Using Object Detection to Navigate a Game Playfield

Peter Kearnan Hyde-Smith

Marquette University

Follow this and additional works at: https://epublications.marquette.edu/theses_open

Part of the Computer Sciences Commons

Recommended Citation

https://epublications.marquette.edu/theses_open/737
USING OBJECT DETECTION TO NAVIGATE A GAME PLAYFIELD

By
Peter Hyde-Smith

A Thesis submitted to the Faculty of the Graduate School,
Marquette University,
in Partial Fulfillment of the Requirements for
the Degree of Master of Science

Milwaukee, Wisconsin
May 2023
ABSTRACT
USING OBJECT DETECTION TO NAVIGATE A GAME PLAYFIELD

Peter Hyde-Smith
Marquette University, 2023

Perhaps the crown jewel of AI is the self-navigating agent. To take many sources of data as input and use it to traverse complex and varied areas while mitigating risk and damage to the vehicle that is being controlled, visual object detection is a key part of the overall suite of this technology. While much efforts are being put towards real-world applications, for example self-driving cars, healthcare related issues and automated manufacturing, we apply object detection in a different way; the automation of movement across a video game play field. We take the TensorFlow Object Detection API and use it to craft an avoidance system in conjunction with a Java front end that allows fire and forget movement to augment normal play.
ACKNOWLEDGEMENTS

Peter Hyde-Smith

To my wife Christina, thank you for being my number one supporter, you have been with me through all the challenges in our lives and I love you dearly, you are a core part of the foundation on which we succeed. Thank you Dr. Yahyasoltani, you have been a great advisor. I appreciate your guidance and motivation. Thank you Dr. Puri and Dr. Kaczmarek, for your time and feedback as committee members. To my daughter and son, Nina and Conrad, thank you for your patience and love while I have worked on my degrees. I can’t wait to see what your own bright academic futures look like and I hope to repay you in kind. Thank you, Alexandra Broeske, you are an amazing mother-in-law and grandmother to my children and have provided me many hours and days of support to further my studies. Finally, to my father Peter Smith and my grandfather, the late Richard Smith. You both instilled the love of science and technology within me from an early age. Dad our conversations about your work as a chemist and the wider natural world. Gramps, our many trips in my childhood to the science museums and your example of being a lifelong learner. I hope to carry on your legacies throughout my career and pass it on to future generations in our family.
TABLE OF CONTENTS

ACKNOWLEDGMENTS......................................................................................i
LIST OF TABLES ...................................................................................................iii
LIST OF FIGURES ...............................................................................................iv

I. INTRODUCTION ..............................................................................................1
    A. Background Information .............................................................................1
    B. Goals ..........................................................................................................2

II. RELATED WORK .............................................................................................3
    A. Machine Learning and Video Games .........................................................3
    B. Object Detection .........................................................................................6
    C. Mean Average Precision (mAP) as a Metric for Model Efficacy ............7
    D. Important Datasets in Object Detection ..................................................8
    E. Single Shot Multibox Detector ..................................................................8
    F. EfficientDet-D2 .........................................................................................10
    G. Faster R-CNN ...........................................................................................11
    H. YOLO Framework ......................................................................................12
    I. GAN ..........................................................................................................11

III. METHODOLOGY ..........................................................................................13
    A. Tensorflow Object Detection API ............................................................13
    B. JavaFX .....................................................................................................13
    C. Java Tensorflow API ................................................................................14
    D. Hutta Dataset ..........................................................................................14
    E. Conditional GAN and the Creation of Synthetic Data ..........................22
F. VSI - Visual Simulation Interface..........................................................26
G. Hardware and Software............................................................................29
IV. DATA AND ANALYSIS...........................................................................30
   A. Performance of Object Detection Models............................................30
   B. Performance of Conditional GAN and its Effect on Model Efficacy.........31
V. CONCLUSION............................................................................................36
   A. Objectives..........................................................................................36
   B. Contributions.....................................................................................36
   C. Future work.......................................................................................37
BIBLIOGRAPHY............................................................................................38
LIST OF TABLES

TABLE I COMPARISON OF MODELS

30
LIST OF FIGURES

Fig. 1. Drivatars in Forza Horizon 5 ................................................................. 3
Fig. 2. Patent of Sony’s user profile to play games ............................................. 5
Fig. 3. Breakdown of the unmodified Hutta Dataset class annotations ............. 15
Fig. 4. An example of the log class ................................................................. 17
Fig. 5. An example of the stone class ............................................................... 18
Fig. 6. An example of the junk class ............................................................... 19
Fig. 7. An example of the stump class ............................................................ 20
Fig. 8. An example of the tree class ............................................................... 21
Fig. 9. Generator Architecture ....................................................................... 24
Fig. 10. Discriminator Architecture ............................................................... 25
Fig. 11. Generated stone images ...................................................................... 31
Fig. 12. Cropped stone artifacts ...................................................................... 31
Fig. 13. Example of placed synthetic object ..................................................... 32
Fig. 14. Discriminator Loses ......................................................................... 34
Fig. 15. Generator Loss .................................................................................. 35
.
I. INTRODUCTION

A. Background Information

Automation in game-play is not a new thing. There are many programs and features out there that allow you to get ahead in the game with little to no user input.

In many cases, this involves having to reverse engineer or gain direct access to the game application memory, which in many cases can be detected and then stopped by a game client and server. Overall, it is a risky process because in many cases (especially in multiplayer games) it violates the terms of service and can result in account bans. In some cases, however, the more famous examples like DeepMind playing StarCraft, there is a sanctioned API that is open to be interacted with. These are clearly more research focused and are posed as problems to be solved rather than any actual practical use. In either case, whether intentional or not, these bots look at the game state and react accordingly.

A less sophisticated way this can be achieved is by capturing via auto-key software, preset routes and commands for different tasks. This does not necessarily interact with the internal state of the game application, but can still be detected based on behaviors.

We propose a third alternative applied particularly to short and medium distance character movement, taking a balance between direct game-state interaction and overt scripting. Using object detection, we can capture segments of the game screen in a custom overlay and feed the images into models that can detect objects and use that information to programmatically assist the user with the inputs for the game. Tools like these are already being incorporated as assistive features, like Forza’s driving assistance.
or are in the process of being developed and will be released in the near future, where companies like Sony create user-customized AI profiles that can play games. Game-agnostic autonomous control is not only convenient but also can powerful tool for accessibility of games that do not have such built-in features.

**B. Goals**

- First, we create the game-agnostic interface that can glean information from the game world similar to that of a player, using TensorFlow Object Detection API and transfer learning. This is object detection in real-time on a novel data set from the game, Star Wars: The Old Republic (SWTOR) [1].

- Second, we improve the efficacy of the model used by the interface to process information by applying techniques from generative adversarial networks (GAN) to generate a more robust training data set.

- Finally, we program an avoidance algorithm to take into account the information capture and move to avoid objects that are in the movement path allowing for a fire-and-forget movement automation.
II. RELATED WORK

A. Machine Learning and Video Games

In a video segment with Ars Technica [2], the Creative Director of the Forza Franchise Dan Greenawalt speaks about the evolution of the game’s drivatars. Forza is a game franchise centered around automobile racing, and a key aspect to it is having other non-player characters inhabiting the roads. From the inception of applying machine learning techniques in 2005, to its current iteration in the most recently released game, Forza 5 Horizons, the differences and sophistication have come a long way. The drivatars are an example of using machine learning to solve game-related problems. Greenawalt goes on to explain, the data from that governed the behavior of these non-player characters (NPCs) in the early games was local and consisted of maybe, a hundred laps generated from the player itself. However, with the newest online game, there were millions of laps worth of data generated in the first week of release from the many players. This data, which consisted of things like car position, track, speed and other analytics like that, helped the developers of the game classify track portions to allow the

Fig. 1. Drivatars in Forza Horizon 5.
creation of behavior profiles that these drivatars could use. As seen in Fig.1, the player’s car in the center and the drivatars around it.

The company DeepMind, which is a subsidiary of Alphabet inc. has achieved two separate instances of game automation using Machine learning. The first was from the work of David Silver et. al relating to the problem of chess [3]. Chess, a problem that has been considered since the inception of the field of computer science, is a good example of creating an automated player. Early solutions were hardware based, these gave way to algorithms and finally with the late 1990's, the shift towards machine learning emerged. What is achieved in [3] is the adaptation of an earlier program, AlphaGo, which was created to solve the game of Go. The training techniques were generalized and formulated into a new program, named AlphaZero, which was then given the rules of chess. Within twenty-four hours, the computer player had surpassed human skill. Because of the generalization, AlphaZero was also able to play the game of Shogi as well.

Similarly, in 2017 a framework was created by Vinyals et. al which combined computer vision and reinforcement learning [4]. The framework included an overlay that could extract information from the StarCraft II, a real-time strategy game known for its competitive ESports space. Top players are usually measured by things such as clicks per second. With the overlay and the reinforcement learning, the agents were able to achieve limited success at a novice level. While this is good for a prototype and proof of concept, later in 2019 Vinyals et. al returned with the breakthrough achievement of their AI, named AlphaStar, being able to function at the Grandmaster ranking [5].

David Papp writes about his experience using machine learning to advance and automate tasks within the multiplayer roleplaying game Runescape [6]. The game, known
for its extensive skill system as well as other things like combat and player economy, requires much time investment. Papp uses Tensorflow and OpenCV in conjunction with the Java programming to create an interface that can control a character and direct it to interact with 'harvestable nodes' on screen. This is completely separate from the in-game engine, in contrast with something like the aforementioned Forza Drivatars. Information is only gained through object detection and then acted upon.

Sony takes all of these, one step further and has applied for a patent [7] to create a 'default game profile' based on information gained from monitoring a player's action when it plays. It can then control and play the game for the player.

Fig. 2. Patent of Sony's user profile to play games [7].
What all these examples have in common are the automation of behaviors of one or many aspects of a game's 'game-loop' - that is the small repeatable tasks that are a part of the greater experience of the game. While some of these 'loops' may have a certain allure when a player is new to the game, over time those can become less enjoyable, while still requiring a significant investment of time. Those enjoyable experiences turn towards a grind and run the risk of player burnout. As games are primarily meant to be subjectively 'fun' it is understandable that even the games industry itself is moving towards these supplementary technologies that can enhance the worth of the game to the user.

B. Object Detection

Generally, object detectors are an extension of CNNs used for classification problems. As such, object detectors are typically built upon a backbone network of an image classifier. In the case of transfer learning, the weights from those classifier backbones are then used as a starting point for training with whatever object detection architecture is used. Where they differ however, while classifiers tend to take the whole image into account and identify it, object detectors must achieve localization of the classified object and then do it for multiple classes. While there are several different notable architectures, in essence each breaks the target images into much smaller sub-images and attempts to ‘classify’ each of these, then remembering where the position was. The most notable architectures in use today are the Faster R-CNN, Single-Stage Detectors (SSDs) and the You Only Look Once (YOLO) framework. Each has their own strengths and drawbacks, which can then be balanced with trade-offs based on the problem domain. Some tend to have better inference times with the tradeoff being a lower mean average precision (mAP), while others are slower but have a higher mAP.
overall. SSDs tend to do well with larger objects, but have a harder time detecting smaller objects, this is offset by having some of the fastest inference times however [8].

C. Mean Average Precision (mAP) as a Metric for Model Efficacy

Mean average precision is the evaluation metric used by object detection to determine the efficacy of a model. The basics of it derive from using true positives, false negatives and false positives. True negatives are not used as they do not apply to the problem. Note, when computing if something is a true positive, the intersection over union (IoU) between the ground truth box and the predicted box is used to determine what qualifies. Typically, this threshold is set at .5 for a prediction to be counted as a true positive. That means if greater than or equal to half the area of the prediction bounding box is overlapping with half of the area of the ground truth box it will count as a true positive.

To compute the mean average precision, you must first compute precision and recall. Precision is computed by taking the number of true positives and then dividing them by the sum of true positives and false positives. False positives are in object detection are defined as the presence of objects being predicted where there is no ground truth specified. Recall is computed similarly to Precision, but instead of false positives, there are false negatives. False negatives in object detection are defined as the presence of in a location specified by ground truth annotations that have not been predicted as objects by the model. That is, the model does not see an annotated object.

To calculate average precision, you need to plot the precision-recall curve which is done by taking the predictions on a body of evaluations ordered by confidence and
calculating the change in recall and precision. This gives a curve. After this precision-recall curve is graphed, you can then calculate the area under the curve. In the COCO challenge, this is taken one step further, using a varying IoU threshold each class’s AP is predicted. From there, over all classes the mean average is taken. In particular the COCO mAP uses the range from 0.05 to 0.95. The .5mAP and .75mAP use .5 and .75 IoU thresholds respectively, which is why the .5 mAP usually is the highest [9].

D. Important Datasets in Object Detection

There are two primary datasets that are used as a basis for training the network backbones of the object detectors. The MS COCO dataset [9] and the VOC dataset [10] and the. Each of these involves a large body of images that are annotated and used as challenge benchmarks for state-of-the-art detectors. Published in 2015 in a paper entitled Common Objects in Context, the MSCOCO data set is used both in the training object detection portion of the pre-trained models in the TensorFlow object detection API as well as in many of the detection architecture papers as an alternative to the other competition datasets. The COCO dataset contains ninety classes and over a million images. The PASCAL VOC dataset [10] is an earlier counterpart to the COCO dataset, published originally in 2005 and was used as a basis for a challenge in object detection research. Starting in 2007 the dataset had 20 classes and 9,000 images which increased over time until 2012 where it had 11,530 images and 27,00 annotations. Examples of classes include person, bird, bicycle and chair.
E. Single Shot MultiBox Detector

The original implementation of the Single Shot MultiBox Detector (SSD) was created to address problems that were involved with contemporary object detection architectures. At that time, the Faster R-CNN architecture had been released, citing a 5 frames per second (FPS) inference time while boasting state-of-the-art model accuracy [11]. However, identifying the bottle neck of having to train both a feature box proposal network and the pixel resampling that came along with it (in addition to the classification portion)

Liu et al. in [12] set out to achieve real-time speeds while also maintaining a comparable accuracy. The original implementation of the Single Shot MultiBox Detector uses the VGG-16 classification network as a backbone. It is a convolutional feed forward network, but the classification layers are removed and replaced with the object detection features. The main feature that sets their SSD architecture apart was the introduction of a fixed number default boxes that are used in conjunction with layers of feature mAPs over different resolutions. At each resolution layer, bounding boxes and their offsets are calculated and confidence scores are generated. This allows multiple potential high-quality detection. During training, anything with a Jaccard index of .5 or higher being labeled as a prediction with the ground truth. With these proposals, it can then choose the best fit during the last layers of the detection model in the suppression layers. One thing to note, boxes in the smaller resolution layers will not predict objects that are at a larger scale, thus preventing many smaller 'positives' inside a larger object.

The original implementation uses the VOC mAP guidelines with results of .79 with 52 fps. This differs with the newer coco metrics in that it does not take into
consideration the average precision taken over multiple IoUs. Therefore, it is more comparable to the mAP at 50% score.

Further improvements were made on the SSD Architecture by Lin et. al involving their modifications to the cross-entropy loss, called Focal Loss, that deals with class imbalance between the foreground and background. Furthermore, they used various ResNet backbone networks with a FPN head [13]. Adopting the improvements from [13] Huang et al. brought the configurations over to the Tensorflow API making them available for transfer learning, of which the model variant we use is one [8].

F. EfficientDet-D2

The EfficientNet-B0 backbone [14] was developed by using Neural Architecture Search which optimizes accuracy and FLOPS. This is then used as a backbone (pre-trained on ImageNet) for EfficientDet-D2 which is single stage detector that uses a bi-directional feature pyramid network (BiFPN) (connected feature network layers that are used to detect objects at different scales and are bi-directional) [15], is smaller, computationally cheaper and scales well [16]. Sawada et al. use the EfficientDet architecture on a custom data set of images taken from the mirror angle of cars. Their baseline implementation achieves an mAP of 47.6% which is then improved upon with their own modifications to achieve an mAP at 50% IoU of 52.2% [17]. Further object detection work was done by Srikanth et. al in an attempt to identify objects with the inclusion of hand to assist the visually impaired. While their results have a decrease of the mAP to 75.85%, they postulate could be due to the small number of images (500) they use in their data set [18].

A final example of EfficientDet in action comes from Li et. al with the detection of watercraft. Using different implementations of EfficientDet (-D0, -D3, D4) they achieved
mAPs of 65%, 76.4% and 87.5% and FPS between 21 and 42. Their tests with YOLO were comparable, but EfficientDet's accuracy was better overall [19].

G. Faster R-CNN

Faster R-CNN is an evolution of the earlier Fast R-CNN architecture. It too uses a classification backbone, but differs from the SSD in that it is a two-stage detector. Instead of default boxes, there is a region proposal network that generates a fixed number of boxes. After this, there is a second stage that goes through every proposed region and classifies it [11]. Faster R-CNN are generally more accurate than their SSD counterparts at the cost of some speed, though advances have greatly increased their inference time [20].

H. YOLO Framework

The Yolo framework is worth mentioning, as it is the state-of-the-art architecture that is constantly pushing the field forward. As of 2022 it boasts as the fastest architecture for object detection, using what it calls a 'bag of freebies' that is constantly being improved. Not only does it have competitive and in some cases, the top mAP scores of object detectors, including an mAP of 56.8% with 30 fps or higher on a V100 GPU [21]. It also has the advantage of not having the drawbacks that the SSDs run into, namely small object detection. Unfortunately for the scope of this project, the framework has limited support for integration with Java where the TensorFlow framework has both the object detection API and a full implementation in Java to make use of that API.
I. **GAN**

While the dataset we use is currently around 7,100 images with many more annotations across five classes, it would be useful to increase the number of images. However, to create more images, we turn to GAN which uses a generative and discriminant network that feed into each other until the generative network can produce samples that the discriminator cannot distinguish from the original data set [22].

These techniques have been taken further, both to create synthetic data sets and to augment and retrain classifiers with improved results. Example data sets include dermatology [23], microbial [24], tomato plant diseases [25] and traffic [26] have all shown that augmenting the original data set with a certain amount of synthetic or GAN transformed images shows an improvement on the baseline. Some of these greatly benefited from the synthetic additions because of class imbalances [23].

Another way GAN has been used, is to optimize existing synthetic images to with features from originals resulting in 'enhanced' synthetic images which then can be used to improve the results of models [27].

In some cases, the usefulness goes even beyond just improvement, in the case where medical laws prohibit or limit the use of patient data, synthetic data can be generated to train models, mitigating ethical concerns [28].

As the previous examples suggest, the application of GAN can be beneficial across a wide variety of data sets. We feel that applying GAN to our own data set will help generate a more diverse examples, while reducing some of the workload needed to annotate them.
III. METHODOLOGY

A. *Tensorflow Object Detection API*

The TensorFlow object detection API [8] is the backbone of this project. It provides a wide range of pre-trained detection models that are a convenient starting point at which we can continue training via transfer learning with our own dataset. For use with our programs, we have chosen to work with the SSD ResNet152 V1 FPN 640x640 and the SSD MobileNet V2 FPN 640x640. These both take an image with a height of 640 pixels as an input and output detection boxes.

B. *JavaFX*

JavaFX is an application framework used to create GUI-based Java applications. It is a partial successor to Swing, though due to evolving demands within the industry it has not fully superseded it. Instead, it exists as robust alternative to creating desktop and web applications. JavaFX programs generally consist of the application, a stage, scene and then all child objects. In some cases, it is possible to implement everything via code, and some of the simpler programs do; however there is also the ability to use FXML, which is a companion format that can be used to define layouts that are then loaded into the application.

Typically, these FXML documents are managed by a controller class which is where much of the dynamic nature of the platform exists. Both in-code and FXML have their strengths. Defining things strictly with code is useful in situations where a more dynamic approach to layouts is needed, while FXML is useful for defining static, well organized and easily readable layouts that don’t need to change as much. Regardless,
JavaFX brings modern GUI features to the Java ecosystem and allows for example, the Java TensorFlow API to be showcased in the form of displaying images and inferences.

C. Java Tensorflow API

The TensorFlow API has an implementation in Java. While it is experimental, it still functions as a bridge between the trained outputs of python TensorFlow APIs and the Java language. This is particularly useful in situations where access to Java’s robust environment is needed to perform certain tasks. It especially useful where primary language familiarity and proficiency is with Java. In essence, the saved model bundles that are the results of training with the Tensorflow Object Detection API can be exported and then loaded into a Java program where it can then be interacted with.

D. Hutta Dataset

The data set we used has been generated from the game, Star Wars: The Old Republic [1] on the planet Hutta (a starting playfield in the game). The images have been taken to match the model inputs. There are five object classes, which were chosen to be the most common and contained that a character in the playfield would run into. These classes are: log, junk, stone, stump and tree. There were two rounds of collecting pictures for this data set, each with their own process of capturing objects for training. To capture images for the data set, we have written a command line Java program that uses the AWT Robot library to fire off a series of repeated screen captures. The program is configurable with the number of images to take, the time between each image capture and the output file names. To annotate images, we have used the VGG image annotator tool [29].
The first collection involved finding examples of objects and using a circling pattern at close, medium and far distances. This created a varied data set, both looking at the object as well as a changing backdrop. There were a total of 1,694 images collected in this manner. One drawback to this, is most of the object annotations ended up being in the center of the image, with less variation along the y-axis.

As such, in an attempt to remedy this, an alternate way of collecting the images was conducted. An object was selected and the capturing point of view was moved in a striping pattern along the y-axis, essentially dividing the image into columns and moving the image along these. 5,500 images were captured in this way. Images where created that had objects located in a more evenly distributed manner in the image, versus the mid-horizon plane of the previous method. A drawback to this was less variance of the background space within the image, but the hope was that combining both into a single data set to train on, would provide good coverage for both.

Fig. 3. Breakdown of the unmodified Hutta Dataset class annotations.
Both methods usually contained one larger object which would have been the subject of the capture, while having many smaller objects in the background. While not every object in the images is annotated, the attempt was made to capture most of those that would be considered potential objects to avoid. Overall, the dataset consists of a total of 7,194 images and 17,202 annotations. Per Figure 3. The log, junk and stump classes are generally balanced, with the tree class having a slightly lower number of annotations. The stone class however does lag behind by about 8-9%, effectively around 1,500 less annotations. To address this class imbalance this is where we apply the use of conditional GAN to increase the frequency of that object class within the dataset. For model evaluation we have used separately generated dataset consisting of 100 images and 635 annotations.
Fig. 4. An example of the log class
Fig. 5. An example of the stone class
Fig. 6. An example of the junk class
Fig. 7. An example of the stump class
Fig. 8. An example of the tree class.
E. Conditional GAN and the Creation of Synthetic Data

We used the CGAN implementation at [30] modified to take labels into consideration. Our generator architecture consisted of a latent noise dimension input of 128, which was up sampled over the course of several layers to a 256x256 output. Radford, Metz and Chintala layout several architecture best practices which [30] incorporates in its implementation. Among the best practices, are the use of Conv2DTranspose, Batch Normalization and Leaky ReLU layers in both the Discriminator and Generator. A dropout with 0.5 in the discriminator and we use the Adam optimizer with a learning rate of 0.0002 and a beta 1 value of 0.5 [31].

The output of an image with 256 height and width was chosen, to create a confined object that would be scalable to within the medium and large area ranges of the COCO eval metrics. The data used to create train the GAN is extracted from the main dataset. Annotations that are of medium size or larger according to the COCO eval metrics are used to copy the sub-image onto a square based on the longest side. The background is set to a random color sampled from the extracted image. These then form the basis for our GAN training dataset, which contains around 6000 of these thumbnails. Since the class imbalance we are primarily trying to fix involves the stone class, we generate a range of images and chose by hand which ones that resemble the subject matter. After this we annotate the image and extract it using python and then modify the existing dataset by placing these extracted object images onto the existing 640x640 in a random place near the horizon band they usually occur.

With this modification we then take note of where each object is placed and bring the number of annotations up in line with the other class distribution. With that we then are
able to measure class AP and check to see if there is an improvement in training based on the additional synthetic annotations.
Fig. 9. Generator architecture.
Fig. 10. Discriminator architecture.
The AWT Robot class is a part of the Java AWT library [32] which is a part of the swift framework. The swift framework was one of Java's answers to creating GUI-based desktop applications. This was later supposed to be deprecated and turned obsolete by the introduction of the JavaFX library, but it has largely survived and, as in our case, can actually be used in tandem with its later counterpart.

The Robot class can be used to control other desktop applications and was originally intended to be used as an automated testing tool for the AWT apps. However, it can be applied further in a way that allows a Java program to automate other tasks. We have used it in two ways primarily, first to capture images and via the screenshot function and secondly, to provide a way to programmatically input keystrokes, mouse movement and mouse clicks to other programs. This was functionality is exemplified in David Papp's automated program for mining in Runescape. We have been inspired by it and adapted it's use to be in line with our MMORPG control set.

The movement functionality is activated with the ctrl+M hotkey which calls the toggleMovement() method. This method toggles a continuous movement mode that drive the in-game character forward using the W key as input. Specifically, the Robot object sends a key-press down signal, then sends a delay signal for a set period of time, in this case we have chosen 100ms, and finally sends a subsequent key-press up signal. This is used in conjunction with the AnimationTimer class from the JavaFX library, which sends a pulse continuously by calling it's handle() method.
The handle() method is called by the JavaFX application framework when the AnimationTimer class is given the start(). This handle() method is actually defined within the aforementioned toggleMovement() the method which is where the AnimationTimer is instantiated if it does not already exist. To clarify, the toggleMovement() method will call start() or stop() on the Animation timer based on the moving variable which is defined in the MainController class (which is where this logic resides in).

The inference and capture functionality are initiated in the same way, by using the ctrl+E hotkey. This calls the toggleContinuous() method which repeatedly draws a screen capture (BufferedImage) on a canvas object in the JavaFX scene. Depending on the inference interval setting, which defaults to 1 per second, the toggleContinuous() method will periodically call the captureSingleImage() method with a true flag. This signals the program to call not only take a screen shot and render it, but also to send the image to the Model class which handles the actual interaction with the saved model bundle.

The Model class opens this saved model (usually the first call takes around six or seven seconds, with subsequent calls being shorter) and allows it to serve its default functionality, which is to take an input image and return inferences in the form of tensors from which things like confidence scores, bounding boxes, inference classes etc. can be extracted. These are then wrapped in an Inference object which is then sent back to the MainController which calls a pruning function on it.

The pruning function sets the confidence threshold. All boxes with a confidence below the threshold are discarded. These are then rendered on top of the buffered image for each inference call, giving a visual representation of what the model is detecting. Though the confidence threshold tends to be an arbitrary setting, generally lower
thresholds yield more boxes per object while higher thresholds yield less objects per object. The strictest settings tend to miss detections, so while it is configurable a setting in the midpoint between the two extremes usually produces good enough inferences to work with.

Finally, the algorithm to avoid objects relies on a few things. First, the objects must come within the screen threshold which is denoted in yellow lines, two vertical and a singular horizontal. The vertical lines represent the x-minimum (left-side) and x-maximum coordinates and the horizontal lines represents the y-minimum coordinate (the max is the bottom edge of the image, so 640).

As objects boxes are detected to have an overlap of the predicted box area with this avoidance threshold, the program initiates the avoidance movement. This consists of sending the button_press_down and button_press_up commands with the W code (for forward movement). There is a similar screen activation portion that fires before this, moving the mouse to a (512, 512) and then sending a few rapid mouse_mutton_up/down. This ensures the focus is on the correct program.

To avoid, the algorithm checks the lower-left and lower-right avoidance thresholds to determine which way to turn. If neither, then it is chosen randomly. While avoiding, the algorithm will not initiate object avoidance again until it corrects back onto its original course. The rationale for this is to ensure the character ends up traveling in the general direction originally chosen, as part of the algorithm involves a turn and forward movement avoiding the object, enough forward movement to pass it, a turn back towards the original path of movement, a small amount of forward movement to and then a final turn onto the original course.
Again, both the detection and movement functionality are meant to be toggled on and off via hotkey and to work in tandem to control the game character and avoid obstacles. It functions as an enhanced auto-run feature.

\textit{G. Hardware and Software}

There are several different pieces of hardware that have been used throughout the process of this thesis. There are the GPU and AI nodes on the Raj HPC cluster \cite{raj} and two personal machines. The AI nodes have eight Tesla V100 GPU with GPU interconnect, which allows them to be fully leveraged by the TensorFlow library, as opposed to the GPU nodes which only leverage two at a time.

The personal machines consist of an NVIDIA GeForce GTX 1080 8GB VRAM and 16 GB of RAM and another NVIDIA 2070 SUPER with 16 GB of RAM. Note, the personal machines are used to run the front-end and real time inference, while Raj was used for training.

Finally, while modern cards would likely be able to handle the workload of running the game Star Wars: The Old Republic [1] and run the detection inferences, we have decided to Parsec [33] which is an application that can be used to host a game on one computer and stream it to another. This is similar to VNC protocol or remote desktop in function, but is optimized to be real-time with a good enough internet connection (which is more achievable than buying a state-of-the-art GPU).
IV. DATA AND ANALYSIS

A. Performance of Object Detection Models

Though the object detection portion chiefly relies on transfer learning, the TensorFlow object detection API [8] has its quirks and learning curves. A few things to note when dealing with the original benchmarks laid out in the TF Model Zoo, they are taken on hardware that is comparable or better than some of the NVIDIA Titan cards. So, while the speed benchmarks are a good starting point for determining which models are the fastest, there is degradation in performance, especially on the considerably weaker hardware of the aforementioned NVIDIA GTX 1080, which is only about 60% as powerful as the TITAN series. Thus, the inference times we were achieving with the full SSD architecture made it difficult to get to something remotely close to real-time object detection, especially considering some of the extra overhead that goes along with the regular Java program. Alternatively, there is the MobileNet optimization of this SSD architecture, which achieved similar efficacy while nearly halving the speed. TABLE I shows the different inference times in milliseconds and mean average precision metrics.

<table>
<thead>
<tr>
<th>Model Architecture</th>
<th>Inference Time (ms)</th>
<th>mAP</th>
<th>mAP@50 IOU</th>
<th>mAP@75 IOU</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD ResNet152 V1 FPN 640x640 (ResNet152 - COCO 2017)</td>
<td>200</td>
<td>0.445424</td>
<td>0.789523</td>
<td>0.116961</td>
</tr>
<tr>
<td>SSD MobileNet V2 FPNLite 640x640 (default)</td>
<td>110</td>
<td>0.407489</td>
<td>0.604904</td>
<td>0.427835</td>
</tr>
<tr>
<td>SSD V1 FPN 640x640 (ResNet V2 152 No Coco - ILSVRC-2012-CLS)</td>
<td>114</td>
<td>0.34195</td>
<td>0.58</td>
<td>0.386</td>
</tr>
<tr>
<td>SSD MobileNet V2 FPNLite 640x640 (GAN-Modified)</td>
<td>-</td>
<td>0.348</td>
<td>0.58</td>
<td>0.386</td>
</tr>
</tbody>
</table>

To achieve some measure of hyperparameter optimization we created a python script to generate configs, randomizing hyperparameters that could be modified in arbitrary ranges. We then prepared a SLURM script that ran singular training jobs based on these
configurations. There were roughly 170 runs that resulted in an increase of 6-7% increase in mAP50 over the default configurations.

B. Performance of Conditional GAN and its Effect on Model Efficacy

Training the GAN was a balance between monitoring for mode failure, seeing how realistic the generated article looked and convergence failure. Mode failure is when the generator becomes too specialized and begins to generate only a small or even singular object. This tended to happen after 300-400 epochs. However, the features did not look realistic enough until well after that. So, it was a balance between finding the best-looking generator checkpoints and to use weights from those to generate artifacts. The artifacts we decided are shown in Fig. 11.

![Generated stone images.](image1)

These crops shown in Fig. 12. were accomplished by using the Microsoft Paint 3D smart crop tool [34] which does a good job of homing in on the ‘object’ part of the image. This
saved a lot of time. Examples of how the crops would be placed are in Fig. 13. By doing this, we were able to generate an extra 1,337 stone objects to add to the dataset and programatically calculate their annotation coordinates. This fixed the class imbalance of the stone, which had less than the others and being able to do this in such a manner saved about 5-6 man hours of manually collecting more images and annotating objects. We then used this new modified dataset to retrain the SSD mobilenet and we were able to achieve a 2-3% mAP increase, measured at across several checkpoints during the training process.
as well as the final 50,000 step mark, a comparison you can in TABLE I of the previous subsection.
Fig. 14. Discriminator loss.
Fig. 15. Generator loss.
V. CONCLUSION

A. Objectives

The intention of this thesis was to prototype a program that would act as an interface between the machine learning aspect of object detection and the practical application of it to a limited-scope real-world problem. We found that that with a custom dataset, transfer learning and a front-end application, it is possible to navigate a game character through a playfield.

B. Contributions

Over the course of work on this thesis we created a new fully annotated dataset comprising of over 7,000 images and over 17,000 annotations. We showed demonstrable results applying that dataset in a practical use-case, adding to the body of work surrounding object detection and what it can be used to achieve. Not only in real-world applications, but also in virtual instances reflected in the computer game subject matter of our dataset.

We also then used a conditional GAN to synthesize data based on the original body of the dataset and add more objects where needed, this resulted in an improvement of the model efficacy by fixing class imbalances.

Finally, using the APIs available we created a non-trivial application that showcases how the implementations of TensorFlow across two different languages can come together with more traditional code to create a useful game-inference visualization interface that aids with the automation of tasks.
C. Future work

With the current prototype, the algorithm is programmatic rather than leveraging machine learning techniques. There is still room for improvement, particularly in the application of reinforcement learning. Future work could improve on the simplistic avoidance measures that are currently being used and create a more robust decision-making framework to control the character. In terms of conventional programmatic solutions, we suspect it would be possible to create some sort of cache of detections, though this would involve quite a bit of work to try and remember both the detections and the movements of the character.

Perhaps a better solution would be to delegate the decision-making responsibility to some sort of reinforcement learning controller, of course this would come with its own challenges and require the introduction of a reward and feedback system based on the visual and spatial information captured by the inference portion.

Further expansion of the automation tasks is another goal to making a more robust and fully featured system, it could include not just navigation but other game tasks, like node interaction, combat and other more interactive tasks.

Finally, it would be interesting to explore beyond the TensorFlow framework and try to incorporate the Yolo framework with more customized solutions and the ability to incorporate those more state-of-the art detectors as well as moving away from the sometimes-unwieldy TF implementations for purpose-built solutions in Java.
BIBLIOGRAPHY


https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf


