IMT4889 - Specialisation in Decentralised Technologies report
Teamfight Tactics Agument identifier

Benjamin Skinstad

Nov 2021

Contents

1 Introduction

For my Decentralised project, I researched and developed an object detection system capable of identifying and classifying icons found inside the game, Teamfight Tactics (TFT). While a player plays the game TFT, they will, over time, be presented with three choices that will strongly affect their gameplay, these choices are called "Augments". First, they will be asked to select three augments, aka permanent buffs, that will affect their board for all future matches until the game is over. If a player is playing the game, they can freely inspect their own, and the enemy augments to learn more about the current augments in play. However, if viewers are spectating TFT gameplay, for example, through the streaming platform Twitch, they will be unable to learn anything about the current augments in play, and since there are 150 wildly different augment possibilities, memorisation is not possible. The developed system is capable of identifying augment with 96% accuracy, making it a potential tool for Twitch viewers and others who need to quickly identify TFT augments.

This paper will go over the research and development of this project. The paper will start with it going through the motivation behind this task. The report will then describe the functional requirements of this system and then showcase the research and development of a system that tries to satisfy these. Finally, the paper will discuss my findings and highlight various challenges I faced through this project. I then present my conclusion and tips to anyone who wants to create a similar custom object detection system.

The research and development of this project started on November 1. 2021. Therefore, the short time frame of the project will be reflected in the quality of this project.

2 Motivation

TFT is a trendy game on Twitch, averaging 55 000 daily viewers, making it the 10th most popular game[1]. On November 3. 2021, TFT released a significant expansion named Set 6[2]. Set 6 introduced a game mechanic called "Augments". Augments are random permanent buffs that affect the gameplay in a significant way. There are almost 150 different Augment icons, making it difficult for players and viewers to memorize them all. When a player plays TFT, they are allowed to inspect all of the current augments in play freely, but while spectating a match, say on Twitch, there is no way other than memorization, manual lookup or asking the current streamer for augment information. As these options are rather tedious or slow, I tried to research and develop a Twitch extension capable of real-time object detection that would identify augment icons in real-time for viewers. This report documents why I did was unable to reach my goal, but it documents the research and development of a web-based object detection system capable of identifying augments in close to real-time. This report will cover object detection using template matching in detail but also touch upon web development, server and endpoint operations with more.

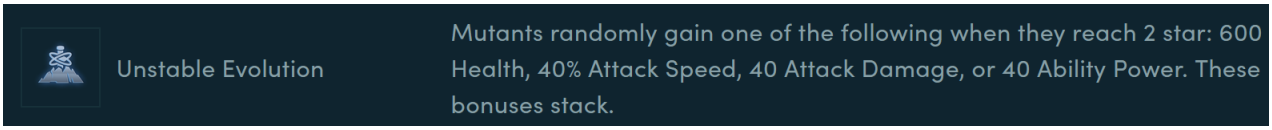

Figure 1: Example of an TFT augment, showcasing an augment icon (left), name (middle) and description (right)

3 System design

The system was designed to be published on the Twitch streaming platform as an extension for streamers to use. Based on prior experience and observation, I knew Twitch supported the usage of object detection extensions for streamers (fig2). These extensions allow viewers to be able to retrieve in-game information they usually are unable to retrieve. When a user hover-over parts of the streamers screen, dynamic information would pop up inside the video player to the viewers.

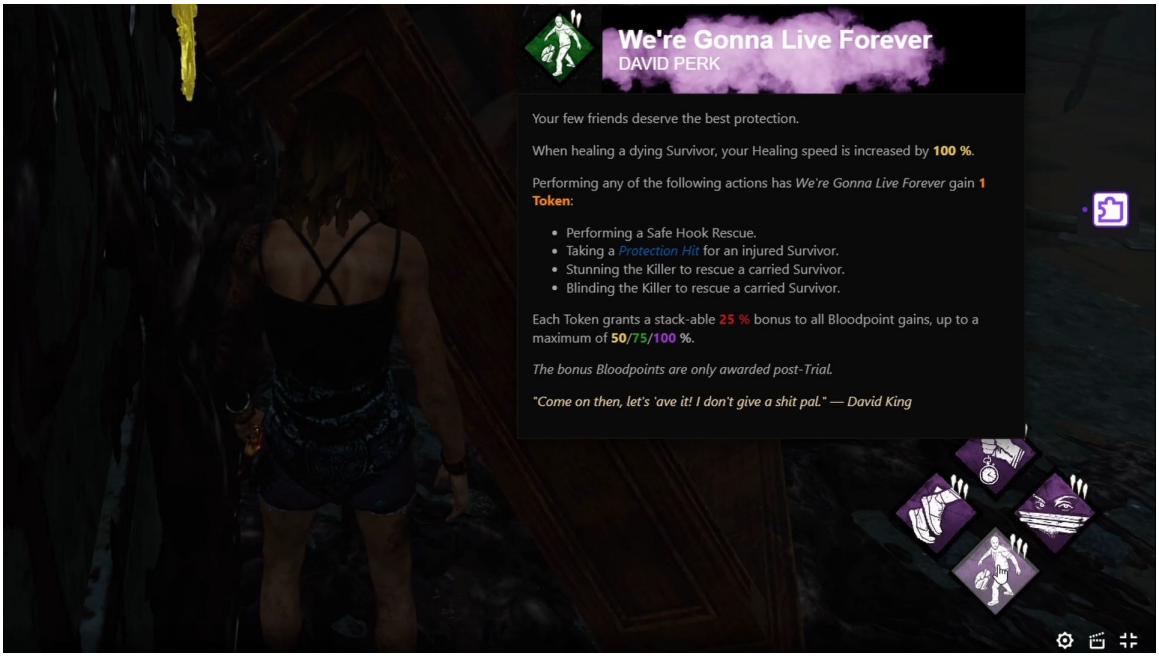

Figure 2: The Dead by Daylight Tool tips hover-over extension demonstration. My main inspiration for the system design and implementation

3.1 Twitch overlay

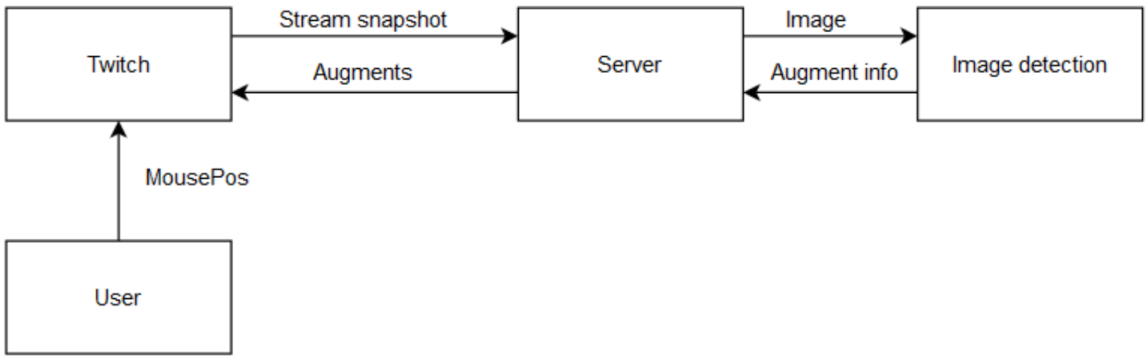

Figure 3: Initial system design built around Twitch

A TFT Twitch overlay needs to instantly identify all augments on the streamers screen and showcase their corresponding information in real-time. The extension must be able to identify the mouse position of a user and accurately convey the information of the augment the user is hovering over. To increase the odds that a streamer would decide to use the application, the system needs to be as lightweight as possible and not rely on external applications or services.

Features the Twitch overlay needs to have:

- Track mouse position of users
- Pass a snapshot of the current view to a server for object detection processing
- Display the server's response in the video player
- Display up to date information about the various augments in real-time
- Stand-alone application, the system cannot rely on external applications or services to work

The main inspiration behind this overlay came from the Twitch extension "Dead by Daylight Tooltips" (fig2). This extension makes it possible for viewers to inspect perks while viewing a stream. After a discussion with a developer of this extension, I was informed that he assumed that the extension used computer vision but was not given information about its inner working. For testing purposes, I created a simple Twitch overlay extension. However, after contacting the Twitch development team, I was informed that Twitch does not support real-time snapshots of the streamers point of view, making my extension impossible to implement. As Twitch extensions are transparent web pages, I opted to create a web application object detection system instead.

3.2 Website

The website should offer most, if not all, of the same functionality as the Twitch extension. The main difference between the potential overlay and the website is the lack of real-time snapshots from a Twitch stream. Like the Twitch overlay, the web page needs to perform object detection on a user-submitted image efficiently. To make it an alternative to the overlay, it needs to have the same functionality as the Twitch player but without the real-time snapshot ability.

Building on the features from the Twitch overlay, the web page needs to uphold these requirements:

- Easy upload of user-submitted images from Twitch streams
- Rapidly process the image for the user
- Display to the users the processed image for debugging purposes
- Display found augments name
- Display found augments icon
- Display found augments description

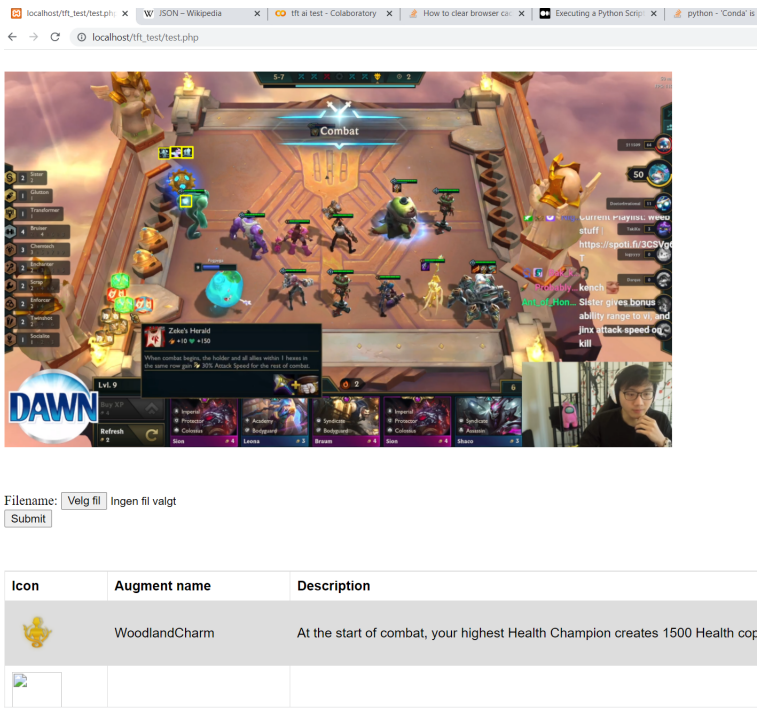

Figure 4: Finalised Website design, showcasing an early flawed object detection system

3.3 Computer vision server

The server handles the object detection on user-submitted images. The server needs to take in an image, perform object detection on said image, classify the various augments in the image, identify the icon and update the website with the correct augment information.

The server needs to fulfill these requirements:

- Perform object detection on an user submitted image, classifying augment types
- Identify the augment type, and its corresponding description 1
- Update the front end with identified augment information

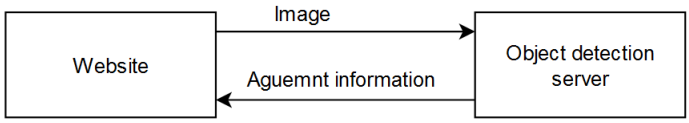

Figure 5: Very simplified system design

4 Template matching

Template matching in object detection has been theorised and explored since the 1960s. The earliest research paper I could find focusing on template matching was "Automatic character recognition: a state-of-the-art report" by ME Stevens from 1961.[3] Other publications on the theory of template matching was published in the same year. However, it is difficult to find information on when template matching changed from a more theoretical framework to a practical one, as the documented history of template matching is lacklustre.

The core principle of template matching is straightforward. Given a template image, often a small-sized icon, and a larger input image, the algorithm will try to slide the template image over the input image, checking for matches. After iterating over every pixel in the input image, the algorithm will return positions where it believes is a high likelihood that a match has been found.

Historically, template matching has been used for facial recognition, but now template matching is rarely used in its pure form these days. Modern template matching is now primarily used in parts of the manufacturing process, robotics, or edge detection. [4] Template matching are currently also used as part of other object detection frameworks.

OpenCV have template matching functionality built into their system. [5] The framework have multiple built in methods for adaptability reasons.

5 Development

5.1 Twitch overlay development

To test the capabilities of an eventual Twitch extension, I followed the official Twitch documentation and tutorials on how to make an overlay extension.[6] The Twitch test extension was an elementary web page capable of user input via JavaScript buttons inside the Twitch video player. Unfortunately, the development of the extension was dropped after a discussion with Twitch developers, who informed me that my design would not work with their platform.

5.2 Website development

The website is the primary interaction point for users. The website was designed for user-friendliness and speed. It is core that users are able to use the website as fast as possible. The website is built using mixed methods, combining CSS, HTML, and JavaScript.

HTML and CSS

The web page is very simplistic; it consists mainly of an HTML image viewer and a table. The image is updated by the endpoint, which overwrites the processed image after an update from the endpoint. The table is also populated with information whenever the endpoint returns any augment information in the form of a JSON object. The CSS is written to improve the table's look and put more content in the middle of the screen for user design purposes.

Filename: Velg fil Ingen fil valgt
Submit

Warning: file_get_contents(http://127.0.0.1:5000/companies): Failed to open stream: No connection could be made because the target machine actively refused it in **E:\xampp\htdocs\tft_test\test.php** on line **54**

Icon	Augment name	Description
E:\xampp\htdocs\tft_test\test.php on line **98** "/>		
E:\xampp\htdocs\tft_test\test.php on line **103** "/>		
E:\xampp\htdocs\tft_test\test.php on line **108** "/>		

Figure 6: An website snapshot, showcasing the skeleton without working JavaScript. Image credit: emilyywang on Twitch.com

JavaScript

JavaScript is used to perform file uploading and to connect to the endpoint, so the system can receive updates. When a user uploads an image, the file is uploaded to a specific folder located on the host machine by pressing the upload button. As speed is a priority, the upload function is designed to work with the snipping tool, making it possible to CTRL+V images captured by the snipping tool. This makes it possible to skip the saving and re-upload screenshots taken from the Twitch viewer, saving the users several seconds.

After a file has been uploaded, the script forces a refresh of the page to update the website with new information retrieved from the endpoint.

Filename: Velg fil Ingen fil valgt
Submit

Icon	Augment name	Description
	CyberneticImplants2	Your units equipped with an item gain 450 Health and 30 Attack damage
	PhonyFrontline	Gain 2 Target Dummies.

Figure 7: An website snapshot, almost correctly showing an processed image with augment information. Image credit: emilyywang on Twitch.com

5.3 Computer vision server

The server handles all of the object detection. The server utilizes OpenCV template matching as its object detection framework. The server also handles the initial system setup, system monitoring, and communication with the front end.

Template matching system

The template matching system is based on the code posted by opencv24, as found on their template matching documentation page. [5] The website demonstrates how little code is needed to perform template matching using the OpenCV library. The code demonstrates how to load in and process the main image of interest while also loading and processing the images (augment icons in this case) used for template matching. Additionally, the documentation demonstrates the various filters that can be used to change the template filter algorithm, how to work and adjust thresholds for detection and the registra-

tion of bounding boxes on the input image. I built the template matching system inside Google collab using python. The notebook can be found at: https://colab.research.google.com/drive/1hh8SBsJ_KpRIJ9TnpPLFcMookFCrsjLU?usp=sharing After the system was deemed satisfactory, it was moved from Google collab to a native machine for performance reasons.

```
i = 0
for template in rezied_icons:

    w, h = template.shape[::-1]

    # Perform match operations.
    res = cv2.matchTemplate(img_gray,template,cv2.TM_CCOEFF_NORMED)

    # Specify a threshold
    threshold = 0.8


    # Store the coordinates of matched area in a numpy array
    loc = np.where( res >= threshold)

    # Draw a rectangle around the matched region.
    for pt in zip(*loc[::-1]):
      cv2.rectangle(img_crop, pt, (pt[0] + w, pt[1] + h), (0,255,255), 2)
      IconsFound.append(icon_names[i])
      print(icon_names[i])
    i += 1
```

Figure 8: Core object detection function

Building on this system, I create functionality to load all relevant icons and scale them appropriately to fit a standard HD image, 1920x1080. The importance of scale will be discussed in more detail later, but the icons are usually scaled to be 32 pixels in height and width, to closer match their in-game counterpart.

The template matching is performed using the TM_CCOEFF_NORMED mode.[7] Quick experiments were conducted to test the accuracy of the various methods available in OpenCV, and TM_CCOEFF_NORMED seemed to be one of the fastest methods while still having high accuracy. The system saves all icon file names with a positive match, with a confidence level over 0.8.[7] The system casts all found icons from a list to a set to remove duplicates, as the system has high odds of registering the same icon multiple times at the exact location.

To speed up the performance of the system, I executed a statistical analysis of the typical input images to mark regions of interest. The region of interest consisted of approximately 90% fewer pixels. This region of interest was cut out of the input image, and by using this cropped image as the input for the system, one complete system iteration went from 15 second run time to 0.6 seconds. After the template matching part is complete, the system runs a lookup by using the dictionary, using the saved icons file names as keys. By doing so, the system finds the correlating name, file name and the corresponding description. The information is saved and formatted before being passed to the endpoint as a POST request.

Initial setup

For portability reasons, the server is responsible for the initial setup of the system. On startup, the server initializes all of its required folders and variables. It is in this step the system prepares a large, 150 entry dictionary list, one entry for each augments. As there was no universal, easily accessible database for augmenting information, I had to gather and order all augment data. The data is based on information found at Mobalytics.gg. The data was organized as a Python dictionary, with the icon filename as a dictionary key and augmented information as to its value. The augment data was then sanitized to avoid errors within python by removal of escape characters and other characters that created errors in the dictionary.

```
IconDict = {
"ANewChallenger" : "Gain a Challenger Emblem",
```

Figure 9: First entry in the dictionary, showcasing syntax of the dictionary. "ANewChallenger" in this case is the filename of the icon representing the augment "A New Challenger"

When the initial setup is done, the system goes into surveillance mode. In this mode, the system permanently surveillance a specific folder on the host machine. The surveillance folder is the same as the upload folder the website uses to store user-uploaded images. When a newly uploaded image is detected in the folder, the surveillance function tells the core template matching system to start, using the newly submitted image as input for their system.

The system setup was originally designed to perform more functionality; this is discussed in the discussion chapter.

5.3.1 Endpoint development

The computer vision server is unable to communicate with the webserver. This is due to conflicts between the web interface and the surveillance system. The endpoint system runs a python instance of a Flask server. [8] The server runs an endpoint on the localhost, handling both POST requests and GET requests.

The endpoints POST request handler reacts to POST requests from the server. The endpoint retrieves POST requests from the server and populates the local endpoint variables with variables retrieved from the server.

The endpoints GET request handler reacts to GET requests from the webserver. The web server responds to GET requests from the web server, sending it all augment information as a JSON object.

Due to the separation of parts, this is what the final system design implementation looks like

5.4 Website

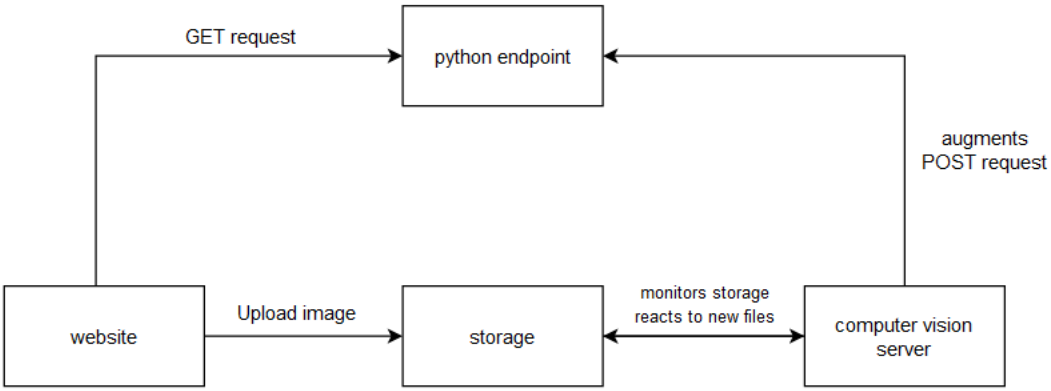

Figure 10: Final system design

6 Development issues

In this chapter, I will highlight some of the development issues I faced during the development of this system.

Bad icons The most time-consuming part of the project was working with the faulty icons. As I wrongly assumed that the icons were legit due to their source and quality, I spent an extremely long time trying to edit and perfect the icons to increase the system's accuracy. Changing one pixel was often enough to change the output by a lot. In addition, the faulty icons occupied various portions of the 256x256, making it difficult and time-consuming to perfect the icons pre-template matching. Template matching requires precise input images, with minimal leeway with input icons.

No API Since TFT does not have an open API, and it is not possible to get accurate augment information. Neither is icon information nor icon images. This makes it difficult to have up to date information on augments, significantly when icons are updated. The recent patch notes show that the icon information changes depending on the current patch, with icons, sometimes changing colour of the augments.

Not possible to verify findings Template matching does account for the scale of the input image. The difference between changing the icon scaling from 32x32 pixels to 33x33 pixels was sometimes massive in terms of accuracy. Running multiple iterations on different scales could work. However, since there is no way to differentiate between real and false positives, multiple iterations would find the same icon multiple times. Running multiple iterations would multiply the run time by the number of iterations, and with speed being of priority, this cannot happen.

Due to the size of the backlog of icons and the random element of the game, it is tough to verify that the system works on all icons. Errors have been discovered and fixed during testing, but it is hard to verify those are all fixed by now.

Linear scaling The run time of the system scales linearly; this means that if TFT were to introduce more icons in the future, the run time of the system would increase. Initially, when testing the speed of the template matching system, I was able to complete a system run in below a second, using 150 icons. I do believe this was a fluke somewhere as I cannot replicate that speed by any means, but it made me falsely believe that template matching was a fast-acting method.

Different coloured icons Some icons share the same design, only different in colour. Since the template matching system relies on the usage of black and white imaging for performance reasons, the system cannot differentiate

between identical icons. Adding extra functionality to perform colour identification whenever an icon with variants are detected could be possible.

7 Results

In this chapter, I will discuss the Performance and Accuracy test ran on the finalised version of the system.

7.1 Performance and speed

The speed and system performance are of utmost priority. The goal of the system was to work in a real-time fashion. Since speed was such a core priority, functionality that would increase accuracy and affect the system's speed was ignored. The current iteration averages its run time completion between 0.6 seconds to 1 second. This is deemed as an acceptable speed.

7.2 Accuracy

The accuracy was performed using the improved icon data set.

To check the system's accuracy, I captured 20 images of various game boards during various game states was uploaded for testing purposes. All images were taken via the Twitch platform from various TFT streamers. To make it test as many augments as possible, I prioritised late-game board states since they more often than not showcase three augments. The findings were manually evaluated and verified. The icons were all scaled to 33x33 pixels during testing. I ignored instances where the system identified an icon with the same look but of a different colour. An example of this is when the system identifies icons "Celestial Blessing 1", "Celestial Blessing 2", or "Celestial Blessing 3", as they are fundamentally the same augment but with different numbers.

Twenty images were captured from various Twitch streams and tested on, totalling 51 augment icons. The system accurately identified 49 icons, missing two, giving the system a 96% accuracy rate. The system has, in some cases, a high rate of false positives. The system is designed with this in mind, showcasing the users all found icon images and information, making it possible for users to weed out false positives vs real positives easily manually.

Regarding the icons, the system was unable to identify them; I discovered that the system was capable of identifying them when the icons were scaled to a different size. However, the system then loses the ability to identify the other icons, indicating some tiny differences between the various icons.

Image	Augment 1	Augment 2	Augment 3
1	yes	yes	NA
2	yes	yes	no
3	yes	yes	NA
4	yes	NA	NA
5	yes	yes	yes
6	yes	yes	NA
7	yes	yes	yes
8	yes	yes	yes
9	yes	yes	NA
10	yes	yes	NA
11	yes	yes	NA
12	yes	yes	NA
13	yes	yes	yes
14	yes	yes	yes
15	no	yes	yes
16	yes	yes	yes
17	yes	yes	yes
18	yes	yes	yes
19	yes	yes	yes
20	yes	yes	yes

Figure 11: Stats from the augment testing, indicating a 96% accuracy

8 Discussion

During this paper, I demonstrated a method to build a simple object detection system using template matching. This project feels like it would be a best-case scenario for template matching, but the results indicate that it is generally slow, with some issues in regards to accuracy. Additional methods had to be implemented to get the speed down to acceptable parameters.

Template matching

The results prove that template matching is now an outdated system, as to be expected from such a framework. Therefore, in the future, neural net-based detection systems should probably be used for object detection systems, even if I assume the training time and overhead are higher than this template system.

The template matching framework was chosen after discussion with students after I had presented my task description. As the augment icons have the properties as displayed in table 8, students argued that a simple model like colour detection or template matching would be enough. Based on some experimentation, I came to the conclusion that template matching would be a good fit for my case. It was later discovered that my initial template matching experiment was probably faulted and probably had an error in its implementation, as my initial experiment run time was sub-one seconds, with full HD input and 150 icons. Even so, using a non-neural net model allowed me to get a working object detection prototype quickly made, and if I had more development time, I'd change it to an alternative system.

Augment properties

- Static - almost always in the same position
- Generally same size
- Same background
- Never occluded or hidden
- Copy of all icons is available.

My implementation proves that it is possible to make a system capable of detecting out-of-the-ordinary objects like TFT augments in a relatively quick manner. Furthermore, due to the modular design, I assume it would be easy to reuse the core of my system in a Twitch extension if Twitch ever adds the ability to capture snapshots from a stream. I believe it could be possible to reuse this system in a web browser extension, but this would require more research.

As mentioned in the result section, it is seemingly possible to make it so my system could have a 100% detection rate. However, I believe such a system would impact the performance greatly. Instead, I believe the development of

a custom object detection neural network should be prioritized instead of perfecting this system. As previously stated, the system is designed in a modular fashion, making it easy to change the core object detection model without any system redesign.

Neural net - YOLO

Based on prior experience in working with object detection systems, I initially designed the system to use the YOLO object detection network[9]. During the course "Computer Graphics Fundamentals and Applications", I developed an object detection network that worked on custom data sets. To make my system work with the YOLO, I had to gather training data, but since there are 150 augment icons, I could not capture augment images from playing the game. As the game is weighting their representation of augments between games, it would be close to impossible to gather enough representative training data. To fix this issue, I made a python script capable of creating synthetic training data.[10] The script pasted icons in their correct position inside various template boards. I had previously removed the augments from various boards using photoshop to generate empty boards for the purpose of generating data. The idea was since I knew where all of the augment icons were placed and their type, I could automatically generate training data and its corresponding annotation file.

The drawback of using this process is that it requires a lot of training time and a lot of storage. For example, my 500 training images occupied 1.5GB. It also needs to be retraining to adjust itself to account for game updates, meaning Id need to retrain the network every week, while the template matching system would reflect updates instantly.

In talks with students, I was wrongly told that the usage of a neural net would be slow in terms of classification, which is one of the main reasons why I stopped perusing this system. The seemingly high training times and storage needed also assisted in my decision to drop the YOLO system.

I conducted a talk with an expert on object detection, Mohib Ullah, after some weeks of development to discuss issues with my system. He informed me that my assumptions on the template matching system and YOLO was faulty and encouraged me to use a YOLO network instead of in the future.

Augments

In terms of maintenance, developing a working API system that keeps track of the most up-to-date icon images and their description should be prioritized. The system does not learn or remember, meaning that any change to the system or icons are instantly reflected in the detection system, but a method to find the most up to date information is needed. Currently, the system requires someone to manually update the icons by using photoshop and, the system also requires someone to manually update the augment descriptions after a TFT update has happened.

Figure 12: Template board vs template board with augments added by scripts for training purposes

Scale

The current system relies on very exact and delicate input images due to the scale issue with the template matching framework. For example, it is not possible for users to submit any other type of image than full HD, something that might not be very user friendly. This paper has discussed potential fixes to address some of the significant issues with template matching, but I do believe that adding this kind of fixes is ignoring the greater issue with the framework and should be considered patchwork and not properly addressing the issue.

As mentioned in the result section, the scale was a reoccurring problem when working with this system, as changing the size of the icons with only a pixel drastically affected the accuracy of the system. A lot of time was dedicated to fine-tuning and testing the icon images to guarantee the highest accuracy, especially when working with faulty icons 8.

Optimization of the template matching system

As the performance was a key metric in this system, various methods for optimization were tried. An example of this was multithreading, as it could enable my system to perform multiple template matching operations in parallel. Unfortunately, multithreading while working on the same resource was difficult, and this module was fast scrapped. Since the system scales linearly based on the number of icons validated by the template matching system, alternative methods need to be implemented to speed up the process. It might be possible to drastically speed up the process with multithreading, but as discussed by Pan Wu, multithreading can only speed up certain tasks, meeting a plateau based on hardware and IO limitations. As the process needs to deal with IO and memory management, it's difficult to assume how much multithreading would improve this system, but it is worth investigating.

I also tried to further optimize the system by implementing CUDA into the system.[11] CUDA is a python platform created by Nvidia; it was designed to improve the effectiveness of OpenCV and other machine learning systems. Adapting the system to work with was complex and required a lot of external setups. It is not possible to make CUDA run the entire system, but I eventually managed to make CUDA run on my primary function, the template matching function. As the system showed no increase in performance after CUDA was implemented, I disconnected CUDA from my system. Since the CUDA system is created to optimize functions run through it using the graphic card, I suspect that OpenCV performs GPU optimization by default.

Faulty data

Faulty icons To gather all of the augment icons, I downloaded all 150 augments icons from the website Mobalytics.gg. Mobalytics has a long, rich history of working with Riot games properties in the past, so I assumed they had up to date information. The images were scraped from their website, and the icons loaded into storage. To avoid future issues, random non-ASCII characters were

removed from their names, including spaces. As the icons were all 256x256 transparent images, I created a python function that pastes the PNG icons on top of a dark background, making them a true copy of their in-game counterpart. As the icons occupy unnecessary space inside their 256x256 image, I hardcoded a trim function. The accuracy of the template matching function was very low when using the icons raw and improved significantly after trimming was performed. Due to the significant difference in icon size, my cropping function removed too much from specific icons, making it so the template matching would not categorize them correctly. The cropped of the icons was performed in the initial setup of the system, as mentioned in the development chapter. The solution to this problem seemed to be a manual categorization and cropping of icons, which I deemed not a priority. I relied on flawed data for the icons under a majority of the project. It is challenging to acquire the correct data set from Riot and even more difficult to validate it. This made it, so I had to rely on 3. party providers for icons and augment information. I was provided with a more accurate pack of icons near the end of the project.

Nearing completion of the project, I was given access to icons that was much more true to their in-game counterpart. I stopped using the old artificially created icons and replaced them all with the new data set. Even if the new icons were much more accurate, I had to manually check and rename all 150 icons due to them being incorrectly named and guarantee that they would fit with my program.

Faulty experiments Most of my assumptions on template matching were based on an early experiment that showed extreme promise in terms of system performance. The test was performance difference between Google collab and running the application natively. The experiment was run a few times for verification purposes, but the early results were mind-blowing. After a discussion with peers, we concluded that the speed was accurate. No further performance tests of the system were performed until the end of development, where I discovered that the error in my initial performance experiment. Occasional should have been performed to closely observe how the system change would affect performance.

Faulty assumptions This development project was plagued with assumptions about the various parts of development. Due to the short development time, I kept relying on the first and the best information I came across, more often than not trough my own faulty experiments or through flawed information given to me by my peers. Due diligence should have been prioritized higher over development, and experts should have been contacted earlier instead of my peers. YOLO is built to work with custom data sets [12], and since fast YOLO is able to perform classification at a rate of 155 FPS, it would be more than good enough for my system. [13]

Endpoint

Since the endpoint and the computer vision system run on the same system, I researched how to pass values between services. Multiprocessing methods and shared memory space between services were studied, [14], but after multiple failed attempts at implementing this functionality, I opted to use networking between the services. The endpoint variables are updated by POST requests from the computer vision service while handling incoming GET requests from the website.

The endpoint was created to separate links between the website and the computer vision server. The flask server system could not run as a stand-alone application while the surveillance part of the computer vision server runs. Running them as individual instances made it easier to develop the systems in parallel as well. The purpose of the system is only to obtain, store and pass information about augments when requested.

Errors in development

The slow development of the system can be traced back to the impact of faulty data, as discussed in 8, but also by the various errors in the development of the web system and in python development. The system is made up of multiple systems, made up of different libraries and languages. I overestimated my own ability in web development, and my slight unfamiliarity with the python programming language also haunted me. Some web development issues and some python development issues took me days to figure out.

9 Conclusion

On November 3. Riot games published their newest instalment of their game, Teamfight tactics (TFT). This update introduced the so-called augment system. This system introduced 150 different permanent buffs that could be in effect during gameplay. While playing TFT, the player can quickly identify every augment currently in development. Still, Riot did not publish any methods for Twitch streamers to make it possible for their viewers to identify the various augments currently in play inside a stream.

This paper documents the research and development of a custom object detection system capable of detecting custom icons/images found within a TFT Twitch stream. The paper relies on using the template matching object detection method. The report highlights the benefits and drawbacks of using this method, ultimately concluding why template matching is outdated and should not be used in new object detection systems.

The main benefits of template matching are the lack of overhead, ease of setting up, and decent performance when using smaller data sets and smaller input images. At the same time, the drawbacks of this system are lack of accuracy, reliance on exact template matching input images, overreliance on the correct scale of input templates, and extremely slow speeds at more extensive data sets and large input images.

This paper demonstrated how, even in a best-case scenario, template matching is a slow and inaccurate system. However, I still highlight methods that can be applied to increase the accuracy and performance of a template matching system. In the future, I recommend testing my design by using a multithreaded program for performance reasons.

This paper highlights the importance of knowing your specifications, doing proper preliminary research or contacting experts within their fields before committing to a development project. However, due diligence must be performed, and information must be verified before making design and development assumptions. During this development project, I made several assumptions based on false information that cost me a lot of development time regarding wrong icons, errors in initial performance estimation, or when I committed to using the template matching system for my project.

Overall I am happy with the project. However, I could have made a better final product if my skills within web development and python programming had been better before this project commenced or if I were to avoid some of the severe pitfalls I met during this project. If I were to redo this project, I would have dedicated my time to perform more background research and likely ended up using a neural network-based object detection system instead of one relying on template matching. Ultimately, my system has a 96% accuracy rate, making me confident that my system could be used by TFT fans.

10 Self-evaluation and reflection

I find it difficult to evaluate my work in this course. My initial project was to perform a literature review on serious games and rehabilitation, focusing on incentives, motivation and engagements. I fell short several times after spending a lot of time developing and refining my study design and research questions. After months of reading various reports and trying to get a good overview of the current state of the art regarding serious games and rehabilitation and repeatedly failing to come up with research questions that had not been answered or researched earlier by other scientists, I decided to work on this project instead. The short time frame affected this project in a highly negative fashion. It wasn't easy to dedicate time and effort to due diligence and feasibility studies with such a limited time frame. A short time frame also made each pitfall hit the project extra hard. To get as much development time as possible, I sadly had to prioritise the development of a minimum viable project over a stable product. A large part of the system is unstable and lacks proper safeguards and stability features; this was done to save time.

I am unable to count the hours I have spent in total on the various projects. Due to the elusive nature of the projects, a lot of time was dedicated to non-specific design and research. In regards to the development project, I spent almost 14 days non stop testing and developing the system, dedicating an annoying amount of time to developing the web elements of the project and working with the flawed dataset. Nevertheless, I am happy with my performance in this course, wishing I could have started this project at the start of the semester and not this late. I do still believe that even if my first project had no tangible results, I would greatly benefit from my findings during my master thesis.

I have been talking with other developers who might use my project as a base for their systems. The system works great at a local machine, meaning that it could be tied to a TFT client at a streamers machine or be used by individuals who need to look up augments very fast. I have been in talks with LoiusR, a French TFT tournament caster who might utilise this system for his tournaments.

In terms of learning outcomes, I have become more comfortable working with Google collab and python systems. I've learned more about working with insufficient data and the importance of verifying findings and data sets, and not trust assumptions. I've also learnt more about incorporating various frameworks to work with each other regarding endpoints, servers, web servers, and python instances.

References

[1] twitchtracker. Tfttwitchstats. https://twitchtracker.com/games/513143 , Accessed 2021.

[2] riot. tft set 6 intro. https://tftactics.gg/set-6-update , Accessed 2021.

[3] ME Stevens. " automatic character recognition: a state-of-the-art report recognition". https://books.google.no/books?hl=enlr=lang$_e nid$ = $iJ7IR_b x47gCoi$ = $fndpg$ = $PA1dq$ = 1999 + %22$template$ + $matching$%22 + $object$ + $detectionots$ = $mlQQMFINGmsig$ = $qSN7t_t tU0Lv$ − $6_O GaP4aI4ahZ8redir_e sc$ = yv = $onepageqf$ = $false$, 1961.

[4] Wikipedia contributors. Template matching — Wikipedia, the free encyclopedia, 2021. [Online; accessed 28-November-2021].

[5] opencv24. Template matching. https://opencv24-python-tutorials.readthedocs.io/en/latest/py$_t utorials/py_i mgproc/py_t emplate_m atching/py_t emplate_m atching.htm$

[6] twitch. Twitch extension tutorial. https://dev.twitch.tv/docs/extensions/ , Accessed 2021.

[7] opencv. opencv template matching documentation. https://docs.opencv.org/4.x/d4/dc6/tutorial$_p y_t emplate_m atching.html, Accessed$2021.

[8] tutorialspoint. Flask tutorial. https://www.tutorialspoint.com/flask/index.htm , Accessed 2021.

[9] yolo. Yolo documentation. https://pjreddie.com/darknet/yolo/ , Accessed 2021.

[10] Wikipedia contributors. Synthetic data — Wikipedia, the free encyclopedia, 2021. [Online; accessed 26-November-2021].

[11] Wikipedia contributors. Cuda — Wikipedia, the free encyclopedia, 2021. [Online; accessed 26-November-2021].

[12] Venelin Valkov. Face detection on custom dataset with detectron2 and pytorch using python. https://towardsdatascience.com/face-detection-on-custom-dataset-with-detectron2-and-pytorch-using-python-23c17e99e162, 2021.

[13] Sik-Ho Tsang. Review: Yolov1 — you only look once (object detection). https://towardsdatascience.com/yolov1-you-only-look-once-object-detection-e1f3ffec8a89 , 2021.

[14] python. multiprocessing python, 2021. [Online; accessed 26-November-2021].