Bachelor Degree Project

Exploring a video game AI bot that scans and reacts to its surroundings in real-time.

Author: Rasmus Karlsson
Supervisor: Johan Hagelbäck
Semester. VT 2018
Subject: Computer Science

Abstract

The buzz surrounding artificial intelligences continues to grow. They are currently used in a wide variety of systems and appliances, such as video games, virtual personal assistants, and self-driving cars. This paper explores the possibility of a self-learning AI that can play the classic arcade game Q*BERT, using only screenshots as input. It is tested to work on several different screens sizes, and the results are collected and compared to that of a human player, as well as results from previous research. The results are fairly positive. While the AI had a hard time of matching the human player on average score, it did get close to the highest score.

Keywords: Artificial intelligence, image processing, neural network.

Table of Contents

1 Introduction

The purpose of this degree project is to explore the possibility of placing an AI bot into a video game, and seeing if it can play the game using only images of the play area, and how well it performs in comparison to the average human player. For this test, the classic arcade game Q*BERT will be used.

1.1 Background

Many computer systems make use of artificial intelligences (AI's) in order to mimic human behaviour. "Artificial intelligence" is a very broad term and there are many different types of AI's, but at its core it is a piece of software developed to make its own decisions, in the same manner a human could [1]. They can be found in a wide array of modern technology, with an equally wide variety of importance, ranging from video game bots and virtual personal assistants like the Alexa, to self-driving cars. Giant companies such as Google and Tesla are very interested in AI's and are currently investing millions of dollars into the development of reliable AI's [2]. Which is not surprising because they can be very powerful programs, and many theorize that with the right research they can potentially replace humans in many simple tasks [3].

Google are especially interested in so-called "self-learning AI's". An AI that, from its conception does not know much, but through trial and error, it can learn progressively more and then use those experiences to calculate the best course of action. When it comes to systems like self-driving cars it is especially crucial that the AI can calculate the best and safest course of action, because a single mistake out on the roads could lead to a crash. These AI's need to look at the environment they are in, for instance the road, traffic signs, pedestrians, and other cars, and based on that information decide on what the correct decision is.

For AI's that needs image-processing and recognition, which is something that this project requires, a neural network can be used [4]. The neural network system was initially inspired from the human brain, and they work by first learning from processing a large sample of correct data [5]. In more practical terms, if a neural network is trained using a set of images depicting a cat or a dog it should in theory be able to differentiate between them, and correctly label them.

The video game that will be used for this project is the classic arcade game Q*BERT. In Q*BERT the objective is to turn every cube in a pyramid shaped map into a different colour by jumping on them, while also avoiding enemies. Points are accumulated by jumping on every single cube in the pyramid, shown in the image below, figure 1.1.

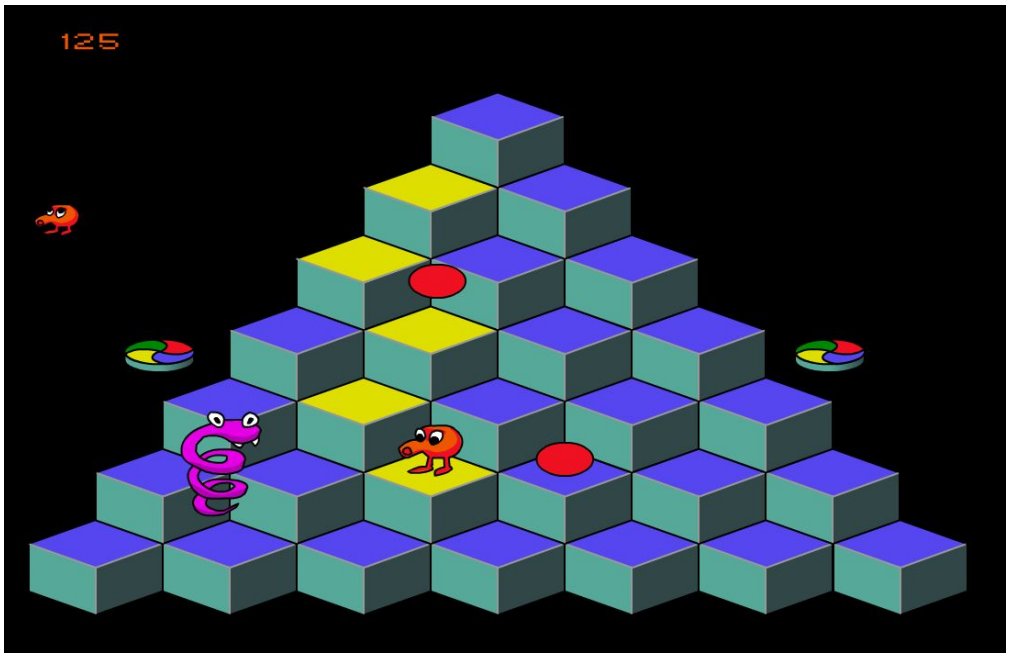

Figure 1.1: A game of Q*BERT in progress.

1.2 Related work

There has been some research and previous related work made on the subject of AI's in video games, or at least similar subjects. Perhaps the most successful recent attempt of teaching an AI to play games is the DeepMind Technology [6], which later lead the company to be acquired by Google. Their paper introduced a new deep learning model which can successfully learn how to play a number of games using only pixels as input. They focused on classic Atari games and achieved better performances than any expert human player could on some games. However, on certain games they could not reach anywhere near that level of performance, including Q*BERT, might be partly due to the fact that they applied their AI to a broad selection of games without adjusting any parameters of the AI.

In a paper published by Guzdial, Li, and Riedl [7], they describe how

their AI can play the first level in Super Mario by parsing the video output and letting the AI make decisions based on where it is and where the enemies are on the screen. This way it can learn how to play the game and finish the level. However they stated that it was only trained for the very first level, so it might not perform as well on other levels. It also cannot handle certain transitions. Therefore a goal for this paper is to make it work for any level or transition.

Part of this project will be to scan and process the game's surroundings into a usable code-based dataset, i.e. image processing. There are also examples of previous work that explores this subject. As mentioned above Google are very interested in AI development. An article by Will Knight showcases their new self-training software called Cloud AutoML [8]. A user-friendly AI that can scan images and find things it recognizes.

A paper that is related to real-time image recognition is one by authors Hoernig, Herrmann, and Radig, where they present an approach for detecting players and field lines in football videos [9]. Their approach reached an accuracy of 97.9%. It is done by taking every frame of the video output and running it through an algorithm, checking its colour values, similarly to what will be done in this project. A further goal is to make the AI use the data to successfully play the game.

1.3 Problem formulation

The goal of this project is to create a video game bot AI that scans the game's virtual environment to avoid enemies and complete the level(s). The point of this is to simulate and show how for example a self-driving car's AI behaves, by scanning its surroundings to avoid collisions and reach its destination. A video game is a very cost-friendly and risk-free environment to run this experiment in. The video game in question that will be used for this project is Q*BERT[1], as mentioned in the background. The reason this game was chosen is because it is relatively simple and it has a fixed map, meaning the layout of the level never changes. This makes the game easier to base the AI on and plausible to complete the project in the given timeframe, since it does not have to continuously recalculate the playing field. It is also a game that gave the DeepMind project some problem as it did not reach the same level of performance as other games they tested on, as mentioned in chapter 1.2. So

[1] https://en.wikipedia.org/wiki/Q*bert

that is something that will also be looked a little closer at. Another area of interest that has been noted based on the previous research is to make the AI equipped to handle any level and any transition the game has to offer.

By the end of the project an AI will hopefully have been made that can play the game competently, and also learn and document how to make an AI that successfully adapts in real-time. The AI's performance will be measured to see how it compares against a human player. Since the game's environment will be scanned by reading from the screen, it could also be interesting to run tests on how well it performs on different screens and resolutions.

1.4 Motivation

Writing a smart and reliable AI is usually very difficult. Especially when you hook it up to the unpredictable outside sources of the real world. Even with the intense focus on tests and safety that these companies put in their programs [10] people still treat it with huge skepticism, which is something that will likely take years to overcome. But these types of AI's have been present in the video game world for years. And since a computer is much faster at calculations than a person they have the potential of being better than a human at most given games, and in the case of turn based strategy games they are almost always better [11]. In fact, the only reason you can beat a computer at a game of chess for instance, is because they are designed to lose occasionally.

The goal of this project is to show that even if you put an AI in an somewhat unfamiliar environment, it can scan its surroundings and behave accordingly, possibly on the same level as a human player, even if it is on a very small scale.

1.5 Objectives

The project will be to place a bot in a video game and see if it can play the game, initially knowing only how the level looks. The preliminary objectives therefore looks like this:

O1	Implement a way to convert the level into a usable dataset.
O2	Implement bot AI.
O3	Test the AI by letting it play the game.

O4	Fine-tune bot if necessary.
O5	Run tests on screen size, resolution, etc.

The experiment is deemed as a success if the implemented AI can navigate through the course completing the game's objectives (eg. beat a level). Depending on how well the AI is implemented it should be able to perform as well as, or at least have a better reaction time, than a human player. Hopefully there is also something to learn from running the tests on different screens.

1.6 Scope/Limitation

It would be interesting to conduct the experiments on other games but since the making of AI's can be quite time-consuming it will only be run on one game, namely Q*BERT. As mentioned above this game is ideal because of its simplicity and the fact that it has a fixed map, which makes it easier to transform into usable data, and a more reasonable scope for the available time-frame.

There is little point in testing every possible screen resolution size possible, so instead only a few screens of varying sizes will be tested.

1.7 Target group

The target group for this report is mostly those interested in the development of artificial intelligences. Those looking to write their own AI's would find this particularly interesting.

1.8 Outline

The paper is structured and divided into several major parts, with various subparts to each chapter. First is the method, which explains how the project's problem will be solved or tested. The next chapter is the implementation where the software developed for the project is described. After that comes the results which simply shows the statistical output from the experiments. Finally there is the analysis followed by a discussion where the results will be further analyzed and evaluated to determine if the project was a success.

2 Method

This chapter is about the method of the paper. It focuses on how the goals of the project will be reached, and also discusses the reliability and validity of the chosen approach.

2.1 Approach

The method for this paper is a controlled experiment. The experiment will consist of running tests on the created AI to see if it can successfully play a game of Q*BERT. There will naturally be a lot of set-up required before these tests can be run. First and foremost the AI needs to be constructed, which is something that will take up much of the available time. Finding a way to give the AI the ability to scan the game's playing field is also part of the required set-up. It also needs to be able to scan the level for enemies, or otherwise unsafe platforms so it can know where to go and what to avoid. This can be done using image processing to recognize where the platforms are in relation to Q*Bert and where the enemies are by continuously processing a few frames every second. The number of processed frames per second can be modified and fine-tuned for optimal results and performance. These frames are simply screenshots taken of the game window.

Then it is the matter of teaching the AI how to play. This can be accomplished in several different ways. For example, one could define a set of rules that the AI needs to adhere to, for instance if it reaches the end of a platform it should stop and go the other way. The problem with this is that it is very difficult to cover every single scenario the AI could run into. Another option is to use reinforcement learning. Reinforcement learning is the act of letting the AI repeatedly try the game out for itself, rewarding it if it does good, and punishing it if it does badly [12]. Eventually it will learn what the ideal route to take is for most scenarios. For this project a combination of both options will be used to reach an efficient level of playing as quickly as possible. Certain simple rules will be implemented, like move away from enemies and do not jump off the edges. but most else will be covered with reinforcement learning. So on top of the game rules a self-learning neural network will be implemented. It will work by letting the AI automatically and continuously play games of Q*BERT, collecting data for the neural network to train on. If the AI does well, meaning if it surpasses a specified minimum score, it will save the direction it took for every frame for that game. The AI

is re-trained based on the new neural network and the process is repeated. Theoretically the bot should become better and better like a self-learning AI.

2.1.1 Tools and Packages

The programming language that will be used to create the AI and the image processing functionality will be Python version 3.5[2]. This language was chosen because it is generally very good for writing scripts and it has a lot of packages useful for this type of program. Important packages and modules used can be found below:

- PyAutoGUI[3]
- Image from PIL[4]
- Python MSS[5]
- NumPy[6]
- Tensorflow[7]
- TFLearn[8]

2.2 Experiment Data

The project will be deemed a success or not based on the data gained by the experiment. After the AI has been sufficiently trained it will be evaluated and compared to human scores. Below is the list of things that will be measured:

- Average levels completed
- Most levels completed
- Average score
- Highest score
- Time passed

Since it is a rather simple game there is not much else to keep track of. The project will be deemed a success if the bot can reach, or even surpass, the scores of a human player. This is something that the DeepMind project did not achieve for Q*BERT so any improvement over their results would be very positive.

---

[2] https://docs.python.org/3/
[3] http://pyautogui.readthedocs.io/en/latest/
[4] https://pillow.readthedocs.io/en/3.1.x/reference/Image.html
[5] http://python-mss.readthedocs.io/index.html
[6] http://www.numpy.org/
[7] https://www.tensorflow.org/api_docs/python/
[8] http://tflearn.org/

Another thing that will also be tested is if the monitor screen resolution and size makes any difference to the AI's performance. This will be a completely separate test from the experiment above, as it will only compare the AI with itself on different screens. The same things will be measured but on this test reaction time will also be taken into account. Since testing every conceivable screen size is impossible, four screens of varying sizes have been selected. They can be found in the table below, table 2.1.

Name	**Resolution**	**Size**	**Type**
Asus VN247H	1920x1080	23.6"	Monitor
Dell 1905FP	1280x1024	19"	Monitor
HP ProBook 430 G2	1280x720	13.3"	Laptop
Toshiba 40L1353N	1920x1080	40"	TV

Table 2.1: Selected screens with specs.

2.3 Reliability and Validity

In theory if anyone tries to recreate the experiment with the same prerequisites they would get the same or at least similar results. They would obviously need the same programs and AI as they could change the results quite drastically if they are not implemented in the same way. Another thing that might prove to be an issue is how the AI learns how to play the game using the reinforcement learning process, which could vary a bit depending on in what order it ends up attempting scenarios. However it should theoretically end up with a similar level of skill eventually.

Any data from the experiment, like score and time, will be taken directly from the game or program automatically. However, one issue is that these results could differ a lot depending on how far the AI or player makes it in the game. Therefore an average will be calculated, as well as the best score either party achieves while playing the game to ensure the results are as fair and accurate as possible.

The validity of this paper has also been taken into consideration. Since the performance results for the AI will be compared to the results from human players it is important that the results from the latter are really representative of the average player. The current world record score for

Q*BERT, set in 2013, is 37.163.080, which took almost 85 hours to achieve [13]. This is not a reasonable goal for the AI to reach in the time-span of this project. Instead it will focus on reaching the ability level of a more average human player. This is something that is very hard to get an exact number on so there will have to be some estimation in this aspect, which could skew the validity slightly.

3 Implementation

To complete this project there needed to be an application made specifically for the task. This chapter will explain how the application was implemented and will describe it on a slightly more technical level.

3.1 Scanning play area

In order to make any progress the application needs to make an initial scan on the play area, and translate it into a data structure it can use. It needs a way to see where on the screen the platforms are so that it can, by extension, see where the player is and on what platform. If the bot only needed to be run on a single screen this could be done easily by manually taking the correct pixel measurements and feeding them to the application, but in this case that is not enough, since several screens and resolutions are involved.

The application starts by taking a screenshot of the entire game window using PIL's ImageGrab. From there it uses a reference picture, a so called template, of a single platform box to match it with the screenshot, drawing a rectangle wherever it finds a match for the platform template. The rectangle is moved up slightly so that it covers the platform. The end result looks something like in figure 3.1. This process is done using OpenCV.

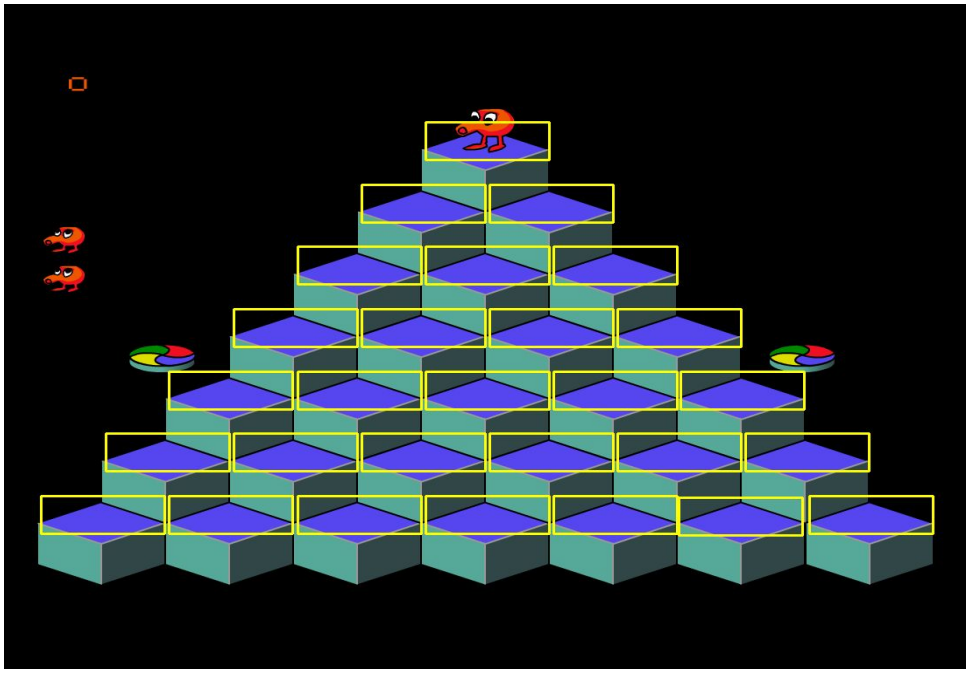

Figure 3.1: Initial scan of play area.

The rectangles are actually invisible, the image is just for illustration purposes. The pixel measurements of the rectangles are then saved in numbered objects instantiated from a Platform class. Since the scan is performed top-to-bottom, left-to-right, the order always stays the same, with the very top platform being 0, the two platforms on the row below are 1, and 2, and so on.

From there it can check if something is standing on the platform by checking for their colours. Orange means Q*Bert, purple means the enemy snake, for example. The enemies move quite fast between platforms, but the application however scans several frames a second, 6-10 fps, which is good enough in this case.

3.2 Self-learning Neural Network

The application uses the scanned data for the AI to determine where it is safe for Q*Bert to go. As mentioned in Section 2 Method, the bot is built on a combination of rules and as a self-learning AI. The rules are a good baseline to base the self-learning AI on.

3.2.1 Rules

The rules are things that are very simple and are used to complement the self-learning AI.

These rules include:

- Do not jump off the outer platforms.
- Do not jump to a platform occupied by an enemy.
- Do not jump to a platform that an enemy can reach in a single jump (unless there are no other options.)

3.2.2 Self-Learning Neural Network

The application uses TFLearn, a library built on top of Tensorflow, to implement the neural network. It is simple in theory. First it needs a very large amount of data containing “correct” moves a player has made, which it then uses as reference for finding the best move the bot can make. To collect this data the application was set to save the screenshot and direction of the move every time a move was made. If it eventually reached a score higher than the average of the games already played the screenshot and corresponding move would be added to the dataset. The screenshots were resized to 160x120 to save disc space and converted into greyscale to make it

more ambiguous regarding what level it was on (different levels have different colours.)

Collecting a data sample that would work well enough would take hours of playing which would be too monotonous and time consuming for a single person. That is why the set of rules defined above was important because the bot could play and collect the data automatically without supervision. The bot with those rules does an acceptable job of simply staying alive on its own, but it struggles with actually completing the levels. This is something that the self-learning neural network helps with.

3.2.3 Primary Loop

Below is a very simplified snippet of code from the loop that does the brunt of the work in the program, code 3.1.

```
while(True):
    img = takeScreenshot(dimensions)
    screen = np.array(img)

    #Check if game over
    if checkGameOver(screen) == True:
        clickNewGame(dimensions)
        img = takeScreenshot(dimensions)
        updateLevel(platforms, img)
        continue

    direct = calcPlay(platforms, img)
```

Code 3.1: Main loop of program.

It starts by taking a screenshot of the game area, which dimensions are identified earlier. Then it converts the image to a numpy array to make it easier to scan for a game over screen. If a game over screen is found it automatically starts a new game and updated the level, since the program needs to keep track of which level the bot is on. Finally the direction Q*Bert should jump is calculated in the calcPlay function, which is where most of the logic in the application resides.

4 Results

The results are separated into three categories for each of the five entities that plays the game. The entities consists of the AI bot on the four different screens in addition to the human player. The latter may be slightly more ambiguous than the others because of a smaller sample pool. The AI could simply be left running to automatically collect a much larger amount of data.

The bot screens are numbered as follows:

Bot Screen 1: Asus VN247H (1920x1080, 23.6”)
Bot Screen 2: Dell 1905FP (1280x1024, 19”)
Bot Screen 3: HP ProBook 430 G2 (1280x720, 13.3”)
Bot Screen 4: Toshiba 40L1353N (1920x1080, 40”)

Player	**Highest Score**	**Average Score**	**Average Time Alive**
Bot Screen 1	6250	2452	72s
Bot Screen 2	8450	2877	81s
Bot Screen 3	6525	2692	67s
Bot Screen 4	4325	1867	39s
Human Player	11550	5893	93s

Table 4.1: Raw data from around 500 games of Q*BERT each (less for human player)

Player	**Level Progression For HS**	**Level Progression For Avg**
Bot Screen 1	4.58	3.06
Bot Screen 2	5.58	3.39
Bot Screen 3	4.96	3.22
Bot Screen 4	3.96	2.74
Human Player	6.79	4.06

Table 4.2: The level progression based on the raw data in table 4.1. The first number stands for the level and the decimals shows the progress into that level. For example a score of “2.50” would be exactly halfway into level 2.

4.1 Score Graph

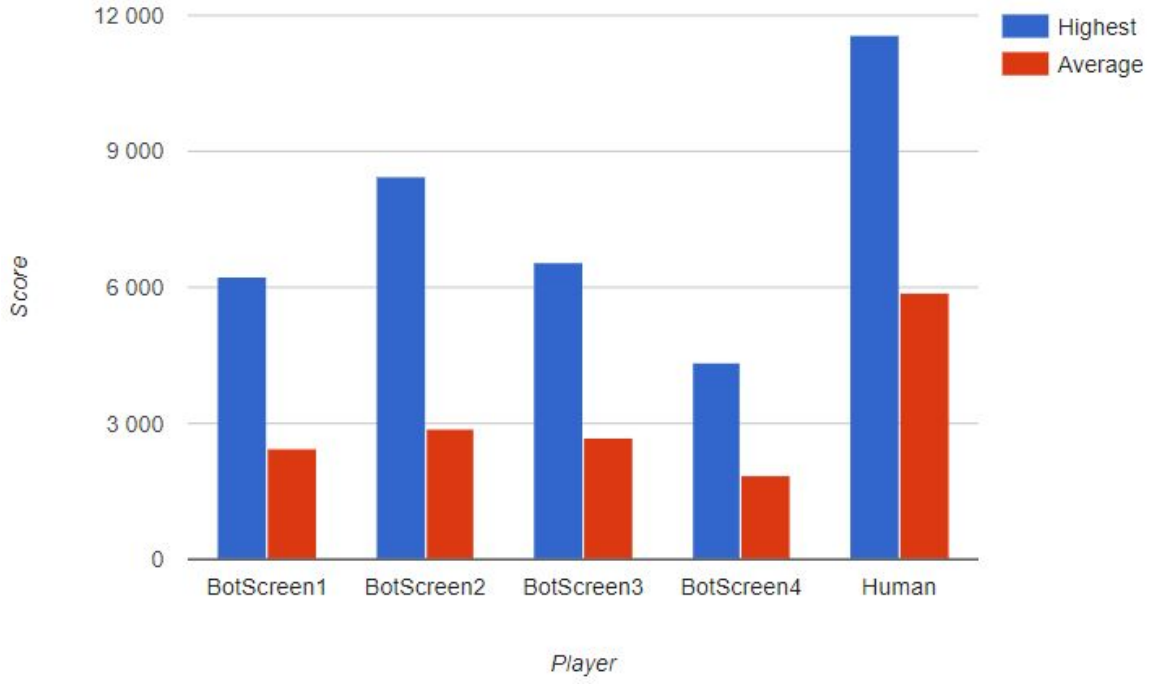

Figure 4.1: Bar graph depicting the highest and average scores for each player.

4.2 Level Progression Graph

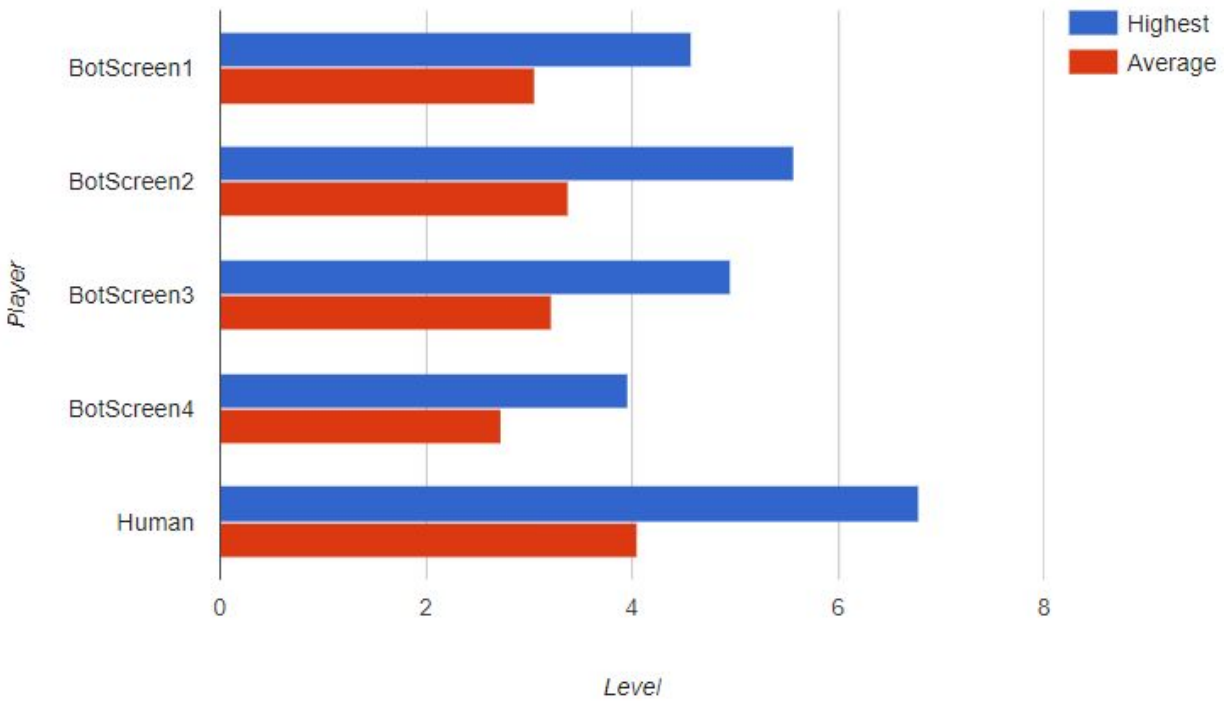

Figure 4.2: Bar graph depicting the level progression for both the highest and average score for each player.

4.3 Time Alive Graph

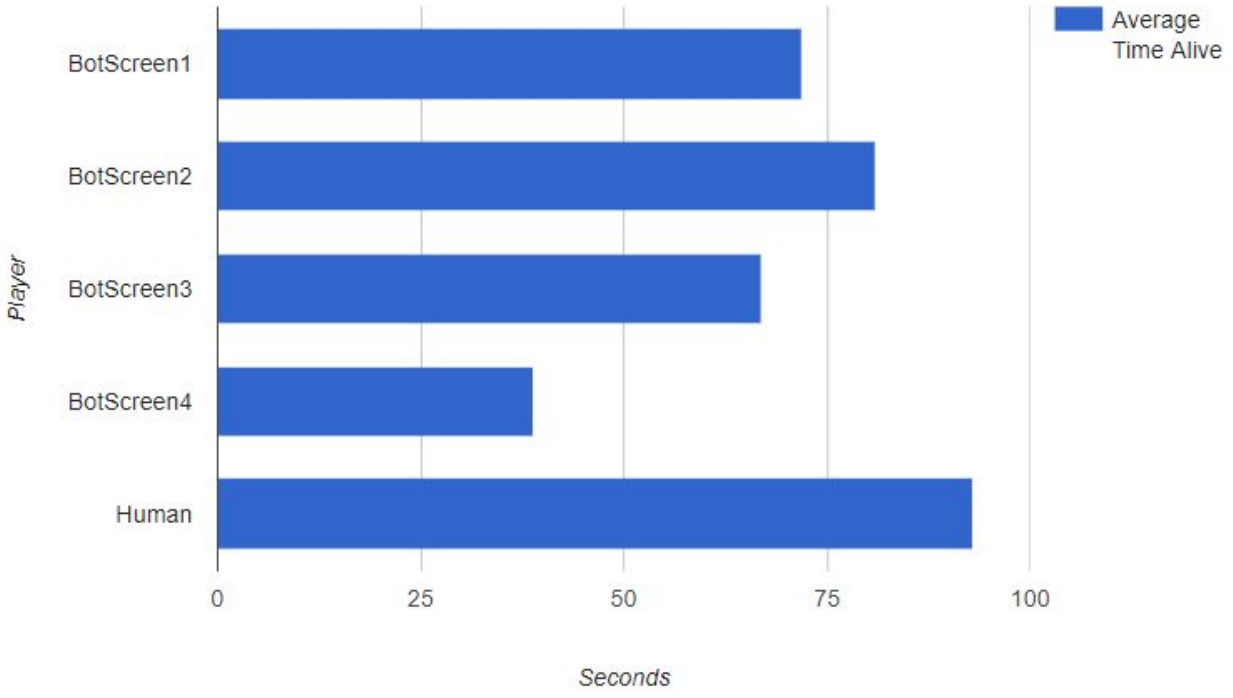

Figure 4.3: Bar graph depicting the average time alive for each player.

5 Analysis

As the data shows the AI bot can beat at least a few levels, which was the goal of the experiment. Another goal was to see if the bot is comparable to a human player. The experiment results suggests that the bot has a decently high ceiling for accumulating points, beating several levels, and even reaching the vicinity of the human player but not quite enough. The human player is also far better on average, which is logical since the AI of the bot has some elements of randomness to it. For example if it finds multiple good moves it can make it chooses one of them at random. The enemies also move in a fairly sporadic pattern, which is hard for the AI to predict. There are a few other wrinkles in the program that can sometimes make the bot reach an early game over, which brings down the average score.

As for the different screens, it seems like there is a slight difference where the bot performs a bit better on the smaller screen resolutions. The screen that the bot performed by far the worst on was the largest one. Since the bot's main function is scanning the screen and calculating the best move based on the resulting image, it makes sense that a smaller screen results in a higher performance rate. The program repeats the process as quickly as it possibly can, and therefore with more screenshots scanned there is a better chance that the prediction is correct. It should be noted that the highest score for screen 2 is slightly misleading because after every level you receive a bonus 1000+ points, and on the highest scored run for screen 3 it was only one platform from reaching the same level as screen 2 in this case.

Another interesting thing to note is that despite the human player having a much higher score, the average time spent alive is a lot closer. This is probably because of the fact that a human player will focus on traversing towards the platforms they have not yet been on to complete the level as quickly as possible, while the AI prioritizes avoiding enemies, which would make the bot spend a longer period of time on each level.

6 Discussion

The question this project was aiming to answer/test was, how well can an AI perform, that reacts to its surroundings with a limited amount of previous knowledge, and how does it compare to a human that does the same thing? Based on that initial problem formulation the project was a success since the question was answered. The AI did not perform any better compared to a human player which was a little disappointing. I believe the reason for this was because the self-learning part of the project did not improve the AI as much as I would have hoped. Perhaps it would have improved more if I had let it collect an even larger data pool but unfortunately the project time was running out.

The results of the project suggest a similarity to the results from the DeepMind project, where the AI they used on Q*BERT also could not reach the performance of a human player, while many of the other games they tested the AI on were much more successful [6]. Their reasoning for this was that games like Q*BERT are more challenging for an AI because it requires a strategy over a long period of time. Their AI reached a score of 4500 as their best for Q*BERT while this project managed 8450, with the average score also being better although not by that much. It is possible that there is a difference in point systems between the different versions of Q*BERT used in the two projects since this project used a remake of the game.

7 Conclusion

The project set out to make an AI bot for the classic arcade game Q*BERT that played the game based on the raw pixel output from the screen. The AI was built on a combination of a set of rules the bot should adhere to, and a self-learning neural network. The report shows the results of the experiment which was conducted on the bot with different screens. This experiment suggests that while it may not be quite at the level of a human player, it can at least beat several levels of the game before getting a game over. It also showed that it performed slightly better on screens with a smaller resolution.

It is not unreasonable to assume that a similar AI would be possible to implement for more or less any game, program, or other GUI screen based application. Of course, there would need to be a lot of changes since this AI is very specific to Q*BERT, but the basic principle should remain the same.

The self-learning neural network did slightly improve the performance of the bot but did not improve it as much as I had originally thought. Perhaps it could be made a bit better by optimizing it. The data it collected and learned from may not have been large enough, so letting it collect more data would possibly be one way to improve it.

7.1 Future work

While the AI plays the game just fine, there are some imperfections with it that could be remedied with a little more fine-tuning. For example, it seems to often get stuck in the two bottom corners, jumping back and forth between two platforms. Improving the self-learning part would also be interesting, maybe by letting it continue to collect a larger/better data sample.

References

[1] Artificial Intelligence.
https://en.wikipedia.org/wiki/Artificial_intelligence
URL last visited on 2018-02-05

[2] The Economist. (7th Dec 2017) "Google leads in the race to dominate artificial intelligence." [Online]. Available:
https://www.economist.com/news/business/21732125-tech-giants-are-investing-billions-transformative-technology-google-leads-race

[3] H. Huang, R. Rust, "Artificial Intelligence in Service", Feb 2018.

[4] D.M. Pelt, J.A. Sethian, "A mixed-slace dense convolutional neural network for image analysis", Nov 2017.

[5] M. van Gerven, S. Bohta, "Artificial Neural Networks as Models of Neural Information Processing", Dec 2017.

[6] V. Mnih, K. Kavukcuoglu, D.Silver, A.Graves, I.Antonoglou, D.Wierstra,
M. Riedmiller, "Playing Atari with Deep Reinforcement Learning", DeepMind Technologies, Dec 2013.

[7] M. Guzdial, B. Li, M.O. Riedl, "Game Engine Learning From Video", School of Interactive Computing, Georgia Institute of Technology, Sep 2017

[8] W. Knight, MIT Technology Review. (17th Jan 2018) "Google's self-training AI turns coders into machine-learning masters" [Online]. Available:
https://www.technologyreview.com/s/609996/googles-self-training-ai-turns-coders-into-machine-learning-masters/

[9] M. Hoernig, M. Herrmann, B. Radig, "Real-Time Segmentation Methods for Monocular Soccer Videos" (3rd Sep 2014)

[10] Vinton G. Cerf. “A Comprehensive Self-Driving Car Test” (February 2018) in *Communications of the ACM,* Vol 61, page 7.

[11] M. Blitz, Popular Mechanics. (14th March 2016) “Checkmate, Human: How Computers Got So Good at Chess.” [Online]. Available: https://www.popularmechanics.com/technology/a19914/chess-computers/

[12] P. Vranx, *Decentralised Reinforcement Learning in Markov Games*, pp. 1-6, March 2010.

[13] G.Levy, TIME. (20th Feb 2013) “Man Breaks World Record by Playing Q*Bert for 84 hours.” [Online]. Available: http://newsfeed.time.com/2013/02/20/game-finally-over-man-breaks-world-record-by-playing-qbert-for-84-hours/