Detecting Quick-Time Events in Detroit: Become Human

Griffin Baxter
Computer Science and Software Engineering
University of Canterbury
Christchurch, New Zealand
grb96@uclive.ac.nz

Richard Green
Computer Science and Software Engineering
University of Canterbury
Christchurch, New Zealand
richard.green@canterbury.ac.nz

***Abstract*—This paper proposes a method to detect Quick-Time Events (QTE) in the game 'Detroit: Become Human' (DBH) by Quantic Dream. The proposed QTE detector uses the Hough gradient method to detect circles in each frame of recorded footage of DBH, corresponding to the boundaries of QTEs. These are cropped and used to detect either text using Optical Character Recognition (OCR), or symbols using the Harris corner detector and the arrangements of coordinate combinations. The proposed method managed to correctly detect 73.1%, partially detect 19.4%, and incorrectly detect 7.5% of QTEs. The method notably had issues with detecting the 'W' QTE, due to the untrained implementation of Tesseract OCR being used. This accuracy was slightly higher than prior research detecting characters in Super Smash Brothers Melee, which reached 68.25%. However, the high confidence and accuracy attained in prior research for the real-time tracking of e-sport games was not reached by this method, as proven by its API usage. This paper is the first in terms of academic literature to propose a method for detecting QTEs within a video game.**

***Index Terms*—computer vision, games, quick-time event**

I. Introduction

A Quick-Time Event (QTE) is a button or gesture prompt that requires the player of a video game to, often swiftly, perform the required action in order to succeed. These are used for the purposes of immersion, in which the actions of the characters in-game are mimicked. Additionally, they are also used to showcase skill by executing fast-paced actions. Because of this, many games containing QTEs that are only displayed for a short duration are difficult to play for those with certain disabilities. A study has shown that the guidelines and strategies for developing video games with accessibility features are often ignored by game developers [1]. Therefore, having the ability to detect QTEs would be extremely helpful for those unable to perform such actions. The proposed method outlined within this paper provides the ability to detect QTEs and the action that they represent in the game 'Detroit: Become Human' by Quantic Dream [2].

II. Background

A paper has been published showcasing the ability to detect unknown objects in video games using qualitative stability analysis [3]. This method used the gravity and stability of objects already tested within the scene, and was proven to accurately and reliably detect objects in the game 'Angry Birds'. Notably, the method was able to pick out blocks from the game of differing symbols, as shown in [3, Fig. 1]. This is relevant to the topic of this paper, due to the blocks being similar to common QTEs, with symbols in the middle indicating the button to press. The paper mentioned the usage of supervised learning methods would optimise the clustering of objects. However, the method they used was limited by the fact that it was passive, meaning that it only worked with static images. Additionally, examining multiple frames could have improved the performance of their method, in which objects could be analysed in terms of how they move in the game environment.

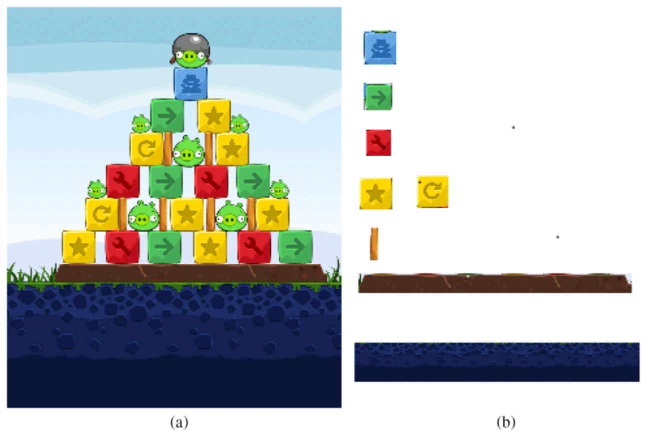

Fig. 1. (a) A scene in 'Angry Birds'. (b) The detected objects in the scene, categorised in rows.

Another paper made use of a customised Convolutional Neural Network (CNN) for detecting characters in the fast-paced fighting game Super Smash Brothers Melee (SSBM) [4]. A visualisation of the customised CNN and its classification-detection relationship is shown in [4, Fig. 2]. Using this approach, alongside training upon additional annotated frames, the accuracy of character prediction reached 68.25% across all four characters. Unfortunately, this accuracy, alongside the fact that this only works with a single game means that this approach requires further work on top of what has already been achieved. This can easily be attributed to the fact that 3D characters had to be detected. Additionally, the method

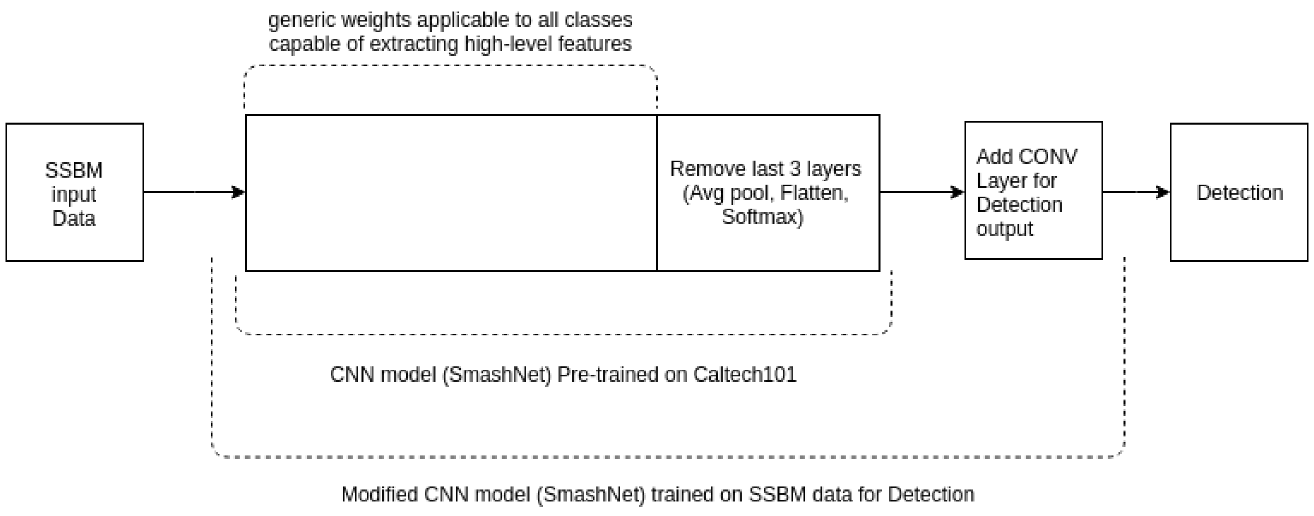
generic weights applicable to all classes capable of extracting high-level features
SSBM input Data
Remove last 3 layers (Avg pool, Flatten, Softmax)
Add CONV Layer for Detection output
Detection
CNN model (SmashNet) Pre-trained on Caltech101
Modified CNN model (SmashNet) trained on SSBM data for Detection

Fig. 2. Visualisation of classification-detection relationship regarding the system pipeline for detecting characters in Super Smash Brothers Melee.

performed at a range of 6 to 10 Frames Per Second (FPS), meaning that not every frame could be analysed in real-time.

Research was also conducted for the creation of a video game testing framework [5]. Pre-processing was undertaken in order to remove non-shadow objects from a 3D scene, using an exclusive-nor (XNOR) operation on images of the same scene with and without shadows, as shown in [5, Fig. 3]. A combination of the Harris corner measure [6] and Canny edge detector [7] algorithms were used to identify jagged edges corresponding to aliasing within the image. The Harris response R, shown in eq. (1), was used as a score to detect such jagged lines. The results certainly worked for the use-case presented, however, is limited in the fact that it can only be used primarily for the detection of aliasing within shadows.

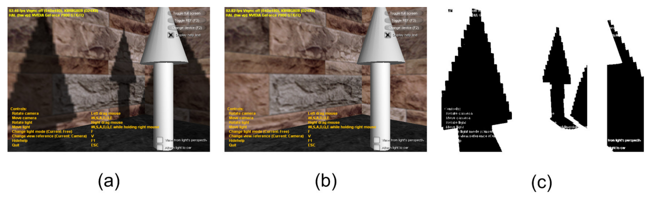

Fig. 3. (a) 3D scene with shadows. (b) 3D with no shadows. (c) XNOR operation performed producing image only containing shadows.

$$R = \det(M) - k(\text{trace}(M))^2 \qquad (1)$$

Research into the detection of objects in video games is limited, and so looking into online articles was beneficial for further information on the topic. One such article describes the enabling of real-time tracking of e-sport game statistics using video object detection [8]. The approach made use of pre-trained Optical Character Recognition (OCR), and a Mask R-CNN. An example showing the pre-trained OCR working with all text on the screen is shown in [8, Fig. 4]. The Mask R-CNN is a variation of CNN for both object detection and segmentation [9]. The method allows for the detection of game statistics such as each player's equipped weapons and the team's score. However, a majority of the elements detected are static in terms of where they are positioned on the screen.

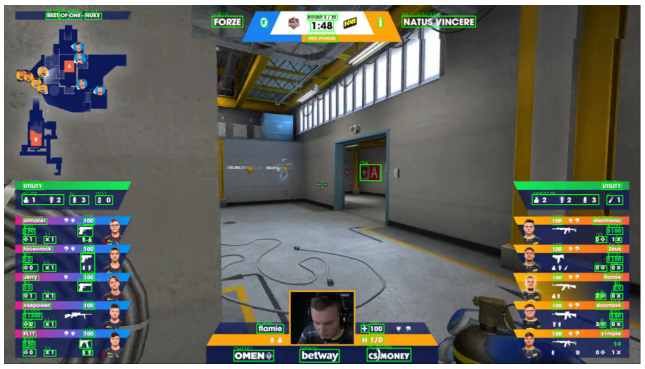

Fig. 4. An example of the pre-trained OCR used to recognise e-sport game statistics, with green bounding boxes around the detected text.

Overall, prior research into the detection of objects in video games is both limited in terms of the amount of research, and the scope and success of their proposed methods. The article discussing the tracking of e-sport statistics [8] was the most successful, in which its success came from its ability to provide simple information accurately, without complexity diminishing both of these attributes. This paper aims to mimic the properties of simplicity, in terms of the information gathered, and more importantly, accuracy compared to prior research.

III. Proposed Method

The proposed method makes use of OpenCV [10] and Tesseract OCR [11] libraries for Python. The QTE detector was designed to work with a video file capture from the PC version of the game 'Detroit: Become Human' (DBH) [2]. As shown in [2, Fig. 5], the method places red circles around the detected QTEs and displays the detected keys and gestures.

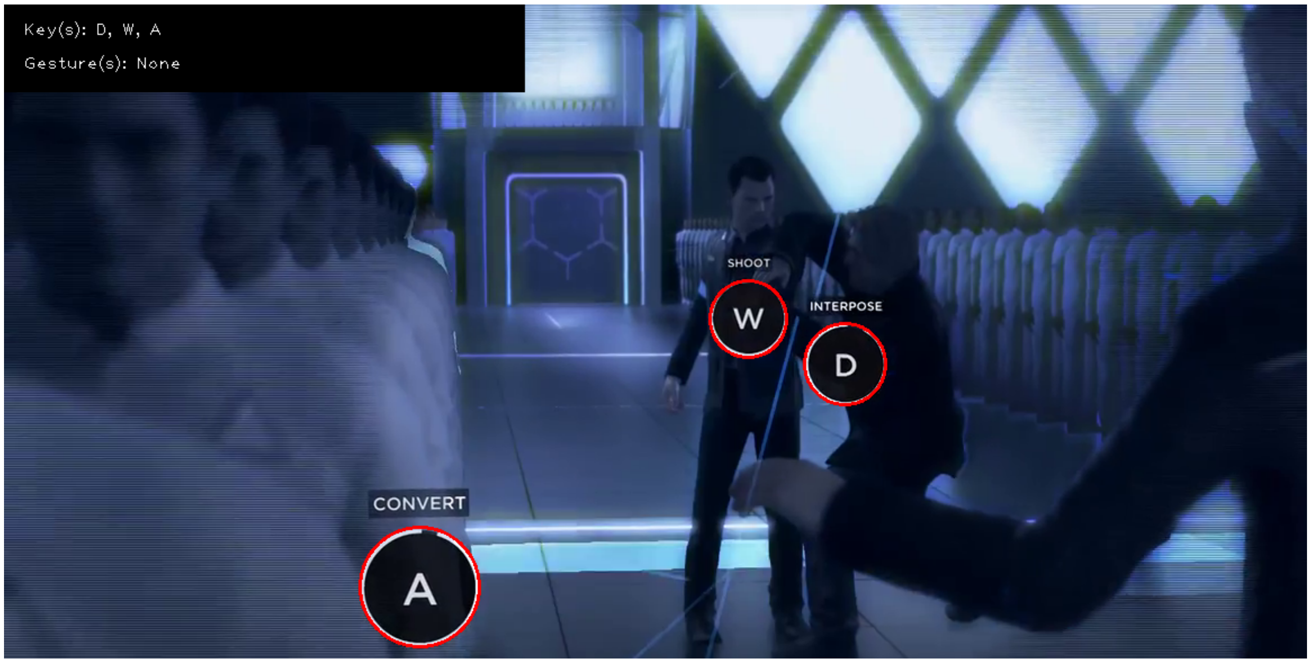

Fig. 5. QTE detector with red circles around the detected QTEs and displaying the three detected QTEs at the top-left.

This paper is the first in terms of academic literature to propose a method for detecting QTEs within a video game.

Additionally, this method overcomes the limitations of prior research. Instead of being able to just separate different symbols, such as in the paper that proposed a method for detecting unknown objects [3], this method is able to detect exactly what the symbol represents, in terms of a QTE action. Furthermore, the paper for detecting characters in SSBM used a customised CNN [4], alongside the article for e-sport statistic tracking which used a Mask R-CNN variation [8], which were both required to be trained. The method proposed here does not make use of neural networks. Therefore, training with the use of high-performance graphics cards was not required as part of the process of creating this QTE detector. Additionally, all QTEs in DBH are moving around the screen as they are displayed, in comparison to the e-sport statistic tracking article [8], of which their method was limited in terms of primarily detecting static objects.

A. *Circles*

The method first detects QTE prompts using a variation of the Hough transformation, the Hough gradient method, for finding circles [12]. This works by applying the Canny edge detector algorithm to an image [7], and using a voting system to determine whether a circle has been detected. To prepare the scene for accurately detecting circles using the Hough transformation, pre-processing was required. A copy of each frame was used where the colours were changed to grayscale, alongside the application of a median blur for removing noise. Parameters were adjusted in order to accurately detect circles outlining the QTEs, instead of any circles within the scene. This included modifying the maximum and minimum radius of the circle, and the minimum distance between QTE circles. Additionally, the upper threshold for the Canny edge detector was set to 250, and the threshold for centre detection at 55. These attributes were modified with regard to the dimensions of the input video, meaning that any video resolution was compatible with the QTE detector. Red circles are displayed around the detected Hough circles, as shown in [2, Fig. 5].

B. *Text*

To start with the text detection using OCR, a copy of the original frame was made with a 65% crop around each detected circle. The frame is cropped for text detection, such that only the portion of the frame containing the QTE text needs to be analysed, and separate circles can be analysed individually. Before the text detection is started, the cropped frame is processed using some of the techniques from an article about improving the accuracy of Tesseract OCR [13].

Firstly, the cropped frame is doubled in size on each axis, using bicubic interpolation. This method of interpolation was a good balance of accurate character recognition and time to compute, when compared to the higher accuracy with cubic B-spline and faster speed with bilinear interpolations [14]. The cropped frame is also Gaussian blurred for noise removal. Additionally, it is eroded, as this produces text thinness that works best with the Tesseract character detection. The frame is also set to grayscale, and then a threshold is applied to convert the frame into a binary image. This makes each pixel either completely black or completely white, determined by said threshold. An example of the final processed frame from a ‘W’ key QTE is shown in Fig. 6.

From here, the processed frame is imported into Tesseract to determine the character corresponding to the QTE key. The

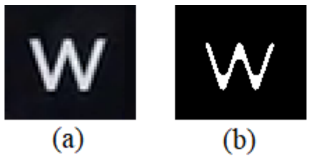

Fig. 6. (a) Cropped frame of a 'W' key QTE. (b) Cropped and processed frame of a 'W' key QTE.

OCR library used, 'tesserocr', integrates with Tesseract's C++ API [11]. The 'tesserocr' library was used due to its fast execution speed and accuracy in recognising text. The detector was originally developed using 'Python Tesseract', which was too slow for stable video playback, due to it being a wrapper of the Tesseract executable [15].

In this instance, Tesseract is set up to only detect single characters, and from a whitelist of possible QTE characters. For example, in DBH, the 'W' key is used for QTEs, but not the 'J' key. Additionally, lowercase forms of the letters are added to the whitelist for some of the QTEs. For example, 'W' could be mistaken for 'w', due to the similarities in their shape. If a QTE key is detected with a confidence of at least 75% from Tesseract, this is then displayed as one of the detected QTE keys.

C. *Symbols*

If no text is found within the cropped frame, symbols are detected using the Harris corner detector [6]. The same Harris formula, shown in eq. (1), and from the paper proposing a framework for video game testing [5], was used to detect corners. Two copies of the original frame are made, each with a 125% and 175% crop around the detected circles. These are used depending on the symbol currently being detected.

To start, the cropped frame is set to grayscale. Then, the frame is analysed to find Harris corners. To retrieve the coordinates of the corners, the returned image is dilated with a threshold applied.

After this, a mathematical combination of the generated Harris corner coordinates is iterated through to find each possible symbol. The number of combinations can be determined using the calculation in eq. (2).

$$\binom{n}{r} = \frac{n!}{r!(n-r)!} \tag{2}$$

In this case, 'n' refers to the number of coordinates, and 'r' is the number of distinct coordinates desired in each combination. Using the example from Fig. 7, there are 10 Harris corner coordinates detected, and 3 distinct coordinates desired. This totals to 120 possible combinations, as shown in eq. (3).

$$\binom{10}{3} = \frac{10!}{3!(10-3)!} = 120 \tag{3}$$

For example, detecting a 'Left' gesture QTE, the larger 175% cropped frame is used as the arrow points out significantly further than the bounds of the circle, as shown in Fig. 7. Additionally, Fig. 7 shows each of the detected Harris corners in red, for a frame cropped with the 'Left' gesture QTE. For the detection of this symbol, 3 distinct coordinates are used from each combination. While each combination of 3 coordinates is iterated through, they are analysed to check whether they correspond to the head of an arrow facing left (a triangle on its side). Additionally, the size of the arrow is taken into account in proportion to the cropped frame, so that QTEs of varying magnitudes can be detected.

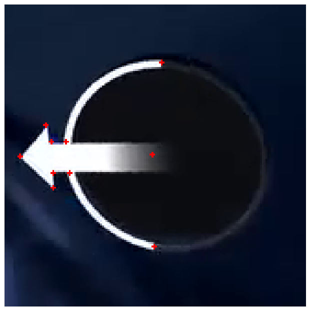

Fig. 7. 'Left' gesture QTE with red dots indicating the detected Harris corners.

Similar processes are performed for the detection of further QTEs from DBH. Most similar are those for detecting the other directional arrows; 'Up', 'Down', and 'Right'. Both the 'Shift' key and a gesture described as 'Centre to right and clockwise to left' are detected using the smaller cropped frame of 125%. Detection of the 'Space' key also makes use of the smaller frame, however, uses only 2 distinct coordinates for each combination. This is to detect a straight horizontal line of a length relative to the cropped frame.

IV. Results

This QTE detector was run on a desktop PC running Windows 10. The PC has an Intel i7-8700 CPU running at a base clock of 3.20 GHz [16], 16 GB of memory at a speed of 2666 MHz. The detector was also developed and run in JetBrain's PyCharm [17] Integrated Development Environment (IDE), using the Python 3.7.3 programming language. Version 4.5.5 of OpenCV [10] was used alongside version 2.5.2 of 'tesserocr' [11].

While the captured footage of DBH was at a 1440p resolution at 60 Frames Per Second (FPS), this was not usable due to the slowness experienced in using such high fidelity footage. The files were converted in Handbrake [18] to a 720p resolution at 30 FPS. The converted files were then used for importing into the QTE detector, which was a good balance between framerate playback stability and high resolution for this PC configuration.

To analyse the effectiveness of the detector, footage from DBH was cut and converted for use. This totalled to 3 minutes and 27 seconds of footage. The footage contains snippets of scenes from the game with an abundance of QTEs. In total, 93 QTEs were analysed. Of these, 73.1% were correctly detected, 19.4% were partially correct, and 7.5% were incorrect, as shown in Fig. 8. In this case, partially correct means that the

QTE was detected correctly during one or more frames, but during at least one other frame it was incorrectly detected.

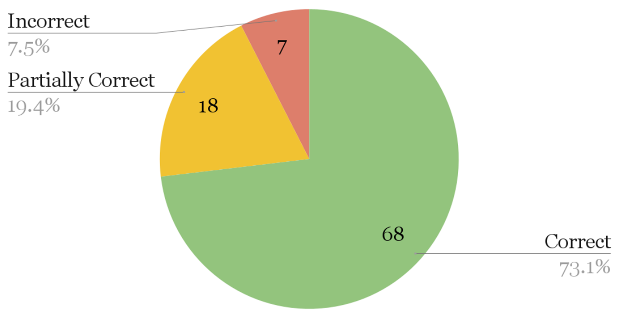

Fig. 8. A graph showing the number and percentages of correctly, partially and incorrectly detected QTEs.

As shown in Fig. 9, the detection of the ‘S’, ‘Centre to right and clockwise to left’, ‘Shift’, and ‘Right’ QTEs were always correct in the testing. There’s no discernible pattern here, as these all cover a wide variety of the techniques used for QTE detection. Whereas the ‘W’ QTE was only correctly detected 18% of the time. Additionally, the ‘Up’ QTE was detected in just 50% of instances.

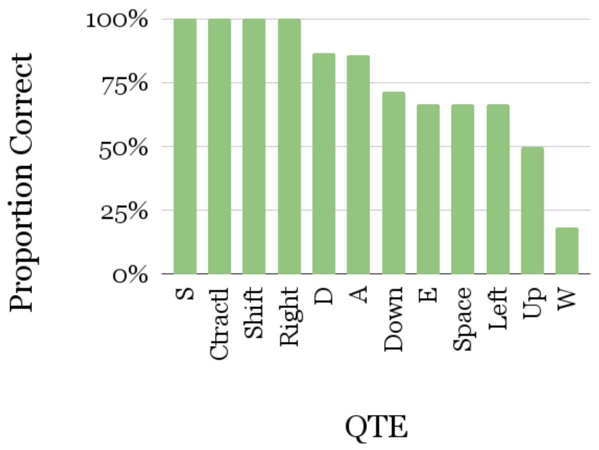

Fig. 9. A graph showing the proportion of each QTE being **correctly** detected. ‘Ctractl’ refers to the QTE ‘Centre to right and clockwise to left’.

Fig. 10 shows the proportion of keys that were partially detected. The outlier in this graph is the ‘W’ QTE, which was partially detected 82% of the time. An explanation for this is shown in the fact that the ‘W’ QTE was often confused with ‘A’ in 9 of the 11 cases that the QTE appeared during the analyses. Additionally, ‘W’ was mistaken for the ‘Space’ and ‘Shift’ QTEs, often while also being mistaken for ‘A’ throughout its duration in view. The default implementation of the Tesseract OCR was limited in this case, which should have been pre-trained with the processed and cropped images generated from the QTE detector in order to improve its accuracy.

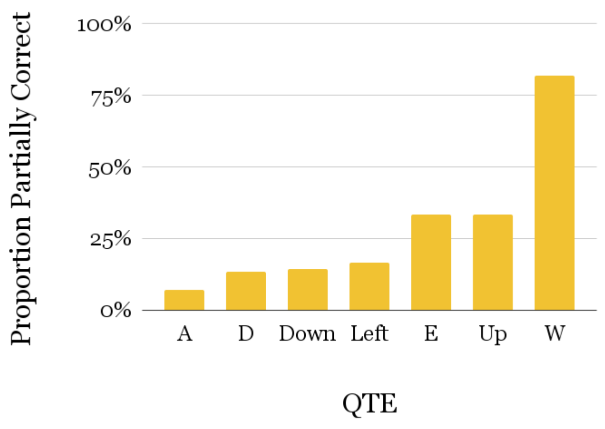

Fig. 10. A graph showing the proportion of each QTE being **partially** detected. QTEs that were never partially detected have been excluded from the graph.

The ‘Space’ QTE was incorrectly detected the most out of any other QTE, as shown in Fig. 11. This limitation was likely due to the fact that only two Harris corner coordinates were used in the detection of this QTE symbol. Using only two coordinates meant that the tolerances for shape changes due to orientation shifts lead to the ‘Space’ QTE being unable to be detected.

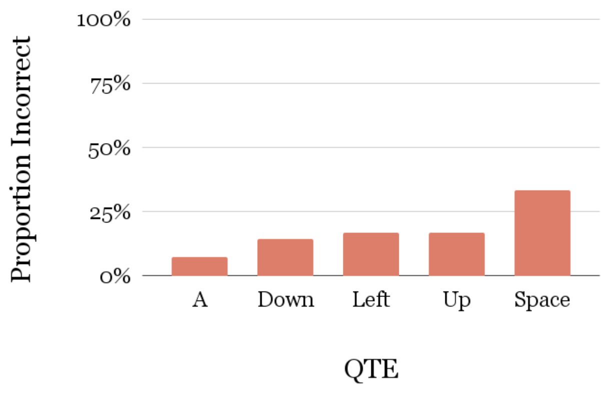

Fig. 11. A graph showing the proportion of each QTE being **incorrectly** detected. QTEs that were never incorrectly detected have been excluded from the graph.

The biggest limitation of the QTE detector is that it only demonstrably works with footage from DBH. In theory, there could be a game using similar QTE icons in which the detector would work, however, the detector was designed solely for DBH. Even then, only a subset of the numerous QTEs found within the game are able to be detected. Additionally, with QTEs only being correctly detected only 73.1% of the time, the detector cannot be relied upon for accurately detecting every instance of a QTE in DBH.

Regarding its usability in DBH, the QTE detector is limited in the fact that it is unable to run in real-time. The reason for this is that whenever text is not detected, each symbol is iterated through in order to check for a match. Furthermore, the number of coordinate combinations exponentially increases with each Harris corner detected. The detector could be adapted to work with live game-play, however, not every frame would be analysed, even with a 30 FPS capture.

V. Conclusion

The proposed QTE detector managed to correctly detect 73.1% of QTEs, partially detect 19.4%, and incorrectly detect 7.5%, as shown in Fig. 8. This accuracy is slightly higher than the prior research for detecting characters in SSBM, which reached 68.25% [4]. Unfortunately, most of the outcomes from the discussed prior research were not quantified, likely due to the limited research into the detection of objects in video games. Further conclusions include the lack of high confidence and accuracy of the proposed method, as opposed to the method attained in the article discussing the real-time tracking of e-sport games, as proven by its usage in an API provided by Abios Gaming [8]. However, the method in this paper tracked QTEs that were all moving around the screen, rather than the static elements found within e-sports. Additionally, the proposed method does not make use of a CNN, such as in the e-sport article [8] and the paper for detecting characters in SSBM [4]. As well as this, instead of being able to just separate different symbols, such as in the paper using qualitative stability analysis [3], this method detects the exact objects, being QTEs, and what they represent.

The usage of Tesseract OCR was limited due to it detecting some characters incorrectly. The e-sport tracking article specifically mentioned the requirement for training on large bodies of synthetic text to attain accurate results, compared to the default Tesseract implementation, which is what was used for the proposed QTE detector [8]. Additionally, the symbol detection was limited, most notably regarding the 'Space' QTE. The proposed method is also unable to run in real-time, with each frame analysed. The detector is also limited to footage from DBH using a subset of the game's QTEs, and even then, the accuracy of 73.1% cannot be relied upon.

A. Future Work

To increase the accuracy of detecting QTEs that contain characters, additional training past the default implementation of the Tesseract OCR should be conducted. Both the accuracy and speed of the QTE detector could be increased with improvements to the symbol detection. While additional Harris corner coordinates could be analysed for each symbol in order to increase the accuracy of the QTE detector, this would exponentially decrease its speed. Using a CNN for detecting symbols, similar to that found within the paper for detecting characters in Super Smash Brothers Melee [4], could increase both the accuracy and speed of the detector. In this case, individual points would not need to be examined, which have an exponential complexity with each coordinate analysed. This could be taken further by expanding the detector to work with more games, and across multiple input methods such as gamepads, which can have differing QTE labels.

References

[1] J. Aguado-Delgado, J.-M. Gutiérrez-Martínez, J. R. Hilera, L. de Marcos, and S. Otón, "Accessibility in video games: a systematic review," *Universal Access in the Information Society*, vol. 19, no. 1, pp. 169–193, Mar 2020. [Online]. Available: https://doi.org/10.1007/s10209-018-0628-2

[2] "Detroit: Become Human," May 2018, accessed 19 Apr 2022. [Online]. Available: https://www.quanticdream.com/en/detroit-become-human

[3] X. Ge, J. Renz, and P. Zhang, "Visual detection of unknown objects in video games using qualitative stability analysis," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 2, pp. 166–177, 2016.

[4] A. Venkatesh, "Object tracking in games using convolutional neural networks," Jun 2018.

[5] A. Nantes, R. Brown, and F. Maire, "A framework for the semi-automatic testing of video games," *Proceedings of the 4th Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2008*, Jan 2008.

[6] C. G. Harris and M. J. Stephens, "A combined corner and edge detector," in *Alvey Vision Conference*, 1988.

[7] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[8] G. Silvestri and P. Sharif, "Enabling real time e-sport tracking with streaming video object detection," Oct 2020, accessed 25 Apr 2022. [Online]. Available: https://modulai.io/blog/enabling-real-time-e-sport-tracking-with-streaming-video-object-detection/

[9] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. [Online]. Available: http://arxiv.org/abs/1703.06870

[10] OpenCV, "OpenCV on Wheels," Mar 2022, accessed 16 Apr 2022. [Online]. Available: https://github.com/opencv/opencv-python

[11] Sirfz, "tesserocr," Jun 2021, accessed 16 Apr 2022. [Online]. Available: https://github.com/sirfz/tesserocr

[12] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Commun. ACM*, vol. 15, no. 1, p. 11–15, Jan 1972. [Online]. Available: https://doi.org/10.1145/361237.361242

[13] K. Kuguoglu, "How to use image preprocessing to improve the accuracy of tesseract," Jun 2018, accessed 16 Apr 2022. [Online]. Available: https://towardsdatascience.com/getting-started-with-tesseract-part-ii-f7f9a0899b3f

[14] D. Han, "Comparison of commonly used image interpolation methods," in *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013)*. Atlantis Press, Mar 2013, pp. 1556–1559. [Online]. Available: https://doi.org/10.2991/iccsee.2013.391

[15] "Pytesseract," Feb 2022, accessed 18 Apr 2022. [Online]. Available: https://pypi.org/project/pytesseract/

[16] "Intel® Core™ i7-8700 processor," Oct 2017, accessed 18 Apr 2022. [Online]. Available: https://www.intel.com/content/www/us/en/products/sku/126686/intel-core-i78700-processor-12m-cache-up-to-4-60-ghz/specifications.html

[17] "PyCharm: The Python IDE for Professional Developers by JetBrains," Apr 2022, accessed 18 Apr 2022. [Online]. Available: https://www.jetbrains.com/pycharm/

[18] "HandBrake: Open Source Video Transcoder," Jan 2022, accessed 18 Apr 2022. [Online]. Available: https://handbrake.fr/