optimal policy (Adan, Y. 2018).

In this thesis, Proximal Policy Optimization algorithm was used, as the reinforcement learning algorithm, mainly due to the fact that the ML-agents toolkit, uses this algorithm. This algorithm has been selected by Unity as it has been shown to be a good general-purpose reinforcement algorithm (Getting Started with the 3D Balance Ball Environment. 2019).

Curriculum learning

Curriculum learning is very familiar to humans, as it is how subjects are taught in school. The idea is to introduce easy concepts at first and then build on top of them, to improve on what has already been learned. A good example of this is mathematics, where first numbers are learned, arithmetic skills, then algebra and lastly calculus. This same principle can be used with machine learning, as is done in curriculum learning, teaching the machine learning the required skills one at a time, from easiest to the most difficult.

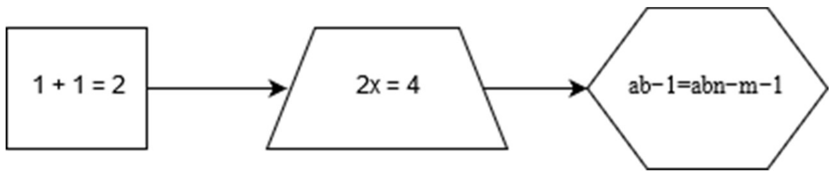


Figure 3 Example of curriculum learning

To make curriculum learning efficient, the learning environment cannot start from an environment with too high complexity, because otherwise the agent would never receive the reward needed to the improve the policy

Imitation learning

In imitation learning, the learning agent is given samples of data to learn from, for example by observing a human play the game, either from offline recordings or in real time. The agent uses this knowledge to perform actions in the same environment, trying to achieve the most performant way to function (Adan, Y. 2018). So, the agent already has prior knowledge of the actions it should take in the environment it is in, which, compared to reinforcement learning, gives a head start for the evolution of the agent. It is possible to combine imitation learning with reinforcement learning, and this is what was done in this thesis, to compare the difference between regular reinforcement learning and imitation learning assisted reinforcement learning.

3 SOFTWARE AND OTHER ASSETS

Creating the environment for the machine learning was done by using the Unity game engine framework. Machine learning portion was created using the ML-agents toolkit. The charts created by Tensorflow were used for the evaluation of the machine learning agent progress.

Unity Machine Learning Agents Toolkit

ML-agents toolkit is a project created by Unity, which has made it possible for everyone to start creating their own machine learning projects. The ML-agents toolkit enables the use of many different methods of machine learning, but in this thesis, imitation and reinforcement learning were the chosen methods.

Tensorflow

Tensorflow is an open source library that is used by the ML-agents toolkit, to create behavior models, also called policies. It is currently not possible, for ML-agents toolkit to use any other library for the models, so there were no options on which library to choose for this project (Background: Tensorflow, 2019).

Tensorboard

Examining the results created by Tensorflow was done using Tensorboard, which is a part of Tensorflow suite. This was crucial for optimizing the agent behavior and comparing the gathered results on how well the agent performed (Background: Tensorflow, 2019).

Art and code

The art used in the project is from a Unity asset called Pixel TopDown Shooter Engine (Stache M. 2018). Many of the used code scripts of Pixel TopDown Shooter were modified, to suit the needs of this project. In addition of the Pixel

TopDown scripts, also new code was created, to tie everything together and to create the necessary programming for the machine learning.

4 SETUP OF THE AGENT CHARACTER AND ENVIRONMENT

The machine learning agent assets and level environment, including its elements, were created by modifying existing asset, to suite the needs of the thesis, even though these were initially not meant for this purpose.

4.1 Setting up the agent-controlled character

The programming of the character control was derived from the Pixel TopDown Shooter Engine. The control system was mainly modified in how the ML-agents toolkit had access to it. The character controls were setup in the same way for both reinforcement and imitation learning methods, as the only difference was, what was controlling the character, the computer, or the human player. Having human controls was also a prerequisite for creating the pretraining material, for imitation learning to work.

Basic functionalities created for the character were moving horizontally and vertically, not including combinations of the axis's, which was a limitation of the discrete action space, as discussed in chapter 5.1. Using discrete action space made moving the character much more efficient, for the machine learning agent to learn from. The other option instead of discrete action space was continuous action space, but this proved to be very inefficient in this use case.

The programming of the agent was modified in such a way that co-evolution would be possible. Co-evolution was used to speed up the training, meaning that multiple copies of the levels, including the agent, were used during the training phase. This significantly improved the training speed, making the overall process more efficient.

4.2 Levels and the environment architecture

The initial testing level, which can be seen in figure 4, was a small rectangular area with surrounding walls. As the training of the machine learning required as much randomization as possible, while still having control over the variables, it was necessary to have the elements inside the levels change programmatically. This was accomplished by creating a randomization function that changed the positioning, amount and state of the level elements.

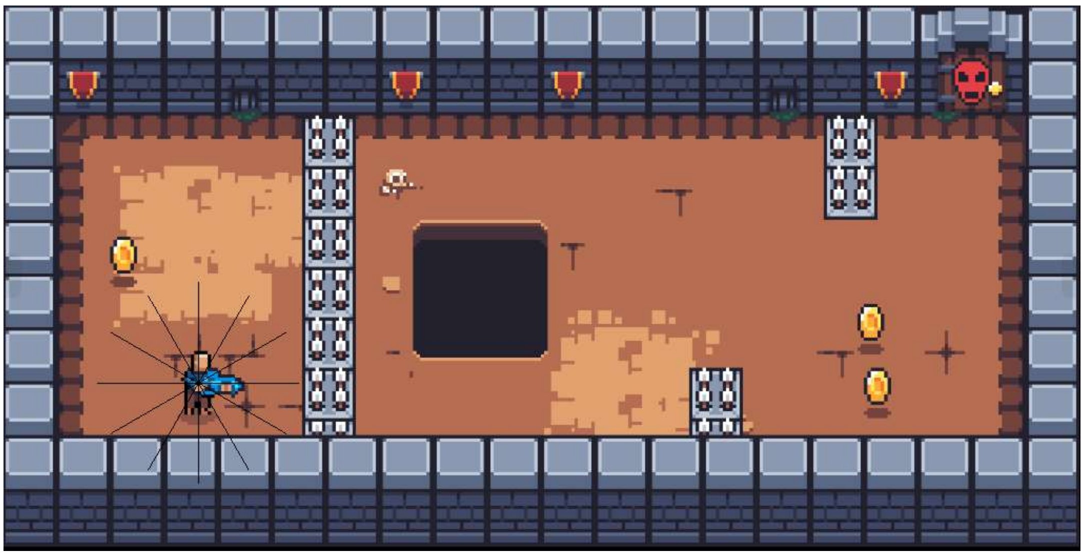


Figure 4 First testing environment

The elements used in the level setup were

1. Collectibles
2. Pit hazards
3. Spike trap hazards that were set to spring the spikes up between set time periods
4. Door for exiting the level after collecting all the collectibles

After the initial tests had proved that the concept was working, a bigger level, shown in figure 5, was created.

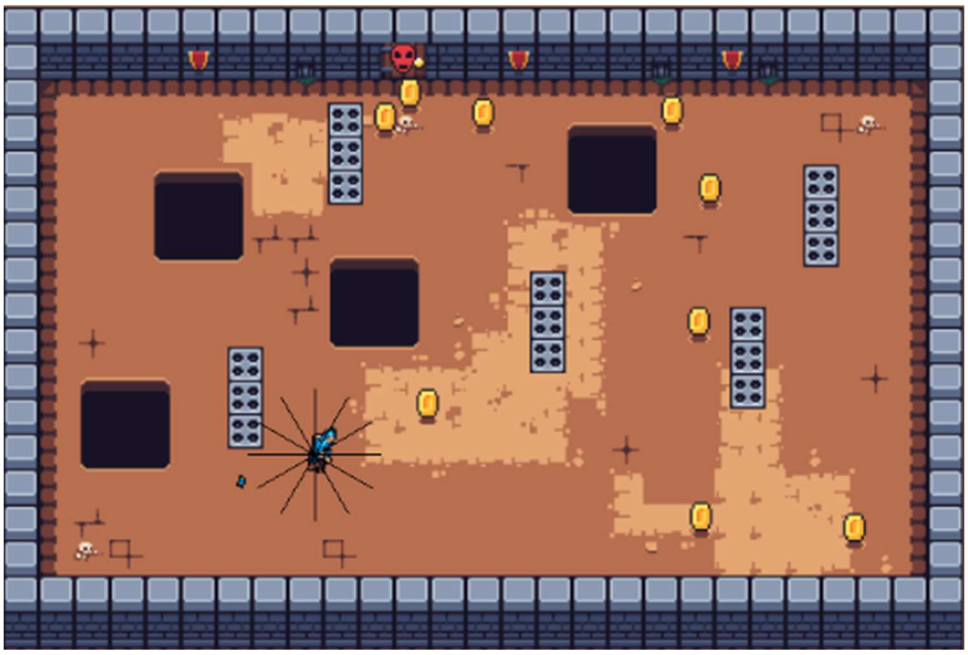


Figure 5 The environment used for training

The bigger level made room for more hazards and made it possible to prove that using machine learning was feasible in a more complex environment. This level setup was kept as the main test environment during the project, for all learning methods.

5 MACHINE LEARNING IMPLEMENTATION AND RESULTS

One of the goals was to compare the results between the different learning methods. Chosen methods were reinforcement, reinforcement with curriculum learning and reinforcement learning combined with imitation learning.

5.1 Reinforcement and curriculum learning

In reinforcement learning, to make the training of the agent possible, three entities needed to be defined, which were observations, actions and rewards (Running Example: Training NPC Behaviors. 2019).

The observation data fed to the agent at all times:

- List of close-proximity objects, including hazards, collectibles and the exit door

- The amount of collectable items
- The normalized position of the next target, being either a collectible or the exit door
- The status of the level exit door (open or closed)
- The state of the spike traps (spikes up or down)
- The time before the spike trap spikes spring up

The actions the agent could take:

- Movement action, either up, down, left, right or staying still
- Collecting the collectables (passive action combined with movement)

The reward signals for training the agent:

- Positive reward for crossing a spike trap without dying
- Positive reward for picking up collectables
- Positive reward for picking up the last collectable in the level
- Positive reward for reaching the exit after picking up all the collectables
- Negative reward for reaching the exit before picking up all the collectables
- Negative reward for death (falling to the pit, dying on spikes)

For curriculum learning, a curriculum list needed to be specified. This included the information on when the difficulty of the level would increase, based on the success of the agent, offering the next goal for the agent to reach. Increasing the difficulty meant adding more simultaneous obstacles, where first just the collectibles were present, then pits were added and finally the spike hazards (Training with Curriculum Learning. 2019).

The end goal for using reinforcement with curriculum learning, was to find out if the agent could successfully learn all the tasks in one training session. For this to work, the challenges set for the agent were first tested separately, making sure they were achievable. This approach offered to possibility to find any issues in the logic or programming and was also a part of the best practices.

The challenges from the easiest to the most difficult:

1. Find the exit door. The door changes its position between levels
2. Find all the collectibles. Collectibles are positioned randomly, then exit the level
3. Evade pits, while finding the collectibles and then the exit
4. Evade spike traps and pits, while finding the collectibles and the exit

Initially the action space, used by the brain to move the character, were continuous, which meant that the numeric values were floating numbers, ranging between zero and one. This caused the movement of the machine learning controller agent to be too limited, causing it to mostly stay in a small area of the map, never finding rewards. This was remedied by changing the movement actions from continuous to discrete, which meant that the values were now fixed values of minus one, zero and positive value of one. Values of minus and plus one moved the agent to opposite directions on either vertical or horizontal axis and zero kept the agent still. Another important step taken to achieve working results, was normalizing the values fed to the brain. When the agent tried to find the next object to reach, it was given a normalized vector, a unit vector, instead of feeding it the full length of a vector.

As seen in figure 6, the vector from the agent to door has a length value of five, but when the value is normalized, it changes to a unit vector with a value of one, still pointing to the same direction. This way the agent was given information for which way to move, and the values fed to the brain stayed inside one and zero, being uniform. Normalizing the values was also important for the algorithm to keep all the information on the same level of importance. This method was also suggested in the best practices by Unity (Environment Design Best Practices. 2019). After these changes, the agent movement was more coherent and started to create positive results.

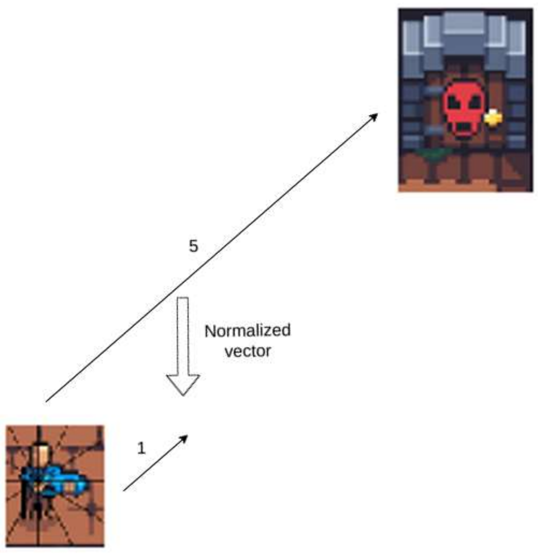


Figure 6 Normalizing a vector

When programming the reinforcement learning, many different variations of the implementation were tested, mainly on what kind of observations and rewards worked well for creating a working policy. To successfully get a working policy, additional parameters, called hyperparameters were adjusted during the process, to the final values seen in table 1. During the early reinforcement phase, it was observed that it was necessary to increase the amount of entropy, to help the agent explore the level properly. More entropy was introduced by increasing the hyperparameter called beta. In addition to increasing the beta value, one of the reward signals available in hyperparameters, called curiosity, was set as true, which encouraged the agent in exploration. The use of curiosity was suggested especially for situations, where there aren't many rewards signals available (Hyperparameters. 2019). This configuration significantly improved the results of the training.

Table 1 Final hyperparameters used for reinforcement learning

trainer	ppo
batch_size	128
beta	1.0e-1
buffer_size	2048
epsilon	0.2
gamma	0.99
hidden_units	512
lambd	0.95
learning_rate	3.0e-4
max_steps	5.0e5
memory_size	256
normalize	false
num_epoch	3
num_layers	2
time_horizon	64
sequence_length	128
summary_freq	2000
use_recurrent	false
use_curiosity	true

curiosity_strength	0.01
curiosity_enc_size	256

5.2 Results of reinforcement and curriculum learning

When initially testing only reinforcement learning, the agent did not always find the exit, but as the hyperparameters were fine-tuned, a working policy was achieved. This resulted in policy that could guide the agent through the level, where there were only collectibles and pit hazards present. An issue with pit hazards was observed, where evading pit hazards did not work when the pit was located at the top of the level, with the top of the pit blocked by a wall, as seen in figure 7. During the reinforcement learning phase, adding the spike hazards in the level at the same time with all the other elements, resulted in a non-working policy.

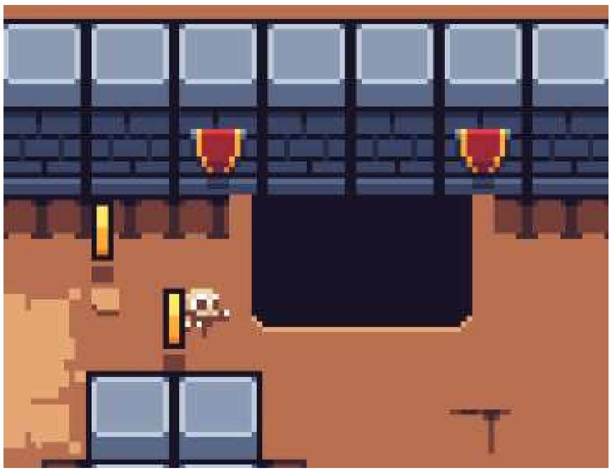


Figure 7 Challenging pit location

After this, curriculum learning was implemented, to examine the success rate compared to pure reinforcement learning. After the initial training sessions, the agent seemed to have very limited success on any action it was supposed to take. Eventually the issue was observed as being the so called "catastrophic forgetting", which is a known issue related to deep learning, where the agent forgets what it had learned in the previous step. As a method to try to prevent this, earlier steps were introduced again during the training process. This helped to a certain degree but did not fully remedy the problem.

This phase of the thesis was deemed a partial success, as the curriculum learning was clearly learning, even though the resulting policy did not achieve the optimal end results.

5.3 Imitation learning assisted reinforcement learning

After training with reinforcement learning and curriculum learning, the next step was to train the machine learning agent using imitation learning assisted reinforcement learning. The implementation was mostly built upon the already existing reinforcement learning setup, so most of the ground work was already done.

With ML-agents toolkit, it was possible to use pre-recorded demonstrations to help the agent learn. For the purpose of reinforcement learning combined with imitation learning, pretraining combined with small amount of GAIL reward signal was the suggested way by Unity (How to Choose, 2019).

Doing the necessary modifications on top the existing setup comprised of recording the necessary gameplay demonstrations and configuring the ML-agents toolkit parameters. Additional parameters seen in table 2 were used for this training phase.

Table 2 Additional parameters used for imitation learning assisted reinforcement learning

pretraining	
demo_path	/demos/Pretrain.demo
strength	0.5
steps	10000
reward_signals	
extrinsic	
strength	1.0
gamma	0.99
curiosity	
strength	0.02
gamma	0.99

encoding_size	256
gail	
strength	0.01
gamma	0.99
encoding_size	128
demo_path	/demos/Pretrain.demo

5.4 Results of imitation learning assisted reinforcement learning

The first tests with added imitation learning, using pretraining combined with reinforcement learning and a small GAIL value, were run with a four-minute recording of gameplay demonstration. The training phase showed positive results, as the agent was able to find the location and evade hazards from the start, but the first resulting model was not very efficient, as reaching the exit door took quite long. The agent movement was also very sporadic, even though the actions were clearly correct. Given enough time, the agent was able to reach collectables and mostly evade hazards but moved too slow and did not always reach the end goal, sometimes getting stuck, moving in a circle.

Next the four-minute demonstration was replaced with a 10-minute demonstration, which resulted to the agent moving less hesitantly, being able to accomplish the level more often. It was clear that the biggest issue was the model not fully comprehending the observations set for the spike trap. This phase of the thesis was clearly more successful, compared to reinforcement learning without imitation learning.

6 ANALYZING THE RESULTS

The imitation learning assisted phase of the thesis was a better solution for this type of environment, as the resulting policies created during all the training sessions were more successful and consistent, compared to reinforcement learning methods. This was mostly due to the imitation learning guiding the agent to the first positive rewards, compared to the reinforcement learning methods, which either needed a lot more time to achieve meaningful amount of rewards, or in the worst case, did not find rewards at all.

Training sessions, which resulted in non-working models, were observed in the very first reinforcement learning training sessions, when only the exit door goal was present. After the collectables were added, which significantly increased the odds of finding positive rewards, the training sessions became successful, which worked as a guidance on how to continue improving the implementation, in all training methods. It was evident that to create a working policy, there either needed to be enough rewards or training had to be assisted with imitation, to guide the brain to the right path, as was done in this thesis.

6.1 Comparing the statistics

Tensorboard has multiple charts that can be used to better understand how successful the training phase has been. The statistics shown below represent the information necessary to get an understanding on the basic level.

Environment Statistics

- Environment/Cumulative Reward – The mean cumulative reward per episode. If training is successful, the value increases.
- Environment/Episode Length – Indicated the mean length of each episode in the environment.

Policy Statistics

- Policy/Entropy (PPO; BC) - The randomness of the model decisions. If training is successful, the value decreases.
- Policy/Value Estimate (PPO) - Indicates the mean value estimate of the states the agent has visited. If training is successful, the value increases.

(Using TensorBoard to Observe Training, 2019).

Looking at the Tensorboard statistics for reinforcement learning with curriculum learning, as seen in figure 8, we can see that at first, the average value of cumulative reward is gradually decreasing, which indicates that at first the training is not very successful. At about 300k steps it stars to increase, which indicates that the model has more understanding of what it should do to get positive rewards. The environment/episode length chart shows that the lengths of the episodes, meaning individual runs when the agent either reaches the goal or fails trying, vary quite much. This is to be expected at first and should get shorter, the longer the training has been running.

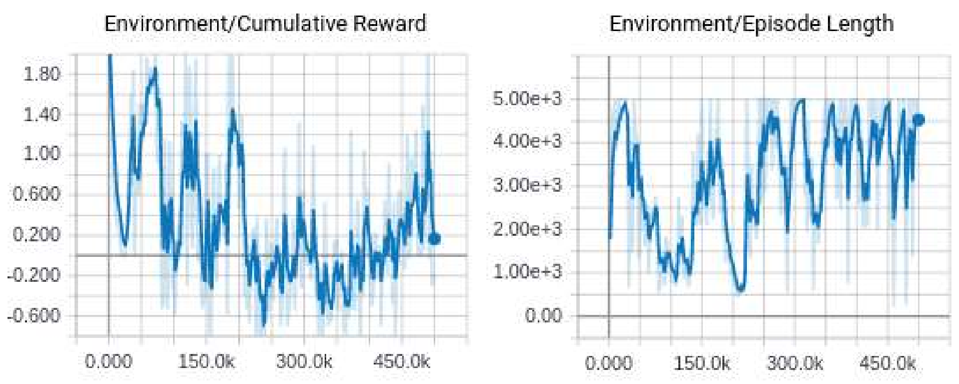


Figure 8 Reinforcement learning environment statistics

As can be seen in figure 9, the policy/entropy values for curriculum learning show that the decisions of the model start to be less random, as more steps have been taken, but the policy/value estimate indicates that the training is not very successful, since the value should be as close to 1.0 as possible.

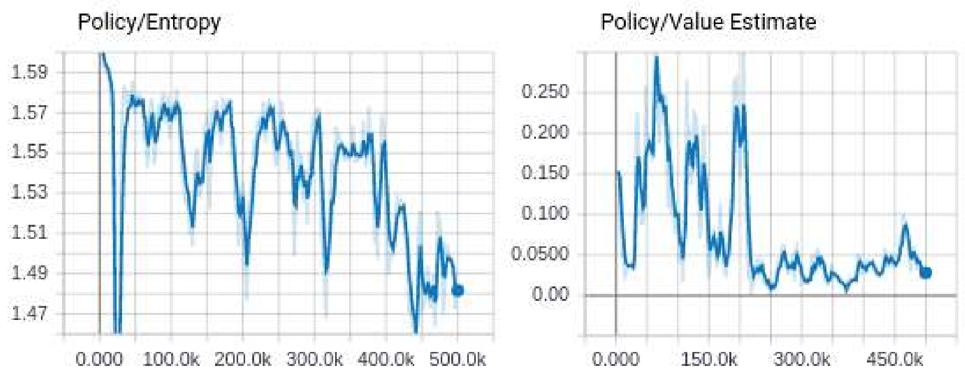


Figure 9 Reinforcement learning policy statistics

The statistics support the findings of curriculum learning, where the model is working to some extent, but failing to successfully reach all the set goals. Contrary to curriculum learning, the statistics for the imitation learning assisted reinforcement learning show that the training is advancing well. This can be seen when looking at figure 10, where the environment/cumulative reward value is clearly increasing, and the episode length is gradually decreasing.

This shows that the agent is both gaining more rewards and reaching the goals faster, as more episodes have been completed.

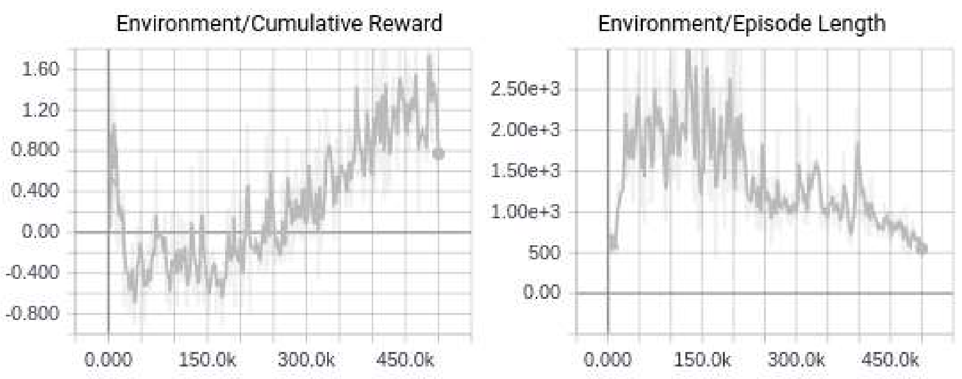


Figure 10 Imitation learning assisted reinforcement learning environment statistics

The policy/entropy values for imitation assisted reinforcement learning, in figure 11 show that the actions are less random, as more episodes are finished. This also supports the finding that the agent is learning correct behaviour. In addition to the policy/entropy values, the policy/GAIL value estimate indicates that the mean value estimate is increasing for the states the agent has visited, further pointing to a successful training session.

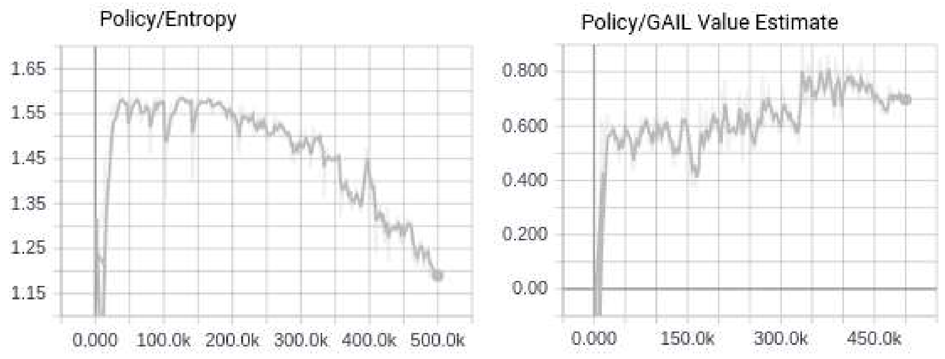


Figure 11 Imitation learning assisted reinforcement learning policy statistics

Examining the Tensorboard charts during and after the training process was an important part of the thesis, giving valuable information on which configurations work best. Comparing the charts between different methods also gave good information on how efficient the different methods were comparatively.

7 EXTRA FUNCTIONALITIES

This project had plans for extra functionalities that were not implemented, due to them being lower priority, compared to getting the basic functionalities working and reaching the primary goals. The extra functionalities are listed below, in no order of importance.

- Additional character actions were created, including rolling, to dodge enemies and obstacles, and shooting projectiles to kill enemies.
- The initial plan for the implementation of imitation learning, was to use a web frontend configured in AWS, servicing a build of the game for human players to play using a web browser, accessible from anywhere in the world. This gameplay would then have served as the training data, used inn imitation learning offline training phase. This was left out, as the setup of the environment would have taken too much time, compared to the benefit received from it.
- Level plans included the use of procedural levels, but this was left out, as the randomized environment elements served the purpose of introducing randomization.
- Collectable items for the agent to pick up, for boosting the character in different way, were planned, but this was left out because of the lower priority compared to other areas that needed development.

8 CONCLUSIONS

The machine learning tools and methods used in this thesis proved to be applicable means for reaching the goals of this thesis. Imitation learning assisted reinforcement learning was the best working option out of the researched methods, even though other methods also showed promise. The next steps to get better results, would have been to tune the observations fed to the brain and to test different hyperparameters. Running longer training sessions could have also created better results.

As a conclusion, it can be said that all the necessary elements for creating a successful machine learning model, for the top-down environment, were present. The resulting models were able to examine the environment and reach the set goals to an acceptable degree, sometimes even commendably. Considering the level of knowledge on the subject, at the beginning of the thesis, the results were a success, and the amount of accumulated knowledge servers as a stepping stone for further development.

REFERENCES

Nigretti, A. 2017. Using Machine Learning Agents Toolkit in a real game: a beginner's guide. Blog. Updated 11.12.2017. Available: https://blogs.unity3d.com/2017/12/11/using-machine-learning-agents-in-a-real-game-a-beginners-guide/ [accessed 23.5.2019]

Dzhingarov, B. 2019. Traditional AI vs Machine Learning and Where Data Labeling Comes into Place. Blog. Updated 6.2.2019. Available: https://www.datasciencecentral.com/profiles/blogs/traditional-ai-vs-machine-learning-and-where-data-labeling-comes [accessed 23.5.2019]

Mueller, J. P. & Massaron, L. 2016. Machine Learning for Dummies. For Dummies. E-book. John Wiley & Sons, Incorporated. Available: https://kaakkuri.finna.fi/ [accessed 23.5.2019]

Stephenson J. 2018. 6 Ways Machine Learning will be used in Game Development. WWW-document. Updated 29.11.2018. Updated 29.07.2018 Available: https://www.logikk.com/articles/machine-learning-in-game-development/ [accessed 23.5.2019]

Sutton,R.S. & Barto, A.G. 2017. Reinforcement Learning: An Introduction. E-book. The MIT Press. Available: http://incompleteideas.net/book/bookdraft2017nov5.pdf [accessed 30.05.2019]

Adan, Y. 2018. What is the difference between imitation learning and reinforcement learning? Discussion group website. Available: https://qr.ae/TiNY2J [accessed 30.5.2019]

Multiple. 2019. Unity Technologies, Background: Tensorflow. WWW-document. Updated 1.2.2019. Available: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Background-TensorFlow.md [accessed 6.6.2019]

Multiple. 2019. Unity Technologies, Running Example: Training NPC Behaviors. WWW-document. Updated 1.2.2019. Available: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/ML-Agents-Overview.md#running-example-training-npc-behaviors [accessed 6.6.2019]

Multiple. 2019. Unity Technologies, Training With Curriculum. WWW-document. Updated 1.2.2019. Available: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-Curriculum-Learning.md#how-to [accessed 6.6.2019]

Multiple. 2019. Unity Technologies, Environment Design Best Practices. WWW-document. Updated 1.2.2019 Available: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Best-Practices.md [accessed 6.6.2019]

Multiple. 2019. Unity Technologies, How to Choose. WWW-document. Updated 1.2.2019. Available: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-Imitation-Learning.md#how-to-choose [accessed 25.11.2019]

Multiple. 2019. Unity Technologies, Using TensorBoard to Observe Training. WWW-document. Updated 24.10.2019. Available: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Using-Tensorboard.md [accessed 26.11.2019]

Multiple. 2019. Unity Technologies, Hyperparameters. WWW-document. Updated 1.2.2019. Available: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-PPO.md#hyperparameters [accessed 6.6.2019]

Stache M. 2018. Moose Stache. Pixel TopDown Shooter Engine. Game framework asset. Updated 14.11.2018. Available: https://assetstore.unity.com/detail/templates/systems/pixel-topdown-shooter-engine-131989 [accessed 6.6.2019]

Multiple. 2019. Unity Technologies, ML-Agents Toolkit Overview. WWW-document. Updated 1.2.2019. Available: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/ML-Agents-Overview.md#running-example-training-npc-behaviors [accessed 6.6.2019]

Multiple. 2019. Unity Technologies, Getting Started with the 3D Balance Ball Environment. WWW-document. Updated 1.2.2019. Available: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Getting-Started-with-Balance-Ball.md [accessed 9.6.2019]

Multiple. 2019. Unity Technologies, Training Imitation Learning. WWW-document. Updated 19.2.2019. Available: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-Imitation-Learning.md [accessed 18.6.2019]