Machine Learning to identify cheaters in online games

Martin Willman

May 18, 2020
Master’s Thesis in Interaction and Design, 30 credits
Supervisor at TFE-UmU: Leonid Freidovich
Examiner: Thomas Mejtoft

UMEÅ UNIVERSITY
DEPARTMENT OF APPLIED PHYSICS AND ELECTRONICS
SE-901 87 UMEÅ
SWEDEN
Abstract

Cheating in online games is a problem both on the esport stage and in the gaming community. When a player cheats, the competitors do not compete on the same terms anymore and this becomes a major problem when high price pools are involved in online games. In this master thesis, a machine learning approach will be developed and tested to try to identify cheaters in the first-person shooter game Counter-Strike: Global Offensive. The thesis will also go through how the game Counter-Strike: Global Offensive works, give examples of anti-cheat softwares that exists, analyse different cheats in the game, consider social aspects of cheating in online games, and give an introduction to machine learning. The machine learning approach was done by creating and evaluating a recurrent neural network with data from games played with the cheat aimbot and without the cheat aimbot. The recurrent neural network that was created in this master thesis should be considered as the first step towards creating a reliable anti-cheat machine learning algorithm. To possible increase the result of the recurrent neural network, more data and more data points from the game would be needed.
Contents

1 Introduction .................................................................................................................... 1
   1.1 Objective .................................................................................................................. 2
   1.2 Problem Statement ............................................................................................... 2
   1.3 Source Empire AB ................................................................................................. 2
   1.4 Limitations ............................................................................................................ 2
   1.5 Ethics ..................................................................................................................... 2

2 Background .................................................................................................................... 3
   2.1 Brawl Gaming ........................................................................................................ 3
   2.2 Counter-Strike: Global Offensive ........................................................................... 3
   2.3 How to Cheat & Types of cheats ............................................................................ 3
      2.3.1 Exploits ............................................................................................................ 4
      2.3.2 Automation ...................................................................................................... 4
      2.3.3 Overlay ............................................................................................................ 4
      2.3.4 State Manipulation ......................................................................................... 5
   2.4 Anti-cheat software ............................................................................................... 5
      2.4.1 VAC ................................................................................................................ 5
      2.4.2 EAC ................................................................................................................ 5
      2.4.3 Summary anti-cheat software ........................................................................ 5

3 Theoretical framework ................................................................................................. 7
   3.1 Artificial Intelligence .............................................................................................. 7
   3.2 Machine learning .................................................................................................... 7
      3.2.1 Supervised learning ....................................................................................... 8
      3.2.2 Unsupervised learning .................................................................................. 8
      3.2.3 Reinforcement learning ................................................................................ 9
      3.2.4 Artificial Neural network ............................................................................. 9
   3.3 Game Physics .......................................................................................................... 11
   3.4 Social aspects of Cheating in online games .......................................................... 11
   3.5 Related Work ......................................................................................................... 12
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Method</td>
<td>4.1 Literature Study</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>4.2 Programming Languages, Virtual environments &amp; libraries</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>4.3 Learning Algorithm</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>4.4 Development</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>4.4.1 Data gathering</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>4.4.2 Building the Model</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>4.4.3 Train Model</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>4.5 Test Model</td>
<td>17</td>
</tr>
<tr>
<td>5 Results</td>
<td>5.1 Results from Input data</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>5.2 Building the Recurrent Neural Network</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>5.3 Testing the Recurrent Neural Network</td>
<td>21</td>
</tr>
<tr>
<td>6 Discussion</td>
<td>6.1 Input data</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>6.2 Building and Testing the Recurrent Neural Network</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>6.3 Social aspects on cheating in online games</td>
<td>24</td>
</tr>
<tr>
<td>7 Conclusions</td>
<td>7.1 Future work</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>7.1.1 Identify cheaters in online games</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>7.1.2 More cheat detection</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>7.1.3 Further use cases for other games</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>7.1.4 New cheat</td>
<td>26</td>
</tr>
<tr>
<td>8 Acknowledgements</td>
<td></td>
<td>27</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>29</td>
</tr>
<tr>
<td>A Source Code</td>
<td></td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>A.1 Recurrent Neural Network</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>A.2 Code for Gathering the input data</td>
<td>33</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Computer games have been around since the early 1950s. The first known computer game was "XOX", developed in 1952 by A.S.Douglas. In the beginning, computer games were only accessible on Universities, but in the early 1970s, computer games become more commercial. One of the big companies, at that time, was Atari. Atari has created lots of big games such as Pong, Trak 10, and Gotcha [1].

Today the landscape has changed and now almost everyone plays games, either on their smartphone, console, or computer. The number of gamers has gone from a few people back in the 1950s to over 2.5 billion in 2016 [2]. As everyone plays computer games, some will try to make it for a living.

Esports, electronic sports, is defined as “sport activities in which people develop and train mental or physical abilities in the use of information and communication technologies.” [3]. The term esport was first heard of in the late nineties and the first reliable source which used the term were Online Gamers Association (OGA) in a press release in 1999. Since then, the esport market revenue has grown rapidly and is estimated to reach more than one billion dollars in 2019 [4]. The big esports games today are Fortnite, Counter-Strike: Global Offensive, Dota 2, League of Legends, and PlayerUnknown’s Battlegrounds. In esports tournaments, the prize pools can reach up to 30 million dollars awarded to the winners. With such high prize pools, some are willing to use forbidden tools to help them get an advantage in the game. There have been some big scandals in the esports community. Valve, who has developed Counter-Strike: Global Offensive, has a list of over 100 professional gamers who are banned from the big tournaments and leagues. They are banned for cheating, matchfixing, account sharing, and more [5].

Cheating in games is defined as “Any behaviour that a player may use to get an unfair advantage, or achieve a target that he is not supposed to is cheating” [6]. There is a range of different types of cheats in online games, but they can be divided into four bigger categories; exploits, automation, overlay, and state manipulation [7]. According to a survey made by around 9500 online gamers worldwide, it was found that 37% of them are cheating [8].

To find these cheaters, modern technology such as artificial intelligence (AI) or more specifically the application field machine learning (ML) can be used to create systems that identify these cheaters. To create a machine learning system that identifies cheaters, large data sets need to be collected. That data set could contain information about where the user is aiming and if the user is aiming at an enemy. This master thesis will focus towards trying to identify cheaters using the cheat aiming robot (aimbot) which is a cheat that helps the user track other targets by controlling their aim [9].
Chapter 1. Introduction

1.1 Objective

The objective of the thesis is to develop and test a machine-learning algorithm that uses deep learning to identify cheaters in first-person shooter games. More specifically, a Counter-Strike: Global Offensive game server will be made that logs how a player is aiming and if the player is aiming at an opponent. The machine learning algorithm will then read the file and identify if the player is using the cheat aimbot or not.

1.2 Problem Statement

The number one question Brawl Gaming users are asking is “How do you prevent cheating?”. The community and Brawl Gaming users does not trust the current anti-cheat softwares, such as Valve Anti-Cheat or Easy Anit-Cheat. With that in mind and the fact that online gamers often are unsupervised when playing games, it makes it easy for players who are willing to cheat to get an advantage over other players. Therefore, a decision was made to evaluate the possibilities of discovering a solution to this problem which can be used by esport contest managers, server moderators or others that would need this to stop cheaters.

1.3 Source Empire AB

The writer of this thesis is CFO and Co-Founder of Source Empire AB. Source Empire AB is a start-up company created by four Interaction & Design students at Umeå University and started in spring 2018. The company is developing an esport platform called Brawl Gaming, which you could read more about in section 2.1.

1.4 Limitations

This master thesis paper will be limited to the game Counter-Strike: Global Offensive. The system will not be implemented into the game, and it is more like a third-party application that could be used to help server moderators to identify possible cheaters.

1.5 Ethics

Brawl Gaming is an esport platform where ordinary gamers could bet on their own performance while playing big esport games and some ethics need to be brought up. This master thesis will not be about the esport platform or the money management. Instead, it will be all about how to develop a machine learning algorithm to identify potential cheaters in Counter-Strike: Global Offensive. Since cheaters are a problem in the community, every attempt to stop these cheaters while playing competitive esport games is seen as the right thing to do and it is something the author behind this master thesis agrees with.
Chapter 2

Background

This section will give some background information about the product Brawl Gaming. It will also give a background on what Counter-Strike: Global Offensive is, which type of cheats exists in the game, and what type of anti-cheat softwares can be used.

2.1 Brawl Gaming

Brawl Gaming is an esport platform, created by Source Empire AB, that enables gamers to play performance-based online games for money. When a bet is made, a matchmaking starts to find an opponent in the same game and with the same bet. The bet from each player is going to a pot. After the game is finished the winner gets the whole pot. The business model is based on that the users compete in big esport games such as Counter Strike and Dota 2. In all of these games cheats and cheaters exist. Therefore, this project will focus on creating a machine learning algorithm to identify cheaters in Counter-Strike: Global Offensive to make users compete on the same terms.

2.2 Counter-Strike: Global Offensive

Counter-Strike: Global Offensive (CS: GO) is an expansion upon the original first-person shooter game, Counter-Strike, which was launched in 1990 by Valve. The game is about two teams that compete in different round specific game modes, when a team has won enough rounds they also win the match. The different types of game modes are Competitive, Wingman, and Casual. Competitive is the most classic and common game mode and involves two teams of five persons each competing to first get 16 winning rounds in either Hostage Rescue or Bomb Defusal game modes [10].

2.3 How to Cheat & Types of cheats

Cheating in online games has been a problem for many years and will remain to be for many years to come. Due to the more secure anti-cheat services, the cheats have become more advanced. CS: GO and many more online games have a client-server architecture. Client-server architecture is used for receiving a low latency. The client is sending the users input to the server, the server then reads the user inputs from the client and executes them.
The client also render scenes and predict both how other users are moving and how the shots are fired [11].

The client-server architecture is, as mentioned, used for receiving a low latency, which is crucial in most online games, but it also allows developers and gamers to create cheats. Any data that the user is sending to the server could be tempered data. The more responsibility the client have, the more could be compromised or hacked. There is a range of different types of cheats or hacks for different games. This range could be divided in four categories: exploits, automation, overlay, and state manipulation.

2.3.1 Exploits

Exploit is the first level of cheats and regards gamers who take advantage of example pixel-bugs, which gives them an upper hand over other players. These types of cheats are not that hard to fix or not that big of a problem, except in big matches or tournaments. Exploits are often design flaws, which are created by the game developers [7].

- **Pixel-bug** - User is taking advantage of design flaw for gaining more information than the user should have.

2.3.2 Automation

Automation is when a users have a script or binded keys, which automate user actions. This is in some games allowed but in other games forbidden. Examples of cheats in the automation category is aimbot and triggerBot [7].

- **Aimbot** - Aimbot, aim assistant, aiming robot, or automatically aim is all different names that refers to the same thing. Aimbot helps the user track other targets by controlling the players aim, and is one of the most effective ways of cheating in the first-person shooter games genre. The cheat works by retrieving state information from the game, which the other players are unable to receive. With that information, the bot then helps the cheater with its aim. By using an aimbot, the user outperform the human ability of performance in first-person shooter games [9]. In this research paper, the machine learning algorithm will be tested to identify this type of cheat.

- **TriggerBot** - A triggerBot works much like an aimbot, but the big difference is that the triggerBot does not help the cheater to aim but instead helps the cheater to shoot when the cheater looks at an opponent in the game. This gives them an advantage over other users.

2.3.3 Overlay

Overlay cheats occur when the ESP, Extrasensory Perception, give out more information to the user to correlate between objects in the game. Wallhack and radar hack are types of cheats that are using this overlay [7].

- **Wallhack** - User is able to see other players through walls or other objects. The cheat is created when the ESP give more information to the cheater then it should. With that information, the program could give the cheater visuals of all of the other players. The cheater then get an unfair advantage over all the other players.

- **Radar hack** - Radar hack works in the same principles as wallhack but instead of showing the position of the other players through walls, they display the position of the other players on the in-game radar, that most online games have.
2.3.4 State Manipulation

State Manipulation are cheats or hacks that completely change how the game is played. Cheats that fit into the State Manipulation category are speed-hack, no clipping, and some other not mentioned in this paper [7].

- **Speed-hack** - Cheater gain unfair advantage by using a speed-hack that helps the cheater by speeding up their actions. This results in that the cheater can move faster than an honest player [12].

- **No clipping** - Allows the cheater to fly and go through walls as advantages.

2.4 Anti-cheat software

There exists some anti-cheat softwares. Valve has for example created an anti-cheat software called VAC which stands for Valve anti-cheat. Some companies only specialize in anti-cheat softwares such as the Finnish company Kamu, which in 2018 was bought by Epic Games [13]. Kamu is the company behind Easy Anti-Cheat (EAC).

2.4.1 VAC

Valve Anti-Cheat or VAC is an automated anti-cheat system designed to find cheats on the users’ computer. If the VAC-system finds any cheats or any third-party applications that gives the user an unfair advantage, the system will automatically ban the user [14].

Since 2006, VAC has banned over 15 million cheaters but it is in the later years most of the bans have been made [15]. A reason for this could be that VAC in 2016 started using deep learning to catch cheaters [16].

2.4.2 EAC

EAC stands for Easy Anti-Cheat and is an anti-cheat tool developed by Kamu. The tool is used to catch cheaters in online multiplayer games. EAC was created in 2001 and started as a third-party application for Counter Strike but became an anti-cheat software for over 30 different kinds of games. According to marketing materials, EAC is using a hybrid between machine learning and driver code. Driver code checks if the user have some corrupted memory, untrusted system files, and unknown game files, among other things on the players computer.

2.4.3 Summary anti-cheat software

There are plenty of anti-cheat software and something which is common to all of them is that they do not explain how the anti-cheat software works. This is because they do not want cheaters to know exactly how they detect cheaters.

Since Brawl Gaming does not exactly know how these anti-cheat softwares works, they want to create their own to know how it works. Brawl Gaming also wants to create its own anti-cheat software to more easily ban cheating players from the esport platform.
Chapter 3

Theoretical framework

3.1 Artificial Intelligence

As mentioned in the introduction, artificial intelligence or AI is an effective way to make computers take decisions when there are massive amounts of data that need to be analyzed repeatedly and fast. The term artificial intelligence has been around for a long time but it was in the 1950s as a research team, led by John McCarthy and Marvin Minsky, initiated the artificial intelligence division of the Massachusetts Institute of Technologies. In connection with this, the official birth of AI in computer science came up [17]. John McCarthy defined AI as “the science and engineering of making intelligent machines” [18]. Since then, the definition has become a bit broader but the basis in still the same.

The field of AI is extensive and has, therefore, been divided into different branches. The different branches are Machine Learning, Deep Learning, Natural Language Processing, Robotics, Expert Systems, and Fuzzy Logic [19]. This research paper focuses on machine learning and deep learning. The other branches will not be mentioned further.

3.2 Machine learning

Machine Learning is a branch or a technology in artificial intelligence and also in the classification type of Artificial Narrow Intelligence. In this time, data is generated everywhere. For example when shopping online, checking stocks, or making a post in social media. With so much data, often called big data, many products and services have been developed and specialized to predict consumer interests. In order for a computer system to be able to predict which type of consumers buys the most frozen pizza or which consumers are most surprised and amazed over this new clothing brand, the system needs to be intelligent. To count as intelligent, the system needs to have the ability to learn and make changes depending on the environment [20].

To create an intelligent machine learning system, one first needs to have some rules on how the system should behave in reaction to the input. There are different ways to
implement these rules or algorithms for various types of projects and data collections, but for
the most parts they are divided into three bigger categories. These categories are Supervised
learning, Unsupervised learning, and Reinforcement Learning [20].

### 3.2.1 Supervised learning

Supervised learning is a type of machine learning algorithm that is used when the output
is known for a large number of samples. The aim is to learn mapping from the input to
a correct valued output, the whole process is provided by a supervisor. An example of a
supervised learning task is classification, used by an image-processing algorithm designed
to recognize images of an apple. The input is photos of all types of fruits. The algorithm
should then recognize images of apples and generate as the output that this image is an
apple and that the other images are not apples. This is a good example of supervised
learning, because humans could control what conclusions the algorithm should draw [20].

The supervised learning process is divided into three levels: build the model, train the
model, and test the model.

The first thing to do is to build the model. When implementing the algorithm, the
problem needs to be understood. Supervised learning problems could be broken down into
two types called, regression and classification. Regression is a supervised learning problem
where there is an input and an output, and the task is to learn the mapping from the input
to output. Regression is mainly used when working with numbers, such as exploratory
variables and scalars, this can be applied for weather data analysis [20]. Classification
problems are when there are two or more classes. Such as cats or dogs, apple or not an
apple, or low-risk or high-risk customers. The information about a customer is the input
to the classifier, whose task is to assign the input to one of the two classes. After training
with the past data, a learned classification rule may be given by

```plaintext
Algorithm 1 classifier high-risk/low risk customers
if income > y1 then
   if savings > y2 then
      customer ← low - risk
   end if
else
   customer ← high - risk
end if
```

After the model is build the model needs to be trained. This is where the data is needed.
When the model is build and trained the model or hypothesis needs to be tested.

### 3.2.2 Unsupervised learning

In supervised learning, the given input could be mapped to the correct output by classes or
labels, while in unsupervised learning, the given input do not have these classes or labels
which make it harder. The aim in unsupervised learning is to find regularities in the given
input to make the output more understandable and useful [20]. Since the data given in this
master thesis will be labeled, unsupervised learning will not be used.
3.2. Machine learning

3.2.3 Reinforcement learning

Reinforcement learning is a type of machine learning where the problem is faced by an agent that must learn its behavior through trial-and-error interactions with dynamic environments. The best example on how reinforcement learning could be used in the real world is when OpenAI Five learned how to play the game Dota 2. OpenAI Five is the name of a machine learning project which compete five-on-five in the game Dota 2, created by the team on OpenAI. The software is build on a reinforcement learning algorithm and uses trail and error to maximize the virtual reward. Each day, the software played equivalent of 180 years of Dota 2. In April 2019, OpenAI Five won against the best human team in Dota 2 [21][22]. Since reinforcement learning is too complex, it will not be considered in this master thesis.

3.2.4 Artificial Neural network

In the field of Artificial Intelligence there is a branch which takes their inspiration from how the neurons in the human brain work, this branch is called Artificial neural network (ANN) [20]. ANN is, through deep learning, a part of the machine learning field. Since deep learning is a part of machine learning, the broader concept of machine learning will be used in this thesis [23]. The aim of an artificial neural network is not to create a replica of the human brain, instead to create useful machines that process data as the human brain. Artificial neural networks are algorithms that are constructed and designed to recognize patterns in data by using artificial neurons or nodes. The neural network needs to have inputs, weights, processing function, and an activation function and from that the neural network will give an output. This is illustrated in Figure (3.1). In the figure, the inputs (red circles) are called the input layer, processing function (blue circle) and activation function (yellow circle) are called the hidden layer, and the output (green circle) is called the output layer. In more complex neural networks there could be more hidden layers. Applications where data patterns can be recognized are in sounds, images, time series or text. These type of applications could either use supervised or unsupervised learning [20][24].

Convolutional neural network

Convolutional neural network or CNN is a neural network used primarily to classify images, object detection in videos or cluster images by similarity. For example, convolutional neural networks are used to help doctors identify brain tumors or to classify street signs [25][26].

When a convolutional neural network perceives images, they do not perceive it like humans do. Therefore, images need to be seen as a feed of three-dimensional objects rather than a flat canvas with a given width and height. Since digital color images have a red, blue, and green (RGB) encoding, mixing those three colors will give the color spectrum for a human. Instead of mixing the colors, a convolutional neural network sees images as three separate layers of colors stacked one on top of the other. A convolutional neural network
receives an image as a rectangular box whose width and height are given by the number of pixels along those dimensions, one for each of the three layers of the colors RGB. When an image move through a convolutional neural network the image will be expressed as a mathematical matrix of multiple dimensions, such as 50x50x5. Rather than just focus on one pixel at the time, the convolutional neural network takes in a two-dimensional matrix of pixels and passes them through a filter. That filter is also a two-dimensional matrix which takes a patch of the original image, the task of the filter is to recognize and find patterns in the pixels. The patch then go through a number of filters to then give an output of what the image is [27].

One main problem with images are their sizes, which makes them cost a lot of time and computing power to process. Convolutional neural networks are designed to make the dimensionality of images smaller in a variety of ways. Filter is one way to make the dimensionality smaller [27].

**Recurrent neural network**

Recurrent neural network or RNN is one of the most well known and powerful subsets with Long short-term memory (LSTM). The Recurrent neural network is a type of neural network designed to recognize patterns in sequential data. Applications where this kind of data types are used are for example in stock market predictions, time-series data from sensors, and when generating text. What differentiates neural networks, such as convolutional neural networks or other neural networks, from recurrent neural networks is that they do not take time and sequence into account, which recurrent neural networks does with a temporal dimension. Something that also differentiates recurrent neural networks from other neural networks is that it does not have any layer structures, just one layer. Recurrent neural networks are called recurrent because the output at each current time step becomes the input to the next time step. This means that at each element of the sequence, the model does not just consider the current input but also what the model remembers from the previous elements [28][29].

In Figure (3.2) a simple example of the architecture of a fold and an unfold recurrent neural network are displayed. The green circles are where the RNN models are, which are made of a layer of memory cells. When this master thesis is written the most popular memory cell to use is LSTM. LSTM is designed to overcome the previous problem that recurrent neural networks had with problems such as error signals that vanished or blow up or learning to bridge long time lags. LSTM can learn to bridge these type of long time steps up to 1000 steps at the same time, even with the case of noisy input sequences, without any loss of time lag [30].

![Figure 3.2: Recurrent Neural Network architecture.](image)

There exist different types of recurrent neural network architectures for different types
3.3 Game Physics

Because this paper will involve the computer game Counter-Strike: Global Offensive, some game physics and human behavior in games need to be mentioned. Game physics is physics in the games, such as how the light will travel, how blocks should behave when they collide, how particles such as fireworks, the ballistics of bullets, smoke, sparks and explosions move, etc. In the beginning, for over five decades ago, each game producer developed its own physics for their games. Today most games do not create their own, instead they are using a third-party framework for physics, rendering, animation memory management, and much more, which is also called a game engine [33]. Counter-Strike: Global Offensive is using the Source 3D game engine, which is developed by Valve. Source Engine has, since 2008, opened up their SDK, Source Software Developer Kit, available to all of its users [34]. They also created Valve Developer Community (VDC) which is a community with lots of useful articles about the games and other useful information [35].

3.4 Social aspects of Cheating in online games

Counter-Strike: Global Offensive is an online game played, for the most part, with other real players, with that in mind some social aspects of cheating in online games need to be mentioned. There have been found that the motivation for cheaters is individual for all cheaters but the motivations can be placed in three different larger groups. The first motivation is purely monetary. The prize pool in online games have increased during the past years as well as the precises on virtual goods, which can be bought in most online games. Virtual goods are for example skins on weapons in CS:GO and can cost up to $20,000 [36].
Chapter 3. Theoretical framework

The second motivation is to get an advantage while competing in an urge to win, these are often very competitive people. Finally, the third motivation for cheaters is just to be able to have fun and become better in the game without putting the time and effort in to the game [37]. A cheat code developer company make about $50 000 each month [38]. Something that could be seen is that cheaters have more cheating friends then a normal gamer have. If a cheater get caught by any anti-cheat software they are more restrictive with the privacy settings because they do not want to show it to the gaming community. This indicates that the cheater knows that cheating is not accepted by the gaming community [37].

3.5 Related Work

Earlier researches have been done in the field of online gaming and especially some in the field to create machine learning based anti-cheat software for online first-person shooter (FPS) games. Although, there are many different researches about anti-cheat software for FPS games using machine learning, different approaches have been used to find the best result. In 2006 a paper was released by Yeung and his research team. The paper was about how to use a dynamic bayesian network (DNB) model to detect cheaters using the cheat aimbot in FPS games. The DNB model, which handles time series, uses six random variables to classify if the player was using the aimbot or not by the probability of the cheater passing a certain threshold. The model displayed good degree of accuracy, but the authors state the importance of fine tuning the threshold parameter [39]. In another research paper from 2011, the research team use different supervised learning techniques to create a framework for cheater detection in the FPS game Unreal Tournament III. The different supervised learning techniques that used were Neural Neural Networks, Naive Bayes, Decision Trees, Random Forest, and Support Vector Machines. The supervised learning techniques that gave the best result were Support Vector Machines and Naive Bayes [40]. Something the research team did not take in consideration is that the aim of a cheating player is changing over time, which means that lack of time series model could be consider as an issue. The most recent research, which was found, in this field was a thesis by Salman Khalifa made in autumn of 2016. In the thesis Khalifa uses the machine learning model Hidden Markov Models (HMM) to detecting cheaters in the FPS game Counter-Strike : Global Offensive. Khalifa's machine learning HMM is showing positive indications to how to classify cheaters [41].

Although, there have been some positive results in previous researches about machine learning for cheating detection, more complex solutions needs to be done to catch the cheaters. Therefore this master thesis will focus on implementing a machine learning, which uses deep learning that uses sequential data in time series, data to identify potential aimbot cheaters in Counter-Strike : Global Offensive.
Chapter 4

Method

4.1 Literature Study

To get an understanding about machine learning and different use cases for it, an literature study were made. The study were conducted to get information about similar cases where machine learning were used to identify cheaters in online games. Since the subject, of preventing cheaters with machine learning, is rather new just some studies have been done. The material was gathered from Google Scholar as well as relevant books and articles written by writers educated in the subject. Search words used were “first-person shooter games”, “machine learning” and “identify cheaters”, amongst others.

4.2 Programming Languages, Virtual environments & libraries

To make scripts in CS: GO, which will be used to gather data for the machine learning algorithm, it needs to be written in the language SourcePawn. SourcePawn is derived from Pawn, a programming language with a C-like syntax. For making the machine learning algorithm, a programming language with extensive documentation will be chosen. One of the most popular machine learning languages is Python. Python is a powerful language and also user-friendly. Due to the enormous documentation, Python was chosen as the language for the machine learning. For Python there is also a virtual environment that is used for data science and deep learning, called Anaconda. With the virtual environment, specific package versions for projects could be installed without worrying about version conflicts. To make the machine learning algorithm or the recurrent neural network the neural network library Keras and also the open-source software library for machine learning Tensorflow, were used. These two libraries were chosen for their user-friendly appearances when building and training neural networks. The data input analyses was made in the free computer program for numerical calculations called GNU Octave.

The following versions were used for this master thesis:
4.3 Learning Algorithm

As could be read in the theory chapter of this master thesis, there are a lot of different learning algorithms in machine learning for many different types of cases. The machine learning algorithm that was chosen for this master thesis was conducted through researching different options and possibilities. Each option was then evaluated by finding advantages and disadvantages. When searching for the most suitable algorithm for the thesis, two learning algorithms were founded. These two different types were recurrent neural network and a one-dimensional convolutional neural network. The final choice was to create a recurrent neural network for algorithm. This neural network was chosen because of the reliable documentation regarding the algorithm, but also for how the input data was shaped in different time series which is preferable for recurrent neural networks.

4.4 Development

4.4.1 Data gathering

To make a reliable machine learning algorithm, a large collection of data is needed in order to train the model. To gather data for training the machine learning algorithm, lots of CS: GO rounds were played. These rounds were played on a Dedicated Server from Valve with CS: GO. The server was created to have the test environment as consistent as possible and to be easily modified. The server was modified to log more data then it should do normally. Data about how the player aimed were logged to an external file while the game was played. All the games were played on the map de_dust2 with the mode Casual and played as two teams of five players. Nine bots were involved in each game as well as one real person. The data, about how the player aimed, were only logged from the real player to make it as consistent as possible.

Data from 300 rounds were gathered. Of these 300 rounds, 200 were played without the cheat aimbot and 100 were played with the cheat aimbot. The cheat aimbot was downloaded from a Github repository. Since the cheat aimbot helps the player aim, data about how the player aiming were gathered to try to identify if the player is using the cheat or not. An example of how the raw data looked when it was gathered for one time step is given by

\[ L02/03/2020–13:08:46:\text{onplayerunruncmd.smx} AimingAtClient=falsePitch=5.207493Yaw=-69.532730. \]  

(4.1)

Each second, 128 time steps were logged while playing the game. For one round, a span between 1636 and 16497 time steps could occur. AimingAtClient is a boolean which tells if the player is aiming at an opponent or not. Yaw is movement in sideways, which goes from -180 deg to +180 deg, and Pitch is movement when looking up or down, which goes from -90 deg to +90 deg. These two coordinates tell if the player looks up or down and left
or right, also shown in Figure (4.1). The Roll value is not given in the data since it is not changing, which means that it will always be zero and can therefore be excluded. After the data was gathered it was transformed so it could be read by the network.

Figure 4.1: Rotations axis (Roll, Pitch and Yaw) for the player motion in the game.

From the 300 rounds of data 122 rounds were chosen since they were bigger than 5376 time steps. The data were then divided into 6 smaller sequences of 896 time steps each, this because LSTM only take up to 1000 inputs at the same time. 52 of those were with the cheat aimbot and 70 were without the cheat aimbot. 30 rounds from each group were picked to analyse if there were some big differences in the data. A fast Fourier transform (FFT) and histogram were created to see any potential differences. The fourier transform take a time-based signal and represent it in the frequency domain, given by

\[
S(f) = \int_{-\infty}^{\infty} s(t)e^{-j2\pi ft}dt, \tag{4.2}
\]

where \(s(t)\) is any practical signal and \(S(f)\) is the signal represented in the frequency domain [42]. Fast Fourier Transform also take, a time-based signal and represent, it in the frequency but is does it in a more efficient way then the Fourier transform does [43]. FFT were computed on the Yaw and Pitch signal to see if there were any big difference when using the cheat aimbot or not.

Histograms are a kind of approximate representation of the distribution of categorical or numerical data. Histograms were made to see if there were any differences in the activity between aiming with the cheat and aiming without the cheat which was given as a boolean value. The boolean were grouped so that if, for example, several -1 came at the same time, they were put together to a value. Then when the value was changed to 1, it was gathered together and so on. The mean and standard deviation was also calculated to get more information.
4.4.2 Building the Model

After the data was collected and transformed, building of the machine learning algorithm was started. The algorithm that was created was a recurrent neural network (RNN) since the data was sequential, which is another name for that the data was in time series. To start building the model, a guide by Jason Brownlee was used to get a grip on how to make a recurrent neural network [44]. From the guide and the information from the theory section, the model was built to fit the right purpose.

To make a recurrent neural network model with sequential input data using Keras, the following command need to be written: `Sequential()`. This is to make the model know what input shape it should expect. Since the model will use long short-term memory (LSTM) a layer needs to be added to the model using the command `LSTM`. A dropout was also added to the recurrent neural network. Dropout is a regularization technique to avoid overfitting (increase validation accuracy). The activation function used for this task was `softmax`. The optimizer used for this machine learning model was `adam`, optimizer is used to reduce errors. The optimizer was used with default values, in Keras this is `learning rate=0.001, beta 1=0.9, beta 2=0.999, amsgrad=False`.

4.4.3 Train Model

After the model was built, the model needs to be trained. For training, 70% of the data was used and the other was used for testing. From the data gathering, the training and test data were divided into fixed size inputs of 896 time steps each since LSTM excess intervals of up to 1000 steps without loss of short time lag capabilities [30]. For training, 510 sequences of 896 time steps were used. The model were trained with 10 epochs, epochs is iterations on the data set. The batch size used for training was 32, which is the default. Batch size is the number of samples per gradient update. Figure (4.2) shows the model loss while the model was in training.

![Model loss](image)

Figure 4.2: Training loss for the recurrent neural network model.
4.5 Test Model

To test the recurrent neural network, the only action needed was to insert the chosen input while knowing what the output should emerge. If the recurrent neural network produces the wrong output, more accurate features or a larger number of elements included when building the recurrent neural network are needed. In the beginning, an actual test was made with a smaller sequence size, 384 time steps, and smaller data size, 60 rounds, to the recurrent neural network. The first test showed a 50% accuracy, which is the same accuracy as a coin flip. To try to increase the accuracy for the recurrent neural network, a bigger sequence sizes and more data when identifying potential aimbot cheaters were added. When a second test with the bigger sequence sizes and more data, 70 rounds were taken without using the cheat aimbot and 30 with the cheat aimbot. This gave a higher accuracy of little more than 70%. But since the data sets were divided 70-30 an imbalance between the data sets and more rounds with the cheat needs to be added. Normally when evaluating a recurrent neural network, a significantly larger amount of tests should be made, which will be shown in the result. An aim of over 90% accuracy will be needed to really be able to identify if a player is cheating with aimbot or not.
Chapter 5

Results

5.1 Results from Input data

To see if there were some changes in the data between cheaters or not cheater, plots were made to be able to interpret the data. The plots are showing 30 rounds when playing with the cheat aimbot and 30 rounds without. Figure (5.1) shows the result from the Fourier transform of 30 rounds with the cheat aimbot (Cheat, orange) and without the cheat aimbot (Fair, blue). As could be seen in the figure, there are no big visual difference between the frequencies for the Cheat and the Fair but there is some minor differences. Both plots, in Figure (5.1), use a sampling rate at 128 Hz. Figure (5.2) and Figure (5.3) shows the result in histograms. In the histograms, data about the activity between aiming with the cheat (Cheat, yellow) aimbot and without the cheat (Fair, purple) aimbot are shown. On the y-axis, is the number of occurrences, which is given by the number of times an opponent was aimed at or not aimed at in a row from the given. The x-axis shows how long time each occurrence takes in seconds. Both figures, Figure (5.2) and Figure (5.3), use a sampling rate at 128Hz. Something that could be observed from the two figures is that for Cheat, using the cheat aimbot, it gives a higher activity in changes between aiming at an opponent or not aiming at an opponent, which makes the number of occurrences to a total of 1581. The number of occurrences for Fair, not using the cheat aimbot, to an total of 495. To use both in one histogram, the shorter vector was padded with NaN. The Fair vector was padded since it had less than a third of the numbers of occurrences than the Cheat vector had. In Figure (5.2) each bin is divided in 250 milliseconds. Since Figure (5.3) shows the histogram of no aiming at an opponent, which is the main part of each round, it had a longer time differences. Therefore, the bin in Figure (5.1) is divided in 5 seconds. The numbers of each bin are given in Table (5.1) and Table (5.2).

<table>
<thead>
<tr>
<th>Table 5.1: Values from Figure (5.2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin, size = 0.25s</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>0-0.25</td>
</tr>
<tr>
<td>0.25-0.5</td>
</tr>
<tr>
<td>0.5-0.75</td>
</tr>
<tr>
<td>0.75-1</td>
</tr>
<tr>
<td>1-1.25</td>
</tr>
<tr>
<td>1.25-1.5</td>
</tr>
<tr>
<td>1.5-1.75</td>
</tr>
<tr>
<td>1.75-2</td>
</tr>
</tbody>
</table>
Table 5.2: Values from Figure (5.3)

<table>
<thead>
<tr>
<th>Bin, size = 5s</th>
<th>Fair</th>
<th>Cheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-5</td>
<td>204</td>
<td>733</td>
</tr>
<tr>
<td>5-10</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>15-20</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>20-25</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>25-30</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>30-35</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>35-40</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>40-45</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>45-50</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>50-55</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>55-60</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>60-65</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>65-70</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>70-75</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>75-80</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

In Figure (5.2) the mean value for Fair was $\overline{x} = 0.215$ and for Cheat was $\overline{x} = 0.115$. The standard deviation for Fair was $\sigma = 0.396$ and for Cheat $\sigma = 0.191$. In Figure (5.3) the mean value for Fair was $\overline{x} = 4.866$ and for Cheat $\overline{x} = 1.479$. The standard deviation for Fair was $\sigma = 11.950$ and for Cheat $\sigma = 5.756$.

Figure 5.1: FFT for aiming in Pitch and Yaw for aiming with cheat and no cheat aimbot.
5.2 Building the Recurrent Neural Network

After building the recurrent neural network unsuccessfully for over 40 different times, the final recurrent neural network was built using 122 rounds of games played were a round is 5376 time steps. Each round is then divided into six sample sizes of 896 time steps, where each time step has 3 elements, giving the recurrent neural network a total of 1,967,616 data inputs to train and test on. From the total inputs, 70% were for training and the rest for evaluation, which gave the recurrent neural network 1,377,331 inputs to train on and 590,285 inputs to test on. The entire building process, to train and give an evaluation, took in average a total of 18 minutes and 23 seconds. This means that it took over 15 hours to get the final building model done for the recurrent neural network. Each time the recurrent neural failed to reach the aim of 90%, the parameters were changed. The parameters that were changed was numbers of memory cells (LSTM), batch size, epochs, input sample sizes and number of training and testing rounds.

5.3 Testing the Recurrent Neural Network

Each time the recurrent neural network was built, an evaluation was automatically created. The evaluation is divided into ten epochs, an epoch defines how many times the learning algorithm will work through the entire training and testing data set. Then the evaluation runs ten times to get a more accurate result. If the evaluation of the learning algorithm failed to generate a successful answer, the training data and other parameters in the recurrent neural network model such as batch size, epochs, and numbers of LSTM were changed.
The final result for the recurrent neural network was an accuracy of 53.077%. The final parameters used were batch size of 128, the number of memory cells used were 200 pieces, number of epochs were 10, input sample sizes were 896, number of rounds for training was 85 and number of testing rounds were 37.

The final result for running the program for ten times looked like this

\[52.941179275512695, 53.84615659713745, 53.84615659713745, 50.22624731063843, 51.131224632263184, 52.48869061470032, 54.29864525794983, 55.203622579574585, 53.39366793632507, 53.39366793632507\]  Accuracy: 53.077% (+/-1.403)
Chapter 6

Discussion

6.1 Input data

While gathering the input data, with the script that logged how the player was aiming, latency issues occurred that decreased the gaming experience. Since the script affected the performance of the game, one solution will be to not use the script on all users at all times but will be used on selected players at selected times and also for random testing. Another solution could be to download the match and the use the script on the replay. This two solutions could be used to minimize that the player do not have a decreased the gaming experience but none of these methods were tested in the work. If the software identify a player as a cheater their friends will also be tested. Since cheaters are friends with other cheaters, as known from the social aspects study, this is reasonable.

The input data was gathered by the same person while playing with bots. To make the input data more significant, more rounds with different players against other real players need to be played. Although the input data was gathered from one person, the person tried to play the games with different types of styles to imitate a number of different players.

As could be seen in Figure (5.1) there were no big visual difference in the FFT, Fast Fourier Transform, between the Fair and Cheat but there are some minor differences. When the FFT was made in the beginning there were major visual differences. The differences depended on that when the Cheat were in the end points of the spectrum they went from $\pm 90$ to $\pm 360$ degrees. But when it was noticed and subtracted with 360, then the values which could be seen in the FFT in Figure (5.1) were given. In the result from the histogram in Figure (5.2) and Figure (5.3) more differences could be seen. One example that differs between Fair and Cheat when aiming and not aiming, is number of occurrences were Fair only have a total of 495 occurrences in 30 rounds and Cheat have a total of 1581 occurrences. This shows that when using the cheat aimbot, the activity is much higher then if not using the cheat aimbot. There is also a difference between the number of short occurrences, where Cheat have a higher number of short occurrences, both in amount and percentage. The mean and standard deviation values, in both Figure (5.2) and Figure (5.3), for Cheat are approximately half of the values of Fair. The values of mean and standard deviation are much higher in Figure (5.3) than in Figure (5.2). That is because the amount of time a player aims at an opponent is just a few seconds in one round. One round can take about two minutes, so a few seconds is not that much percentage of a round.
6.2 Building and Testing the Recurrent Neural Network

As written previously, to build an accurate machine learning or a neural network a lot of data is needed. When the first test of the model was made, only 60 of the 300 rounds were used, this resulted in an accuracy of around 50%. After that, 40 more rounds were added to try to increase the accuracy of the recurrent neural network. This gave an accuracy of about 70%, which is better than 50%, but since the distributions between the data set was imbalanced. The model would have a good accuracy score if it simply predicts not cheating every time. The final result was an accuracy of about 50% which is a lot lower then the aim of 90%. This could be since the given data not showing that much big differences between using the cheat aimbot and not using the cheat aimbot. Due to the lack of time and experience with LSTM and neural networks a higher accuracy could not be received. To possibly get the result more accurate, a lot of data need to be gathered and even more data points. Such data points that could be gathered to make the accuracy increase are opponents position, length to the opponent, and the players own position. The data could also be standardized and normalized to increase the accuracy of the recurrent neural network model. All rounds were not used since they had to be more then 5376 time steps long to be used. Rounds with fewer than 5376 time steps occurred when the player died an early death. The chosen time size restriction was determined after the data was gathered, resulting in that rounds with times under the restriction time were excluded from the training and testing data. Unfortunately, the excluded rounds could not be remade because of time constraints. The training parameters such as number of LSTM, batch size, and epochs could also be changed to get a more accurate result. Since LSTM is still under development more inputs could be used in a recurrent neural network. If, for example, the recurrent neural network could have the ability to read a whole round a time, instead of just the maximal of 1000 inputs, the accuracy of the recurrent neural network could maybe been higher [30].

6.3 Social aspects on cheating in online games

Based on the literature study made in this thesis, one concludes that cheaters have different motivations to why they cheat. Cheaters who cheat for purely monetary reasons are probably the most difficult to persuade to stop cheating, due to the fact that they pay money on cheats developed by skilled developers. But they are also the most important to convince to stop cheating. Both to develop and increase the seriousness for esport but also to maintain the good atmosphere in the gaming community. Every professional player which is proven guilty of cheating is a setback for esport and the gaming community. The other cheaters do not have the same impact on the big esport stage, but they could create inequality in the community. If a player sees or hears someone cheating, they should report that player. In the event that a player turns out to be cheating, the player gets banned from the game and then their friends should be investigated for cheating as well since it is proven that cheaters have more cheater friends. This social aspects knowledge could lead to that more players cheating can be tested and potentially banned from the game.
Chapter 7

Conclusions

The goal of this master thesis was to create a machine learning algorithm that could identify if a player was using a cheat aimbot in the first-person shooter game Counter-Strike: Global Offensive by using the data from how the player was aiming. The aim was that the machine learning algorithm should have an accuracy of over 90% to really be able to identify if a player was cheating with aimbot or not, this was not achieved. Although, an accuracy of just over 50% was achieved. This shows that the recurrent neural network, which was created, could just find minor conjunction in the input data and could just draw minor conclusions. But probably with more input training rounds, more data points, and standardized or normalized input data the accuracy of the machine learning algorithm could be increased. These potential accuracy increases could not be tested due to time constraints since the data gathering and developing the recurrent neural network took a longer time than expected. Cheating in online games is a problem for the individual player, the esport stage, and the gaming community. Due to that, all solutions to stop the cheaters is necessary especially when there is money in the prize pools.

However, the result of this master thesis is the first step towards creating a reliable anti-cheat machine learning algorithm and an implementation to a CS:GO server to collect required training data.

7.1 Future work

7.1.1 Identify cheaters in online games

Cheating in online games is a problem today for the individual fair players and also for the community. Since the prize pools and online tournaments starting to increase more focus need to be directed to developing anti-cheat software to make users compete on the same terms. Although the machine learning algorithm created in this master thesis did just show to draw minor conclusions, a lot of more data and programming in needed to make it really reliable and effective against cheaters.

7.1.2 More cheat detection

This master thesis is focused on identify cheaters using the cheat aimbot. Aimbot is not the only cheat for online gaming and in order to create a reliable anti-cheat system, more
research is needed to discover the other cheats. In order to make an reliable and trustworthy anti-cheat system lots of time and research needs to be put in to it.

7.1.3 Further use cases for other games

Since this master thesis report is limited to the first-person shooter game Counter-Strike: Global Offensive, the machine learning algorithm may not work directly on other games. To make the machine learning algorithm work on other first-person shooter games, the data input from these games need to be collected in the same format as how it was gathered from CS: GO.

7.1.4 New cheat

Since this master thesis will be a free text for anyone to read, there could be people who read this in purpose to develop a new cheat that will not be detected by the machine learning algorithm. That is why those who are creating anti-cheat software today do not show how they detect it. But since this is the first step to develop Brawl Gaming’s anti-cheat software, this first algorithm will be free to access to inspire people to create their own anti-cheat system with the help of machine learning. As this machine learning algorithm need to be further developed in the future, it will undoubtedly look a lot different.
Acknowledgements

I want to thank my supervisor at Umeå University, Leonid Freidovich, for helping improve this master thesis. Another thanks goes out to the other co founders at Source Empire AB for helping to conduct this thesis, Jonas Gustavson, Fredrik Johansson and Emil Ottosson. Last but not least, a big thanks to Alex Norrman for the motivational support during the peer reviewing moments.
References


[38] [Table] IAmA: We are aimbot coders, ask us anything. https://www.reddit.com/r/tabled/comments/ttmig/table_iama_we_are_aimbot_coders_ask_us_anything/, May 18, 2012. accessed: 2020-03-20.


Appendix A

Source Code

A.1 Recurrent Neural Network
A.2 Code for Gathering the input data
# RNN model
from numpy import mean
from numpy import std
from numpy import dstack
from pandas import read_csv
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import LSTM
from keras.utils import to_categorical
import tensorflow.python.framework.dtypes

def load_data(prefix=' '):
    # Load all training data
    train_inputs, train_outputs = load_data_group('train', prefix)
    # Load all test data
    test_inputs, test_outputs = load_data_group('test', prefix)
    # Zero-offset
    train_outputs = train_outputs - 1
    test_outputs = test_outputs - 1
    train_outputs = to_categorical(train_outputs)
    test_outputs = to_categorical(test_outputs)
    # The loaded dataset returns asn test and train fpr inputs and outputs
    return train_inputs, train_outputs, test_inputs, test_outputs

def read_file_load(filepath):
    dataframe = read_csv(filepath, delimeter_whitespace=True)
    return dataframe.values

def load_data_files(filenames, prefix=' '):
    loaded = []
    for name in filenames:
        data = read_file_load(prefix + name)
        loaded.append(data)
    # Stack the data so the features are in 30.
    loaded = dstack(loaded)
    return loaded

def load_data_group(group, prefix=' '):
    filepath = prefix + group + '/'
    # Load both falt and cheat content, as a single array
    filenames = ['AimingPool96.txt', 'AimingPitch96.txt', 'AimingYaw96.txt']
    # Load input data
    Inputs = load_data_files(filenames, filepath)
    # Load output data
    Outputs = read_file_load(filepath + 'Output96.txt')
    return Inputs, Outputs

def build_model(trainInputs, trainOutputs, testInputs, testOutputs):
    # Build the RNN
    timesteps, features, outputs = trainInputs.shape[1], trainInputs.shape[2],
    trainOutputs.shape[1]
    model = Sequential()
    # Added LSTM
    model.add(LSTM(100, input_shape=(timesteps, features),
                   return_sequences=False))
    model.add(Dropout(0.3))
    # Activation function
    model.add(Dense(outputs, activation='softmax'))

2020-05-08 - AntiChesTempy
A.2. Code for Gathering the input data

```python
2020-05-08

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the RNN
model.fit(trainInputs, trainOutputs, epochs=10, validation_data=(testInputs, testOutputs))
model.summary()  
# Model evaluation
__, accuracy = model.evaluate(testInputs, testOutputs, verbose=1)
return accuracy

def main():
    train_inputs, train_outputs, test_inputs, test_outputs = load_data()
    scores = list()
    for r in range(10):
        score = build_model(train_inputs, train_outputs, test_inputs, test_outputs)
        score = score * 100.0
        print('#%d: %.3f' % (r+1, score))
        scores.append(score)
    print(scores)
    print('Accuracy: %.3f%% (+/-%.3f)' % (mean(scores), std(scores)))
    if __name__ == '__main__':
        main()
```

```java
#include <sourcemode>
#include <zstrike>
#include <sdktools>
#include <sdkhooks>
#include <gamelp>
#include <clientprefs>

public Plugin myinfo =
{
    name = "Master Thesis Data Gathering Plugin",
    author = "Martin Willman"
};

//Convars
Handle g_hLogFile = INVALID_HANDLE;

//Convars Values
char g_sLogFile[PLATFORM_MAX_PATH];

// Other Values
char g_sLogPath[PLATFORM_MAX_PATH];
public OnPluginStart()
{
    // Convars that create the logfile
    g_hLogFile = CreateConVar("sm_sim_logfile", "logs/aimingCheat20.log", "The
    logging file starting from the SourceMod directory.");

    // Hook ConVar Changes
    HookConVarChange(g_hLogFile, CVarChanged);
    HookEvent("round_announce_match_start", Event_WarmupEnds,
    EventHookMode_PostNoCopy);
    HookEvent("round_start", Event_RoundStart, EventHookMode_PostNoCopy);
}

// Check if the player is in the game and alive, in that case
public void CVarChanged(Handle hCvar, const char[] sOldv, const char[] sNewv)
{
    // Configs Executed()

    // Auto-Mode
    for (int i = 1; i <= MaxClients; i++)
    {
        if (!IsClientInGame[i] || !IsPlayerAlive[i])
        {
            continue;
        }
    }

    //
    public void OnConfigsExecuted()
{

    GetConVarString(g_hLogFile, g_sLogFile, sizeof(g_sLogFile));
    BuildPath(Path_SM, g_sLogFilePath, sizeof(g_sLogFilePath), g_sLogFile);
}

// Called when a clients movement buttons are being processed
public ACTION::OnPlayerRunCmd(int client, int &buttons, int &impulse, float
    vel[3], float angles[3])
{
```
A.2. Code for Gathering the input data

```c
if (!IsFakeClient(client) || !IsClientInGame(client))
    return Plugin_Continue;

bool isPlayerAlive(client){
    if (isPlayerAlive(client)){
        // If the player looking at someone this is logged
        LogToFile(g_sLogFile, "AmingAtClient = true Pitch = \%f Yaw = \%f", angles[0], angles[1]);
    } else{
        // If the player not looking at someone this is logged
        LogToFile(g_sLogFile, "AmingAtClient = false Pitch = \%f Yaw = \%f", angles[0], angles[1]);
    }
    return true;
}

stock bool GetClientLookingAt(int client)
{
    float vAngles[3];
    float vOrigin[3];
    float AnglesVec[3];
    float EndPoint[3];
    float pos[3];
    float Distance = 6000.0;
    GetClientEyeAngles(client,vAngles);
    GetClientEyePosition(client,vOrigin);
    GetAngleVectors(vAngles, AnglesVec, NULL_VECTOR, NULL_VECTOR);
    EndPoint[0] = vOrigin[0] + (AnglesVec[0]*Distance);
    EndPoint[1] = vOrigin[1] + (AnglesVec[1]*Distance);
    new Handle::trace = TR_TraceRayFilterEx(vOrigin, EndPoint, MASK_SHOT,
                                           RayType_EndPoint, RayHitPlay, client);
    if (TR_DidHit(trace))
    {
        new Target = TR_GetEntityIndex(trace);
        if ((Target > 0) && (Target <= GetMaxClients()))
        {
            new String::name[64];
            GetClientName(Target, name, sizeof(name));
            if (GetClientTeam(client) != GetClientTeam(Target))
            {
                return true;
            }
        }
    }
    TR_GetEndPosition(pos, trace);
    // draw your beam
    CloseHandle(trace);
    return false;
}
```
3.png

// This function returns a boolean if player aims at player or not
public bool RayHitPlayer(entity, mask, any data)
{
  if(entity == data)
  {
    return false;
  }
  else
  {
    return true;
  }
}

// Log the event "ROUND START" to File
public Action Event_RoundStart(Handle event, const String name[],
bool dontBroadcast)
{
  LogToFile(g_LogFilePath, "ROUND START");
}

// Log the event "WARMUP END" to File
public Action Event_WarmupEnds(Handle event, const String name[],
bool dontBroadcast)
{
  LogToFile(g_LogFilePath, "WARMUP ENDS");
}