Modelling Human Player Sensorial and Actuation Limitations in Artificial Players

André Soares
Instituto Superior Técnico
Porto Salvo, Lisboa
andreghsoares@gmail.com

ABSTRACT

In game design, one of the most important tasks is associated with the playtesting process, as this is where game designers are able to understand if the game experience they are trying to create is indeed being passed to the players. How a player perceives and reacts to a game should be important in the game design process. Work developed in the Deep Learning research field has proved to be a great source of information as to understand how an agent is capable of achieving great runs and scores playing various games from scratch. The capability of testing different games using a screen capture artificial player powered by Convolutional Neural Networks allows for a good understanding of how an agent is capable of extracting important features from the game screen without additional information. In this work, we took the developments in the Deep Reinforcement Learning field applied to Atari environments, mainly Deep-Q Networks, modulated different types of player limitations, tested and documented the results achieved. The results seem to indicate the existence of different types of playing patterns for different limitations.

INTRODUCTION

Looking at the development process of a new game, playtesting is seen as an indispensable part of the game design and development process but in part somewhat costly. Game developers need to expose their games to their intended audience and only through playtesting will a level designer understand whether the components he or she carefully assembled are able to elicit the game experience they were designed for, identifying potential design flaws and gather feedback. This game experience can vary from player to player, taking into account details related to how a player can perceive and interact with a game.

A person affected by color blindness can end up developing different strategies on how to play the game. Take Splatoon 2[1], as an example. Ink is the main projectile used in the game and is also a way of moving through the game's levels. The ink used by each team have different colors and is used by each players teams to perform various actions. If a player has difficulty distinguishing between his team color and the enemies team color he might have to find different ways and strategies in order to gain advantages. If we look at the comparisons of Actions per Minute (measure of how many clicks and key presses a player can perform in sixty seconds) between professional players and casual players of the game Starcraft 2[2], there's a big disparity in numbers. According to an article by Wong [12] a professional player averages about 300 APM going up to 400 and beyond, while a casual player averages around 60 going up to 100 APM. In this case the strategies applied by professional players will be different from casual players, since they are able to act and react faster. Having the game developers capable of exposing their games to different types of artificial players as a way of simulating a specific type of audience might bring good results in their chase for the best design options. The area of General Game Playing has specialized itself in producing artificial agents capable of playing games they never been in contact before, with a variety of approaches, ranging from different types of frameworks to different types of learning methods.

We try to understand if it's possible to accurately modulate a certain type of player limitations through an artificial agent. Can this model replicate the nuances of different types of player limitations, be it either in how the game information is perceived in the form of input and how the following processed information is passed and executed as output of the game information.

In order to solve the problem described, we looked at replicating an artificial player capable of having a level of resemblance between itself and a real human player. Since we tried to understand how the manipulation of input and output of game information influences the type of results and patterns the artificial player might have, our proposed hypothesis consists on using a model that allows us to replicate different types of sensorial and actuation limitations on top of the deep learning algorithm we use to create the artificial player. Using specific scenarios(delays, lag, action limitations), we want to guarantee that this type of model is capable of achieving the same type of results and resemblance that a limited player.

[1]Splatoon 2, Nintendo, 2017, Nintendo Switch, https://splatoon.nintendo.com/

[2]StarCraft II: Wings of Liberty, Blizzard Entertainment, 2010, Dustin Browder and Chris Sigaty, PC, https://starcraft2.com/

GENERAL GAME PLAYING

General Game Playing(GGP) [3] is seen as the idea of having an Artificial Intelligence capable of playing a game it has never been in contact with before, without any kind of human help. These agents are systems capable of accepting the descriptions and the list of possible actions of a game and with that information evaluate what is the best possible course of action to take in order to play the game. By continuously playing the game, the agent will be able to learn from its past experiences and develop new strategies that allow it to have a better performance and edge closer to the way a human player might have played that game.

General Video Game Playing

By applying the concepts of GGP into the video game world, the agent must understand how the game reacts to the actions it performs and how these actions affect the rewards it gets from performing said actions and ultimately understand how to beat and win the game. The agent should be provided at least with the current state of the world it is in and what actions it might perform, with the rest being up to the agent to decide. This allows the agent to be put in several different games, with unique stories, characters and goals across them with the only constant being the method the agent uses to interact with the current game it's playing.

Arcade Learning Environment

A framework that provides an interface to a large number of Atari 2600 games [6], where the agents given as their input the game screen capture and the score counter and from that they are able to say which set of outputs (set of buttons of the Atari controller) determine the next action to execute in game.

GENERAL VIDEO GAME AI COMPETITION

Set up in 2014 the General Video Game AI(GVG-AI) Competition explored the problem of creating controllers for General Video Game Playing and how developers could create a single agent that is capable of playing any game it is given [11]. Throughout the competition, created agents are run in a number of video games in order to observe if they show a type of general intelligence and behaviour that can be associated to a possible human player behaviour. Since its initial conception, the framework has been expanded in order to meet the demand of different research directions. So in its current iteration, as described by Perez-Liebana et al. [9], the competition covers 3 major branches:

- Agents capable of playing multiple unknown games with/without access to the game simulations;
- Agents capable of designing new game levels;
- Agents capable of generating game rules.

LEARNING APPROACHES

Knowing that the agent should be learning from scratch and not from preconceived data, the focus of this work is on Reinforcement Learning algorithms since the agent in the absence of data, must learn to achieve a goal in an uncertain, potentially complex environment.

Convolutional Neural Networks

Since this work tries to emphasize the aspect of having the capability of recognizing the screen game as the input to the artificial agent, we needed a way on how to extract information from what the artificial agent is able to see on the screen. In the realm of Neural Networks, Convolutional Neural Networks are a type of Neural Network usually responsible for tasks like image recognition and classification (image data feature extraction). It was popularized and brought to attention in the work of Krizhevsky et al. [4] where they trained a large CNN in order to classify 1 million high-resolution images with smaller error rates than the competition they faced in the year of that work.

The CNN image classification receives an input image that it classifies under certain categories. This inputs are passed as arrays of pixels to the CNN and depend on the image input resolution. The layers of a CNN are organized according to 3 dimensions: width, height and depth. The hidden layers of a Convolutional Neural Network are the component responsible for the convolution and pooling operations, which are performed in order to detect features followed by a classification process where fully connected layers serve as classifier for the previously extracted features.

First the convolution process is applied on the input by using a filter over that input. After executing multiple convolutions on the received input using different filters resulting in different feature maps, these are put together and form the convolution layer's final output. Since the resulting feature map is smaller than the input received, usually padding is used in order keep the spatial size constant, improve performance and to secure the filter and stride fit the input. It's usual to add a pooling layer after a convolution layer in order to continuously reduce the dimension, parameters and computation along the network resulting in smaller training times and preventing overfitting. The classification task is formed by a set of fully connected layers where their neurons are connected to all the previous layer neurons in order to, after training be able to classify the outputs.

Recently Gatys et al. [2] showed how a Convolutional Neural Network can be used to create a picture that combines the content of one picture with the artistic style of another picture. In the context of General Video Game Playing, Kunanusont et al. [5] used CNNs in order to extract important features from a game using screen capture in a work closely based and related to Mnih et al. [7, 8] work.

Deep Q-Network

In the Mnih et al. [7, 8] proposed model, the neural network used is the previously described Convolution Neural Network, trained with a variant of the reinforcement learning technique Q-learning, in which the CNN is able to successfully learn control policies by receiving as input different Atari 2600 games screen's raw pixels from the Arcade Learning Environment [1], and return as output a q-value function estimating future rewards.

With the algorithm pseudocode 1 as reference we take a deeper look at the Deep Q-Network structure.

Input

The model applies to the raw Atari input frames some preprocessing methods in order to reduce its dimensionality, turning it less demanding from a computational aspect. The game frames are converted from a RGB representation to gray-scale and down-sampled. As a final result the input representation is given by cropping an 84x84 region of the image that captures a small size of the playing area.

Network

The exact network structure as described by the authors consists of a first hidden layer which convolves 16 8 × 8 filters with stride 4 with the input image and applies a rectifier nonlinearity. As for the second hidden layer, it convolves 32 4 × 4 filters with stride 2, again followed by a rectifier nonlinearity. The final hidden layer is a fully-connected layer and consists of 256 rectifier units. The output layer is a fully-connected linear layer with a single output for each valid action [7, 8]. This outputs have the corresponding action predicted Q-values for the input state.

Experience Replay

All the observation and information is packed together in what is called “experience" at each time-step, to be stored in an experience pool. Some drawn at random experiences in the pool are sampled to update the network using Q-learning according to the last 4 frames of the selected experience, in what is called “experience replay". Knowing that in reinforcement learning successive states are similar, there’s a risk of the network forgetting what a state is like in which it has been in some time. By applying replay experience the network will still be able to access past frames. Although the replay is a good way of accessing past experiences it is limited to the last N experiences and samples uniformly at random while updating, which in some aspects is limiting since the memory is unable to form differentiating important transitions and needs to constantly be overwritten with newer experiences.

Policy

After the “experience replay" process the agent selects an action taking into account an E-greedy policy, which means that sometimes it picks actions randomly in a way that the model can learn about an action it doesn’t think is optimal, and sometimes pick actions according to the model so that it moves forward in the game and learns different states.

BREAKOUT

’Breakout’ [3] [6] begins with eight rows of bricks, with each two rows with a different color, with the color order beginning from the bottom to top as blue, green, yellow, orange and red. The game’s screen is presented in Fig. 2. By using a single ball, the player must knock down as many bricks as possible, using the walls and/or the paddle controlled by the player as a way of ricocheting the ball against the bricks and eliminate them. If the paddle misses the ball’s rebound, the player will lose a life. For each run, the player has three lifes to try to clear two screens of bricks. The yellow bricks earn one point each, the green bricks earn three points, the orange bricks earn five points and the the red bricks earn seven points each. The player controlled paddle shrinks to one-half its size after the ball has broken through the red row and hit the upper wall, with the ball speed increasing speed after four hits, twelve hits and after hitting an orange or red row. This was the chosen game for some reasons. The game ball has different trajectories and different speeds that could influence the way a player with physical limitations or with slower reaction interacts with it and affect their capability of achieving a better highscore after finishing the game. Another reason is attached to the fact that this game has been used in various implementations of Reinforcement Learning and has a wide range of testing results available to cross check and compare.

[3]Breakout, Atari Inc., 1976, Nolan Bushnell and Steve Bristow and Steve Wozniak, Atari 2600

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

Figure 1. DQN algorithm pseudocode

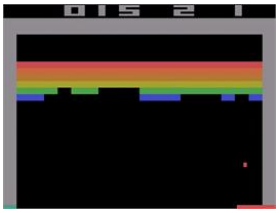

Figure 2. Breakout’s game screen

IMPLEMENTATION

In this section, we present the base on which our solution model is rested. We present the technologies that were used in order to replicate the reinforcement learning algorithm used, the results achieved by training and testing the network in our environment of choice as well as comparing the results with the documented results on the Mnih et al. [7, 8] work papers.

Technology

The development language used for the replication of the algorithm and further implementations and tests was Python. In order to simulate the Atari and game environment we used a Python library called Gym[4] developed by the OpenAI company. Gym is a toolkit for developing and comparing reinforcement learning algorithms and capable of simulating large numbers of reinforcement learning environments, including Atari games as we wanted for the solution. In order to replicate

[4]Gym, https://gym.openai.com/

the Neural Network of the reinforcement learning algorithm, we used the Keras API library[5] with the Tensorflow GPU implementation[6][7] as its base.

The algorithm implementation is provided by Plappert, Matthias Keras-RL[10]. This library implements some state-of-the art deep reinforcement learning algorithms in Python and is integrated with Keras, working with OpenAI Gym, providing easy evaluation and tinkering of different algorithms.

Replicating Paper Results

In order to guarantee the accuracy of the implementation and results, we needed to run the code on our setup, so we could cross check it against the results obtained in other runs of the code and the data presented in the Mnih, et al paper.

For this we performed a series of tests that allowed us to check how similar our results are. We trained the network and extracted the game high scores, the reward per episode evolution, the mean q value through the network training process and an average of the actions performed. Following Mnih, et al paper each epoch corresponds to 50000 steps of training, so we looked to limit our training data to a maximum of 35 epochs (1.75M steps). Acording to Mnih, et la paper, in order to have a accurate depiction of a DQN network we should have an average Q value of around 2.5 and an average reward per episode of over 60 for 1.75M training steps.

In the data we got from training our network (see Figures 3 and 4) the results are in part similar to what was expected according to the paper data. The average Q value evolves in a similar fashion to the corresponding paper data, while for the average reward per episode case the results are not always similar. Initially the reward progression is in line with the paper values but, although that for the 35 training epoch mark of the paper it had a somewhat close value (around 60 to ours 51 average), it's possible to see that it also oscillates around higher values at this point. The difference in values might be due to different software versions and hardware configurations since the progression of values seems similar, therefor we will use our obtained results as our baseline moving forward.

Solution Model

After forming the baseline to our work, we were able to take the algorithm implementation and apply our hypothesis on top of it.

Following Mnih Deep Q Network architecture[8], we propose the model present on Fig.5. Here we see 2 major differences in comparison to the original architecture. The original model elements are represented in green and blue, while our model is represented by the orange states. The blue and green states represent the procedures a normal DQN implementation would take. And so, on top of that model, first we have a state where we treat the action that the network passes to be executed on the environment. This means we are able to regulate the action

Figure 3. Average Q for 1.75M steps training

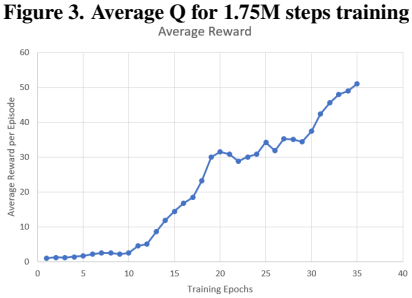

Figure 4. Average reward for 1.75M steps training

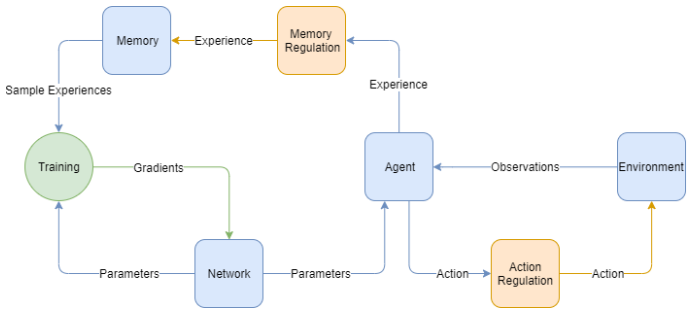

Figure 5. Solution Model Architecture

how we deem fit to simulate some types of limitations we want to enforce on the agent. This action is previously selected by the agent, and might suffer some type of change when passed through our state. Next is a state that regulates the storage of experiences on Memory, limiting the set of experiences the agent can save and sample through it's experience replay. We wanted to limit what the agent would remember from executing different actions for different states, limiting what it counts as training information.

With this model our intention was to try and model limitations in a way we formulated our hypothesis, focusing on simulating errors on the execution of certain types of actions, applying delays on the execution of actions and force a possible visual misses of certain states. These limitations, procedures and results are further explained in the next chapter.

With this model we hope to comprehend how these limitations can influence the network we are training, how does it behave and what kind of results it achieves in comparison to the original network and between different limited networks.

[5]Keras, https://keras.io/
[6]Tensorflow, https://www.tensorflow.org/
[7]Tensorflow GPU, https://www.tensorflow.org/install/gpu
[8]DQN Lecture, https://drive.google.com/file/d/0BxXI_RttTZAhVUhpbDhiSUFFNjg/view, Last accessed: 27 October 2019

Limitation	Value	Reward	NºSteps	Highscore	Don't Move	Fire	Right	Left
Standard	-	51	1646	105	541	124	487	494
Frame Delay	1	27	991	39	318	205	211	257
Frame Delay	2	19	720	28	194	18	86	422
Frame Delay	3	13	627	16	17	45	311	254
Frame Delay	4	25	958	37	138	477	123	220
Frame Delay	5	19	757	25	280	119	148	210
Frame Delay	8	15	580	27	225	6	83	266
Both Actions Reverse	10%	37	1300	67	297	97	250	656
Both Actions Reverse	30%	31	1143	46	0	359	270	514
Both Actions Reverse	50%	22	846	28	69	78	239	460
One Action Reverse	10%	48	1635	90	137	457	427	614
One Action Reverse	30%	47	1447	92	699	288	265	225
One Action Reverse	50%	44	1230	107	341	393	186	310
Both become 'Don't Move'	10%	45	1506	84	85	96	628	697
Both become 'Don't Move'	30%	41	1517	77	505	112	430	470
Both become 'Don't Move'	50%	41	1476	71	445	85	495	451
Miss Frame	10%	42	1285	96	171	193	340	581
Miss Frame	20%	27	897	27	165	153	252	327
Miss Frame	30%	8	414	8	163	146	37	68

Figure 6. Table with all the Test results from various training agents with limitations

RESULTS AND ANALYSES

In this section we present the results of the various types of manipulations tested, with multiple iterations between themselves. All the networks trained and presented from here on further were trained following the same basis as the original network, albeit following our model structure for specific situations. We use rewards and highscores as benchmarks because highscore simply shows the final score obtained while the reward represents the total reward the agent achieved per episode of play. They are related but the values might differ, although smaller rewards tend to have the same value as the highscore.

Action Change

Set of action limiting implementations based on the principle of having an error probability associated with the execution of a certain type of action. This way, we could formulate an idea of what happens when the action intent doesn't go according to plan. This is a way of either simulating the execution and try of last second changed actions, the limitation of performing an action, the execution of an action contrary to what it was supposed to happen or at a more simple level, a possible problem associated with hardware malfunction. In this section we split the action change study into 3 main subsections, each having 3 different error probabilities.

Both Actions Reverse

Results

We can see a clear worst performance, with the reward and highscore(see Table on Figure 6) values achieved by the agent dropping when compared to the original implementation results. Subsequently the number of steps the agent is able to take during an episode of a game is also lower. When observing the progression it has for different error probability values, it's possible to see that the drop off in values between the 10% and 50% error probability is as accentuated as it was between

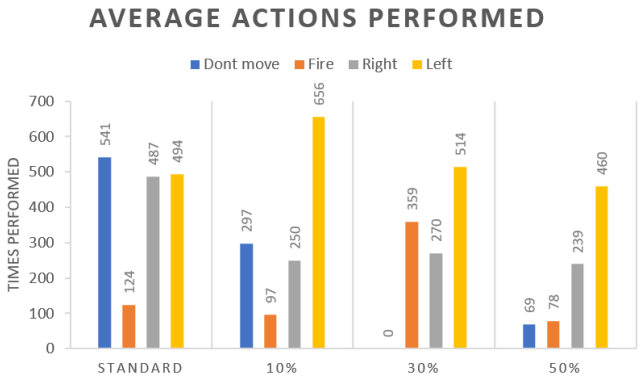

Figure 7. Action Distribution for Both actions reverse Training

the standard agent and with the 10% error probability with the same pattern standing for the highscores.

In terms of actions performed(see Figure 7) we can state that the 10% action distribution resembles a lot the standard action distribution, having a good balance between non moving and moving actions, albeit with more focus on the 'Left' action. As the error probability increases we see a number of notable changes. The slight decrease of non moving actions and also the decrease in the difference between the 'Right' and 'Left' action from 10% to 30% but not from 30% to 50% where the proportions are maintain.

Analyses

The discrepancy between the standard values and this scenario training values is quite significant, which led us to believe that the unpredictability and uncertainty associated with reversing both actions provides the agent a big hurdle to overcome when trying to train for the best possible outcomes.

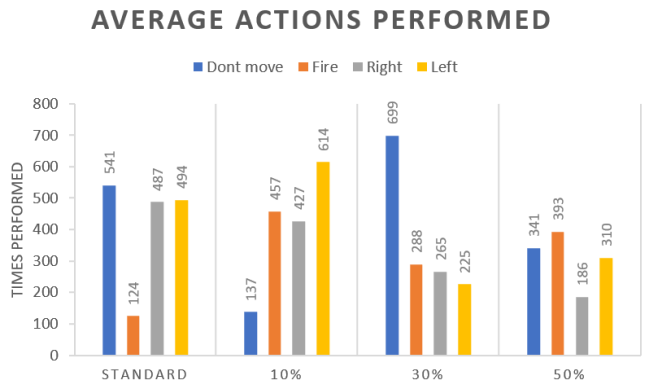

Figure 8. Action Distribution for One action reverse Training

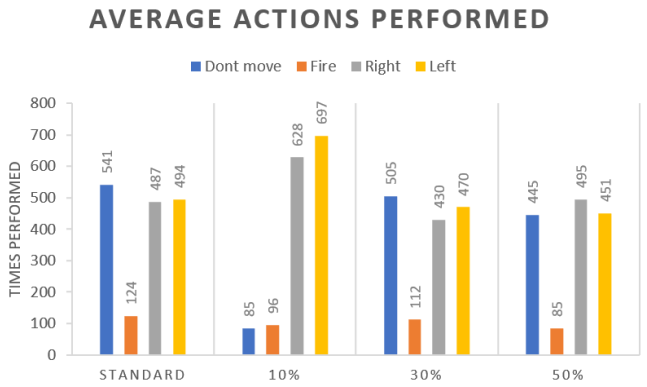

Figure 9. Action Distribution for Both actions changing to 'Don't Move'

If we look at the actions the agent performs in Fig. 7, it's possible to state that despite the number of steps the agent is capable of taking per episode is reduced, it still is able to maintain a similar proportion of moving actions. He does that at the expense of performing non moving actions and although it isn't able to achieve great results, we can see that it tried to adapt to it's imposed limitations.

One Action Reverse

Results

For this scenario we obtained some interesting results. Looking at the whole spectrum we can see that the reward progression throughout the increasing higher error probabilities displays only a drop of under 10 reward values. And looking at the high score values (see Table on Figure 6) it's possible to state that after initially dropping off in relation to the standard training, they progressively see an increase in value while increasing the error probabilities. In terms of steps executed by the agent, the number of steps per episode don't see a big reduction between the standard training and the different reversion error probabilities, and thus it's no surprise that for the lowest of error probabilities the actions performed are still similar to the standard trained agent. As for the other error probabilities, it's visible a clear increase in the number of non moving actions performed at the expense of moving actions(see Figure 9).

Analyses

Here we can conclude that the agent is capable of better adapting to the limitations where it is only required to reverse one action. The unpredictability still exists but in smaller quantities. As stated before, the action distribution for 10% seems to be identical to the previous tests, seemingly indicating that the uncertainty to this point caused on the agent the same type of reaction as it did in previous tests. After that the agent clearly shows signs of preference towards non moving actions, indicating that it would rather have the paddle stay in the same position and only act on situations it would be more certain of the outcome and trying to reduce the possible error while applying the 'Right' action.

Although the only one type of action can reverse scenario might not be as important to replicate real human limitations, it can still represent some kind of impact at the game design level, for example by having one action execution more facilitated than the other. Seeing as the highscore obtained for 50% error probability is higher than standard values, it might be interesting to explore a game with this types of characteristics.

Both Actions become 'Don't Move'

Results

Again we have decreasing values for average reward and high scores. similar to the One Action reversing scenario. The number of steps executed by the training agent are very consistent between error probabilities and very close to the standard values. In terms of action distribution (see Figure 9), for 10% error we see a high abundance in moving actions, with very little execution of non moving actions. From there on out, the executed actions paradigm changes a bit as the moving actions decrease in terms of executions, while the non moving actions ('Don't Move' mainly) increase.

Analyses

The agent was again able to adapt in similar fashion to the limitations in play. The maintenance of close to standard numbers of steps per episode is a clear evidence of that, albeit the rewards and high scores are not up to par with the standard iteration. The actions executed for the 10% error probability are intriguing. Here we have a high volume of moving actions whereas the non moving actions represent a small percentage of the total actions executed. The speculation is that this increase in volume of non movement actions and the conservation of the execution of high number of moving actions is due to the fact that the agent executes a higher number of actions that lead to movement in order to counter the existence of a probability of changing the actions to 'Don't Move'.

Action Delay

For this type of limitation, we replicate a delay in the execution of an action by the player. This might be result of slower reaction times on players or movement deficiencies. For that, every state we calculate which action is to be executed on the environment and store it onto an array of delayed actions. From then on the delayed actions stored are to be executed in future frames of the game according to the delay we define beforehand. On Figure 10 we display a visual guide of the process that happens on every frame for processing the current calculated actions and the delayed actions.

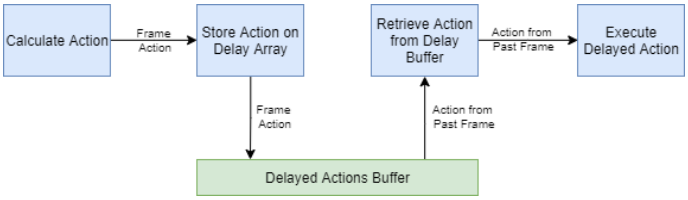

Figure 10. Delay Process for each Frame

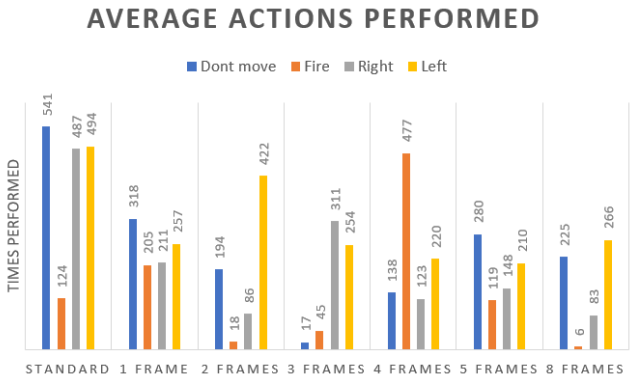

Figure 11. Action Distribution for Both actions changing to 'Don't Move'

Results

The results are much worse than the standard values. The rewards and high scores obtained are much smaller, as well as the number of steps executed on each episode. Although the drop off is high for this type of limitation, it's visible that for 1 and 5 frame delays the action distribution it's in line with what we have been expecting from standard values. Beginning on the 2 frames delay the action distribution begins to be more erratic (see Figure 11).

Analyses

The delay implemented caused some problems to the training agent. The 1 frame delay is the best in terms of results, which seems plausible because we are only forcing the agent to pass it's intended action to the next frame. The actions executed for the 1 frame delay also represent the best adaptation to the delay by the agent, since it has a good distribution of actions, while the higher delay frames don't.

In the 2 frame delay the agent prefers to execute the 'Left' action a larger number of times, trying to maybe compensate the possible incapability of following normal standards and reaching the ball. If these results are confirmed as patterns adopted by real players, it would be of extreme importance to the game design, since such a small delay would be important of taking into consideration during the game design process.

For the 3 frame delay the high percentage of moving actions indicate that the agent is trying to do its best to achieve higher reward values, knowing that with the delay it might not be possible. By forcing more moving actions may lead the agent to reach the ball in more occasions.

We can state that when the delay spreads into another training step (4 delay frame), it exists an increase in reward values.

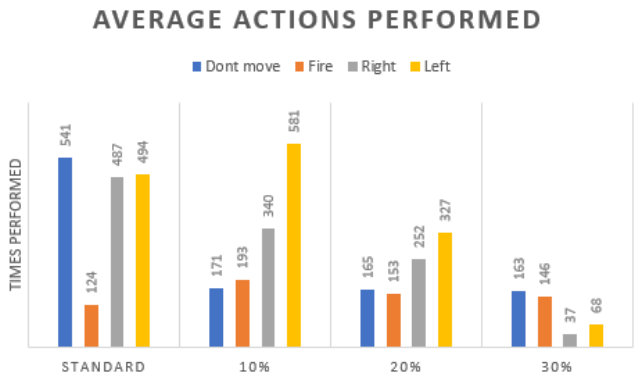

Figure 12. Action Distribution for Missing Frame Training

This seems to indicate that by pushing the actions 1 training step forward allows the agent to better adapt to the delay in place. On this case, it decided that using more non moving actions would allow for better results, something that looking at previous results and the 1 frame delay might be true. From 4 frames forward the rewards decrease again and spreading the delay into another training step(8 delay frame) might be too much for the agent to handle and adapt.

Miss Frame

We wanted to simulate a visual type of limitation a player might have, or a simple visual miscue. How would an agent react if from time to time it couldn't observe the environment? Basically, every state we have a probability of having what we called a visual miss. This consists of telling the agent that the frame in question is not to be actuated and observed. We do that by forcing the action to be executed on that state to be either 'Don't Move' or 'Fire', because the environment always forces an action to be executed. Additionally we don't store that state and information of the action in the memory of experiences. This way it's as if we never saw the current frame and so we simulate a missed frame.

Results

The results obtained for this limitation scenario brought some problems with them. Through various test runs they didn't seem to be always as coherent between themselves, achieving a wide range of results. The values seem to indicate a fast decline in reward and highscore values when the probabilities increase and in terms of action distribution (Figure 12) the decrease of moving actions is evident, while the non moving actions tend to maintain the same volume of execution.

Analyses

Looking back at the training process, we are inclined to believe that the variation of results we obtained where due to the fact that we are simulating a wide range of frames being missed. That might be the cause to why with higher percentages the results tend to dip in performance so abruptly. If we look at the action distribution, we see a pattern of play that evolves with the increase of miss probabilities. This pattern seems to be maintained throughout the various probabilities, in which the agent forces the execution of non moving actions in detriment of moving actions. The results obtained show that this pattern

doesn't seem to achieve great results but it's probably the best option the agent found to counter the increasing probability of missing more and more frames.

CONCLUSION

In this document we proposed a model that would allow to accurately modulate certain types of player limitations using deep learning algorithms. The results achieved allowed us to conclude that a series of patterns of plays we believe would be visible in real human players and real playing limitations are present in our trained agents. For example, in the case of changing actions the agent found various ways of countering the limitations imposed. The patterns ranged from forcing the executions of a certain type of action in order to compensate for the probability of changing actions, to showing preference in having the games paddle stay in one place and only move when it was most rewarding, compensating the possibility of actions changing a high number of times. For the delay the conclusions are not as clear, but we know that the agent was able to better adapt to delays that spread exactly from one training step to another while the intermediates tended to achieve worse results due to cross training actions being executed between 2 training steps. The missing frames tests don't provide us we great conclusions, since the results obtained varied for the same training, but taking into account the averages, the agent tried to adapt to the situation in play by reducing the moving actions, therefore changing its pattern of play as well. If these observations and patterns are indeed confirmed, through user tests, we could affirm we have found a good way of simulating various types of player limitations using Deep Learning methods, through the implementation of our proposed model.

Future Work

Although the results documented seem promising, they are merely theoretical. In order to prove the accuracy of the results obtained, the future works needs to focus on testing the same type of limitation in real players. Be it physically limited players, or simply through the applications of limitations into the games core mechanics. As an example, taking the results of the moving actions becoming non moving actions, it's concluded that the agent tries to force an execution of more moving actions to compensate for the high probability of action changes. By forcing this limitation on testing players, we should be able to observe the same type of results or since our training was limited, the same type of strategies as we were able to obtain. If the results are indeed confirmed through user testing, it should also open the door to a wider variety of limitations to be implemented and explored, as well as testing with other types of games and algorithms.

ACKNOWLEDGMENTS

I would like to thank my parents and brother for their friendship, encouragement, caring and patience over all these years, for always being there for me through all the challenges and without whom this project would not be possible. I would also like to acknowledge my dissertation supervisors Prof. Carlos António Roque Martinho for his insight, support, sharing of knowledge and understanding that has made this Thesis possible on various levels. Last but not least, to all my friends and colleagues that helped me grow as a person and were always there for me during the good and bad times in my life. To each and every one of you – Thank you.

REFERENCES

[1] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2012. The Arcade Learning Environment: An Evaluation Platform for General Agents. *CoRR* abs/1207.4708 (2012). http://arxiv.org/abs/1207.4708

[2] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 2015. A Neural Algorithm of Artistic Style. *CoRR* abs/1508.06576 (2015). http://arxiv.org/abs/1508.06576

[3] Michael Genesereth and Nathaniel Love. 2005. General Game Playing: Overview of the AAAI Competition. *AI Magazine* 26, 2 (2005), 62–72.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105.

[5] Kamolwan Kunanusont, Simon M. Lucas, and Diego Perez Liebana. 2017. General Video Game AI: Learning from Screen Capture. *CoRR* abs/1704.06945 (2017). http://arxiv.org/abs/1704.06945

[6] Jay Miner. 1977. Atari 2600. (1977).

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR* abs/1312.5602 (2013). http://arxiv.org/abs/1312.5602

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[9] Diego Perez-Liebana, Jialin Liu, Ahmed Khalifa, Raluca D. Gaina, Julian Togelius, and Simon M. Lucas. 2018. General Video Game AI: a Multi-Track Framework for Evaluating Agents, Games and Content Generation Algorithms. *CoRR* abs/1802.10363 (2018). http://arxiv.org/abs/1802.10363

[10] Matthias Plappert. 2016. keras-rl. https://github.com/keras-rl/keras-rl. (2016).

[11] Tom Schaul. 2013. A Video Game Description Language for Model-based or Interactive Learning. In *Proceedings of the IEEE Conference on Computational Intelligence in Games*. IEEE Press.

[12] Kevin Wong. Last accessed 19 Dec 2018. StarCraft 2 and the quest for the highest APM. (Last accessed 19 Dec 2018). https://www.engadget.com/2014/10/24/starcraft-2-and-the-quest-for-the-highest-apm/