

Q3Vision: A Vision-Based Framework for Game-State Reconstruction in Quake III Arena

Duc-Minh Pham
Telecom Research Unit
University of Technology Sydney
Sydney, Australia
Duc.Pham@uts.edu.au

Jaehye Kim
Telecom Research Unit
University of Technology Sydney
Sydney, Australia
jaehye.kim-2@student.uts.edu.au

Daniel Franklin
Telecom Research Unit
University of Technology Sydney
Sydney, Australia
Daniel.Franklin@uts.edu.au

Mehran Abolhasan
Telecom Research Unit
University of Technology Sydney
Sydney, Australia
mehran.abolhasan@uts.edu.au

Raymond Owen
Telecom Research Unit
University of Technology Sydney
Sydney, Australia
Ray.Owen@uts.edu.au

Abstract

Computer vision offers new opportunities for understanding gameplay and player behavior in esports and interactive digital media. However, the performance of modern vision models in legacy first-person shooter (FPS) environments, characterized by rapid camera motion, motion blur, occlusion, and stylized heads-up display (HUD) elements, remains insufficiently studied. This paper presents Q3Vision, a vision-based framework for reconstructing game-state information in *Quake III Arena* using only screen-captured video. The system used YOLO-based object detection to identify enemies and interactable items, together with OCR-based extraction of player-status indicators such as health, armour, ammunition, and weapon state. A reproducible OBS-based data acquisition pipeline enables frame-level alignment of visual data and supports evaluation across varied gameplay conditions. A reproducible data acquisition pipeline based on Open Broadcaster Software (OBS) enables frame-level alignment of visual data and supports evaluation across varied gameplay conditions. Experimental results demonstrate that vision-only state reconstruction is feasible, though OCR performance is limited by low-resolution, stylized HUD fonts and visual noise. The findings highlight both the potential and the practical constraints of applying computer vision techniques to fast-paced FPS gameplay, informing future work on domain-specific OCR fine-tuning, temporal modeling, and multimodal extensions.

Keywords

Quake III, First Person Shooting Game (FPS), YOLO (You only look once), Optical Character Recognition (OCR), EasyOCR, Heads-Up Display (HUD), Open Broadcaster Software Studio (OBS Studio)

ACM Reference Format:

Duc-Minh Pham, Jaehye Kim, Daniel Franklin, Mehran Abolhasan, and Raymond Owen. 2026. Q3Vision: A Vision-Based Framework for Game-State Reconstruction in Quake III Arena. In *2026 Australasian Computer Science*

Week (ACSW 2026), February 09–12, 2026, Melbourne, Australia. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3793811.3793821>

1 Introduction

Gameplay analysis is a vital way to help players understand their decision, mistakes made, and what can be done better in the next round. Similar to having coaches assisting professional athletes' growth in competitive sports, esports players also have specialized assistants to analyze and hone player's strategy and skills during worldwide competitions. However, this is not a luxury that could be enjoyed by casual or amateur gamers. Players often evaluate their skills and decisions by watching recorded video of game sessions. However, it is very difficult for average or less experienced players to analyze their weaknesses and improve their gameplay. Previous work indicates that traditional tools lack interpretability for gameplay data, and that gamers struggle to translate such outputs into actionable insights for performance improvement [3].

While machine learning and data-driven technologies have been widely applied in the gaming industry, the use of artificial intelligence for analyzing player performance and providing recommendations remains limited. Such analysis could provide both average and elite players with specific technical feedback, enabling targeted training to improve performance. Mogavi et al. demonstrate that player behavior can be predicted and interpreted through gameplay analysis, highlighting the potential for tailored feedback and in-game learning strategies [7]. However, this research domain remains underutilised in practice.

A common approach to analyzing player performance and gameplay dynamics is to rely on event logs generated by the game engine. While such logs provide structured records of major events and state transitions, they often offer an incomplete representation of the gameplay environment. In particular, engine logs typically omit important visual context, including player status indicators such as health, armour, ammunition levels, and the on-screen information available to the player at a given moment. As a result, analyses based solely on log data may fail to capture the situational factors that influence player decisions and in-game actions.

This work addresses the challenge of missing gameplay telemetry and context by leveraging computer vision methods to extract information directly from the game screen. In addition to textual



This work is licensed under a Creative Commons Attribution 4.0 International License. *ACSW 2026, Melbourne, Australia*

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2315-5/26/02

<https://doi.org/10.1145/3793811.3793821>

log data, visual data provides continuous cues about player status and scene context that are otherwise unavailable. By applying YOLO (You Only Look Once) for object detection and OCR (Optical Character Recognition) for parsing numerical HUD elements, the system recovers key aspects of the game state that are visible during gameplay but not recorded in conventional game logs.

Building on this motivation, Q3Vision is designed specifically for fast-paced first-person shooter (FPS) games, where rapid viewpoint changes, motion blur, stylized HUD elements, and dense visual effects make vision-based state extraction particularly challenging. Prior work in the analysis of FPS environments has examined object detection under extreme motion [10, 16], as well as visual analytics for esports and game understanding [5, 19]. However, these studies typically focus on event recognition or high-level tactical insights rather than reconstructing fine-grained numerical and spatial state directly from screen-captured footage. Q3Vision extends this line of research by targeting detailed state reconstruction in an FPS setting, providing a vision-only alternative to engine-level telemetry.

This paper makes the following contributions:

- We propose *Q3Vision*, a vision-based framework that reconstructs gameplay state information missing from conventional game logs by combining object detection, and OCR-based HUD parsing.
- We demonstrate that modern computer vision models (YOLO and OCR engines) can recover fine-grained player status and environmental entities in fast-paced FPS gameplay without access to internal game telemetry.
- We evaluate the feasibility and limitations of vision-only telemetry recovery across diverse gameplay conditions, highlighting challenges introduced by motion blur and stylized HUD elements.

The remainder of this paper is organized as follows. Section 2 reviews related work and provides the background for this study. Section 3 outlines the system design and data acquisition pipeline. Section 4 details the vision processing modules for object detection and HUD parsing. Section 5 presents the experimental results and discussion. Section 6 concludes the paper and outlines directions for future work.

2 Related Work

Research on computer vision for fast-paced games has explored object detection and action recognition under challenging conditions such as rapid motion and visual clutter. While these studies demonstrate the applicability of modern vision models to dynamic gameplay, they generally focus on event classification rather than reconstructing detailed game state. Complementary work in visual analytics and esports has examined automated highlight detection, player behavior analysis, and tactical inference from video, but typically relies on broadcast footage or simplified visual cues. Unlike prior work, this paper combines spatial detection and HUD parsing to reconstruct a fine-grained game state directly from screen-captured video.

One of the state-of-the-art algorithms for object detection tasks is YOLO. YOLO is an object detection model that transforms the traditional multi-stage detection methodologies, such as R-CNN, into a single, unified convolutional neural network [15]. Instead

of a region-based approach, it is achieved by framing the task as a single regression problem, predicting bounding boxes and classes of the entire image in one evaluation. Although YOLOv1 developed by Joseph Redmon et al. has suboptimal performance compared to traditional methods such as Faster-RCNN, it compensates in its model efficiency and simplicity [15]. The YOLO model is also able to generalize better than region-based models on more diverse datasets, even on its oldest version.

YOLO is also a rapidly growing model, progressing in terms of efficiency and accuracy. Jiang’s work reviewed the development from YOLOv1 to YOLOv5 [9], which highlights the technical improvements and design changes in each version. It is shown that later versions of YOLO show significant improvements in performance compared to its previous version. YOLOv4 for instance, improves significantly compared to YOLOv3 due to adding built-in data augmentation using Bag-of-Freebies and Bag-of-Specials technique. Although earlier versions underperformed compared to traditional methods, later versions have substantially closed the gap, achieving faster prediction times and comparable accuracy, as demonstrated by Youssouf et al., who compared YOLOv4 with Faster R-CNN for traffic sign classification [20]. Results show that YOLOv4 outperforms faster R-CNN with a mAP of 59.88% and 35 FPS, as compared to Faster R-CNN’s mAP of 43.26% and 6 FPS [20].

A popular method for tackling character recognition tasks is through OCR models. Mori’s research reviews the development of OCR methods [12]. OCR originally detects characters by template matching and feature extraction techniques such as contour following and background analysis, then later improved significantly by incorporating machine learning techniques and neural networks. However, OCR models still struggled with lack of multi-lingual support, detecting stylized text and noisy background, which could be a potential challenge for this project due to the low resolution of the Open Arena game environment.

One of the more modern versions of the OCR model is EasyOCR [1], an open-source Optical Character Recognition (OCR) tool built using PyTorch by the Jaided AI team. EasyOCR’s recognition model is based on the CRNN inspired by the work of Shi [17], which is a robust architecture combining ResNet feature extraction, LSTM sequencing, and CTC decoding. This backbone model is the main reason for EasyOCR being a powerful model, since it also considers sequences of characters instead of a simple classification. The effectiveness of this model is complemented by the fact that it could be easily plugged into any ML applications by installing and importing it as a Python library. As a result, the EasyOCR model becomes the chosen model for extracting player status information.

In prior work, Jiayu et al. developed a moving-target detection algorithm for FPS games by training a YOLOv3 model, demonstrating that such systems can evade traditional anti-cheat detection methods [11]. The model is able to detect enemies with almost 99% accuracy. Cheng also implemented YOLOv5 and YOLOv7 for a similar use case [4]. The research captures game frames from the FPS game CS:GO and annotates the images manually to form a custom dataset. The dataset is then being fitted into YOLOv5 and YOLOv7 for enemy character detection. Results show that both YOLO models perform effectively in real-time character detection in FPS games and indicate suitability for developing intelligent aimbot or assistive systems for gamers. Together, these two papers

motivate an approach in which a YOLO model is trained to detect enemies and items before extracting the relevant information for subsequent data processing.

OCR models are also being applied in video games. Ninagan developed a desktop application that detects and translates on-screen text in video games and displays the text directly back on the original text's location using EasyOCR [13]. To translate text, the user selects the region of interest, and the text content is extracted using OCR, the extraction information is then passed to Google translate API for transcription. The results of Ninagan validates the hypothesis of the project that OCR could be leveraged for extracting game text information, which includes player status data.

Another application of OCR can be seen in the work of Garcia et al. [6]. In this study, an automated pipeline was proposed to extract structured information from basketball video game footage using machine learning methods. The pipeline involves the use of YOLOv8 for detecting objects such as players and basketball, and leverages EasyOCR for jersey number recognition to uniquely identify each player. Although YOLOv8 excels in detecting the different objects with a mAP@0.5 of 0.83, EasyOCR is underperforming due to partial occlusion, small fonts, and scene complexity. This paper is an example of how video game data could be collected by combining YOLO and OCR models and also provides insights on the potential limitations for EasyOCR.

Thanapong et al. proposed a custom-built 3D CNN-based model, DeepLogger, which generates user input logs through frame-by-frame analysis of gameplay footage [8]. The model performs with a multi-label accuracy of 79% on Tetris and 70.6% on Mega Man X. Although applying this approach to modern video games is challenging due to increased environmental complexity, the study demonstrates the feasibility of using computer vision techniques to extract gameplay data and uncover latent behavioral insights.

3 System Design and Data Acquisition Pipeline

3.1 Problem Definition and Design Objectives

This work aims to reconstruct game-state information from screen-captured video using a vision-only approach. The system must operate under the visual constraints of fast-paced FPS gameplay, including rapid camera movement, motion blur, and stylized HUD elements. Key objectives include accurate detection of gameplay entities, reliable extraction of HUD values, and a reproducible data acquisition process. The game frames are then being extracted and annotated for detecting what the player has seen in the gameplay environment, including enemies and items.

Figure 1 shows a sample of the annotated image, which shows two annotated enemies. The detected entities will be either items or enemies. To generate a suitable dataset for object detection training, 5 different classes are defined to build the object detection model for detecting the presence of enemies and items, including weapon, armour, health, ammo, and enemy. Moreover, since player status such as ammo and health are shown in fixed locations on the screen. For example, the number 10 is the number of ammunitions, 92 is the player health points, and 100 refers to player armour points. These numbers can be extracted and identified with machine learning methods such as OCR. To prepare sufficient game frames for



Figure 1: Annotated image include annotated enemies.

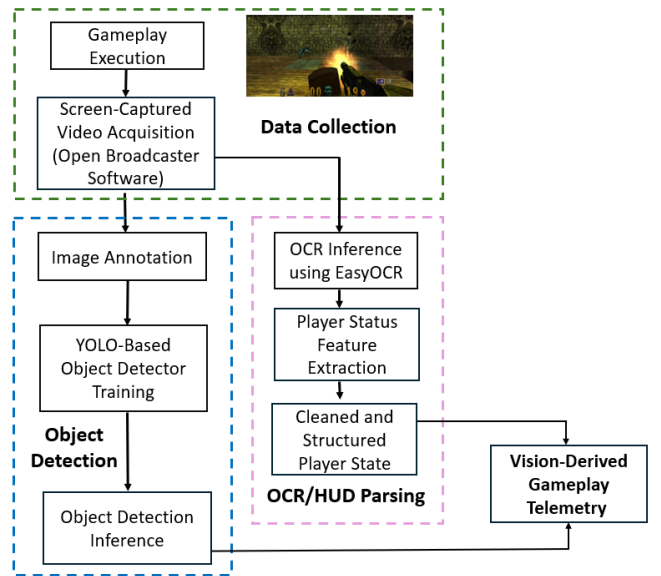


Figure 2: System Architecture Overview.

the computer vision dataset, game frames will be extracted and annotated while playing the game.

3.2 System Architecture Overview

The proposed framework consists of three main components:

- **Data Collection:** a video acquisition pipeline that records gameplay using OBS Studio.
- **Object Detection:** an object detection module that identifies enemies and interactable items.
- **OCR/HUD Parsing:** an OCR-based HUD parser that extracts player health, armour, ammunition, and weapon status.

Figure 2 illustrates the overall data flow from input video to structured game-state output.

The first component focuses on non-intrusive visual data acquisition, which provides the sole input to the framework and enables applicability to legacy FPS games without engine-level instrumentation. The second and third component performs vision-based state

extraction, including object detection and OCR-based HUD parsing to recover spatial and numerical cues from gameplay footage.

The final output of the Q3Vision framework is a vision-derived gameplay telemetry representation that combines player-status information and scene context on a per-frame basis. This representation integrates numerical HUD indicators, such as health, armour, and ammunition, with spatial information about detected in-game entities. By reconstructing gameplay state directly from visual input, the resulting telemetry enables detailed analysis of player behavior and game dynamics in scenarios where engine-level logs are incomplete, unavailable, or lack sufficient contextual detail.

3.2.1 Video Acquisition and Input Characteristics. Video acquisition constitutes the input layer of the Q3Vision framework and is critical for enabling vision-only reconstruction of gameplay telemetry. At the start of each gameplay session, a recording script is executed on the player’s machine to capture the on-screen video stream. The script records each frame rendered by the game together with a corresponding timestamp, enabling precise temporal alignment with extracted visual features. This process yields two complementary data sources: the raw textual logs generated by the game engine and the sequence of screen-captured frames containing player-status information and visible in-game entities.

Gameplay footage is captured using OBS Studio at predefined resolutions and frame rates (30 and 60 frames per second) to ensure consistent visual inputs for downstream processing. Different recording configurations, such as variations in field-of-view, resolution, map layout, and movement intensity, are incorporated to assess robustness across diverse gameplay conditions. These variations capture realistic visual challenges encountered in competitive FPS environments.

3.2.2 Annotation and Reproducibility Considerations. Annotation supports the vision-based components of Q3Vision by enabling both model training and systematic evaluation. A subset of screen-captured frames is manually annotated to facilitate the training and evaluation of both the object detection and OCR components. For the YOLO-based detector, annotations consist of bounding boxes identifying enemies and interactable items. For the OCR module, HUD regions containing health, armour, ammunition, and weapon indicators are cropped and labeled with their corresponding ground-truth values. All recording configurations, preprocessing steps, and annotation guidelines are documented to ensure reproducibility and to support future extensions of the dataset and model training process.

4 Vision Processing Modules

4.1 Object Detection

The YOLO features dataset is a cleaned dataset extracted from the objects detected through a fine tuned YOLO model. The steps for preparing such dataset are as follows.

Firstly, game frames are extracted from the trial game runs to collect in-game images for creating data annotations. Around 4000 images are collected and annotated in this process. The raw images are then uploaded to the annotation website *MakeSenseAI* for creating the annotations manually with five different classes: weapon, enemy, armour, health, and ammo [18]. By using *MakeSenseAI*, the

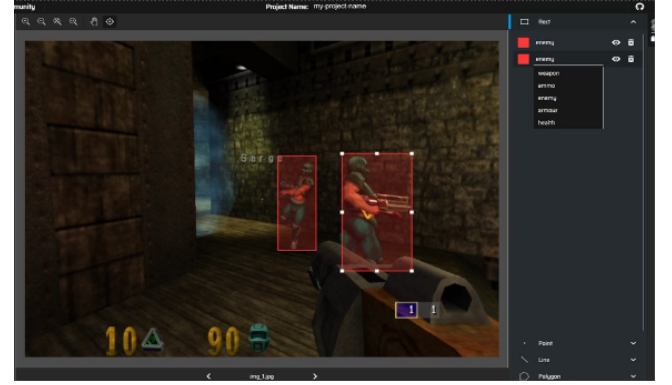


Figure 3: Annotated image from makesense.ai.

Table 1: Key Features for YOLO Object Detection Event

Attributes	Description
raw_timestamp	Timestamp of the event
map	Placeholder value, to be completed in final dataset
game_round	Placeholder value, to be completed in final dataset
event_type	Type of event (YOLO only in this CSV)
event_details	Details of YOLO detection event

annotation process becomes more efficient, as the platform provides an intuitive and easy-to-use interface. As illustrated in the Figure 3, users can simply drag bounding boxes over the target objects and assign labels during dataset construction. The annotated images are then exported in YOLO annotated format. The YOLO model is then being fine tuned with the newly annotated dataset. The reason YOLO is being used is its capability in real time detection by a single forward pass of the neural network while maintaining high accuracy, when compared to other object detection methods such as Fast RCNN [20]. This makes YOLO suitable for this scenario since video streams are being processed. Another reason YOLO is chosen is its global context understanding ability [15]. Unlike its counterpart RCNN which uses a region bounding box, YOLO processes the entire image at once, and hence understands context information better. This property allows YOLO to be less susceptible to false positives and background noise.

After the YOLO model is trained, it is being placed into a trial game video different from the one used to collect the image annotation dataset. The key features, including the detected entity and the position of bounding boxes, are being extracted and stored as in CSV format. This CSV data extracted by YOLO will later be combined with the cleaned textual data and OCR extracted insights to form the final output dataset. Table 1 and Table 2 below show the key features and details extracted from the trained YOLO model.

Table 2: YOLO Detection Event Attributes

Attributes	Description
player_ip	IP of the player character
yolo_obj	Type of YOLO object detected
confidence	Detection confidence score
x	X-coordinate of the top-left corner of the bounding box
y	Y-coordinate of the top-left corner of the bounding box
width	Width of the bounding box
height	Height of the bounding box
box_area	Area of the bounding box
center_x	X-coordinate of the box center
center_y	Y-coordinate of the box center
diff_height	Difference in bounding-box height from previous frame
position_of_object	Array storing [x, y, width, height]

Table 3: Key features for OCR detection event

Attributes	Description
raw_timestamp	Timestamp of the event
map	Placeholder value, to be completed in final dataset
game_round	Placeholder value, to be completed in final dataset
event_type	Type of event (OCR in this CSV)
event_details	Details of OCR detection event

4.2 HUD Parsing and Numerical State Extraction

This section discusses how the cleaned OCR data is being detected. As mentioned above, player status such as health and armour locates at a fixed position in the game screens. This means instead of performing object detection to locate the player status, hardcoded cropping can be applied to the game screen to extract the status data. This status data will then be passed into the optical character recognition model EasyOCR for number extraction.

EasyOCR is being adopted due to it being an open-sourced and fully supported Python library that could easily be used in extracting player status data without any training and finetuning [1]. It is also a lightweight and fast model that works without GPU, making it a suitable model for processing video streams, which could be computationally intensive when building multi-model systems around it. Moreover, EasyOCR supports stylized format due to its deep neural network structure, which would be useful for detecting the game hub text.

After the game frames are being passed into the EasyOCR model. The features and event metadata are being cleaned and extracted for further processing, as shown in Table 3 and Table 4 respectively.

The OCR processing pipeline consists of four main stages: cropping fixed HUD regions from the game frame, applying OCR inference to the cropped images, extracting numerical features from the

Table 4: Event details for OCR detection event

Attributes	Description
player_ip	Player IP address
pred_roi_ammo	Extracted Ammo value
conf_roi_ammo	Confidence of ammo recognition
pred_roi_health	Extracted health value
conf_roi_health	Confidence of health point recognition
pred_roi_armour	Extracted armour value
conf_roi_armour	Confidence of armour point recognition

OCR outputs, and performing post-processing and data cleaning to improve temporal consistency.

To mitigate issues arising from low image resolution and limited contrast between HUD text and background, the cropped images undergo preprocessing prior to EasyOCR inference. The cropped images are preprocessed using contrast and intensity scaling via the `convertScaleAbs` function provided by OpenCV [2] to improve character visibility. Following intensity normalization, the cropped HUD regions are resized using linear interpolation to increase their effective resolution prior to EasyOCR inference. This upscaling step improves character legibility by reducing aliasing effects and providing finer-grained pixel information, which in turn enhances the robustness of numerical recognition in low-resolution HUD elements.

The processed images are then passed to the EasyOCR model, and the raw predictions, along with confidence scores and timestamps, are stored in a raw OCR output dataset in CSV format as shown in Table 4.

5 Results and Discussion

5.1 Object Detection Performance

We evaluate the accuracy of the YOLO-based object detector across multiple gameplay scenarios, including variations in resolution, field-of-view, and player movement speed. Metrics such as mAP, precision, recall, and IoU [15] are reported. The trained YOLO model for extracting objects and enemies from the Quake III game is evaluated by mAP (mean average position), average precision of each class, recall and F1 score [15]. The meaning of IoU (intersection over union) is used as the threshold for determining whether a predicted box counts as a correct detections [15]. Real time testing was also be done by loading video data into the YOLO model to predict the correct labels.

Table 5 shows the evaluation results of the YOLO model. It is shown that the model scores a mAP@0.5 for 91.2%, recall of 95% and F1-score of 91% . The model excels in predicting weapon, ammo and enemy, achieving an accuracy of more than 90% (Table 6). It is slightly underperforming in predicting armour and health when compared to the first three classes, with an average precision of 83.3% and 88.1%. Overall the YOLO model shows excellent performance in detecting what the player has seen in the game environment.

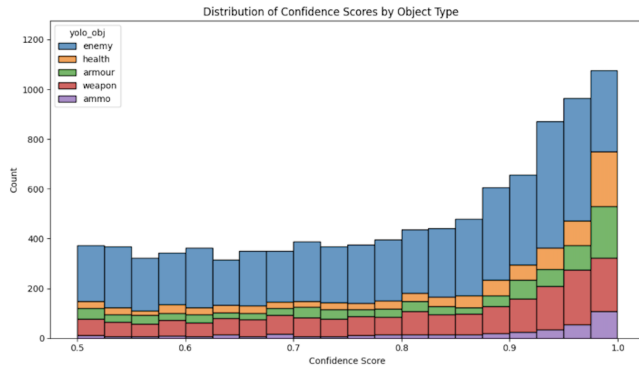
Figure 4 shows the detection confidence distribution for the trained YOLO mode. It is shown that the model has a high confidence in detecting all five classes, reflected by the relatively higher

Table 5: Overall evaluation metrics of the trained YOLO model.

Metric	Value
Detections count	345
Ground-truth count	173
Precision (conf = 0.25)	87%
Recall (conf = 0.25)	95%
F1-score (conf = 0.25)	91%
Average IoU	67.29%
IoU threshold	50% (AUC per recall)
mAP@0.50	91.20%

Table 6: Per-class average precision and detection performance of the YOLO model.

Class ID	Name	AP (%)	TP / FP
0	weapon	94.79	23 / 6
1	ammo	92.44	13 / 1
2	enemy	97.37	101 / 10
3	armour	83.30	15 / 5
4	health	88.10	12 / 3

**Figure 4: YOLO detection confidence distribution.**

number of detections counts from 0.9 to 1.0. It can also be seen that the most detected class is enemy.

Figure 5 shows a screenshot of the detection results from the YOLO model. YOLO can clearly identify the three enemies spotted in the gameplay environment with three bounding boxes with a high confidence level. The trained YOLO can also detect enemies in overlapping bounding boxes. This shows that the YOLO model is generalizing well to the game environment.

5.2 OCR Accuracy Under HUD Variations

HUD parsing performance is analyzed across conditions involving lighting changes, weapon flashes, motion blur, and low-resolution digit rendering. Character and sequence-level accuracy metrics highlight how reliably the system extracts health, armour, ammunition, and weapon indicators. The impact of preprocessing and temporal smoothing is also evaluated to quantify improvements under visually challenging scenes.

**Figure 5: YOLO Detection Results.****Figure 6: OCR Detection Results.**

The effectiveness of the EasyOCR model is evaluated by real time testing. The playback video is loaded into the model for EasyOCR to perform predictions. After the prediction is done, the results are being embedded into the game frames to be cross checked with the ground-truth player status. Figure 6 shows a snapshot of the OCR recognition result. It is shown that player health, armour, and ammo are being correctly identified and shown in the top left corner. When going through all evaluation frames, it is discovered that the model is able to perform correct prediction on all three metrics in 90% of the frames.

5.3 End-to-End Game-State Reconstruction

This subsection integrates detection and OCR outputs to assess the overall fidelity of reconstructed game-state information. We analyze how well the system approximates the true player state over time, identifying sources of error such as inconsistent HUD readings, lost tracks, or missed detections. If internal telemetry is available, we compare the reconstructed state against the ground truth to measure the completeness and temporal alignment of the final output.

Figure 7 illustrates the combined detection results of the YOLO and OCR models applied to a sample gameplay sequence. The results show that the YOLO model successfully identifies both the number and types of objects present on the screen, while the OCR module accurately recognizes all visible player-status indicators. The visualized predictions demonstrate that both spatial detections and numerical HUD values can be directly extracted from model inference and stored in structured form for subsequent analysis,



Figure 7: Detection results from the playback video.



Figure 8: YOLO detection failure case I.

visualization, and integration into the game-state reconstruction pipeline.

5.4 Failure Modes and Observed Limitations

Although Q3Vision successfully reconstructs key elements of the game state, several limitations arise from the visual complexity of fast-paced FPS gameplay. Extreme motion blur, rapid camera rotation, partial occlusion, and intense visual effects such as explosions or muzzle flashes can degrade detector confidence. HUD elements may also be partially obscured or rendered with low contrast, leading to inconsistent or noisy numerical readings. These challenges reflect the broader difficulty of applying off-the-shelf computer vision models to highly dynamic gaming environments, and they highlight opportunities for further refinement through domain adaptation, temporal modeling, or specialized preprocessing.

One limitation of the YOLO model is its difficulty in detecting objects that are positioned closely together without a visible gap. As shown in the Figure 8, the model detects the purple-box object, which has some separation, but fails to detect the overlapping object highlighted in the red box.

A second failure case occurs when YOLO is unable to distinguish the object from the background due to their similar colours. In the Figure 9, the model correctly detects the ammo next to the health item, but it fails to differentiate the health item from the background.



Figure 9: YOLO detection failure case II.



Figure 10: OCR detection failure case I.

A more specific limitation concerns the OCR component used for parsing HUD values. In this work, player-status extraction relies on directly applying EasyOCR without any fine-tuning or domain-specific training. As a result, the model struggles with stylized, low-resolution game fonts and the non-uniform backgrounds typical of Quake III and OpenArena. OCR models are especially sensitive to character deformation, low contrast, and irregular edge patterns, all of which are common in FPS HUD elements. Since EasyOCR has not learned the font characteristics and rendering artifacts of this environment, it exhibits generalization errors that lead to misread digits or unstable frame-by-frame predictions. These issues suggest the need for targeted OCR fine-tuning, improved preprocessing, or custom training on annotated HUD imagery.

Figure 10 illustrates an example of an OCR detection failure. Although the actual number of ammo displayed in the image is 8, the OCR module fails to detect any text (No text detected) and returns a null result. Another case of OCR failure is shown in Figure 11, where the model incorrectly recognizes the value as 741, despite the ground-truth number being 71.

5.5 Implications for Esports Analytics and AI Agents

Vision-based reconstruction of player status and on-screen context provides insights that are not available in traditional textual logs,



Figure 11: OCR detection failure case II.

which typically record only high-level events such as kills, item pickups, and score updates. By recovering additional state information, such as health, armour, ammunition levels, and nearby game entities, Q3Vision enables a richer interpretation of why players choose particular actions. For example, decisions related to weapon switching, retreating, or resource collection can be better understood when the underlying player status is known. This closes an important information gap and supports more comprehensive analysis of player behavior in competitive settings.

The ability to reconstruct dense, frame-by-frame game-state information from raw video also has implications for AI systems that rely on visual input. In particular, perception-based reinforcement learning agents depend on accurately capturing both spatial information and numerical state cues from pixels alone. Q3Vision provides a structured, reproducible method for extracting this information, making it possible to use legacy FPS footage as training or evaluation data for such agents. The fidelity of the reconstructed state directly affects an agent’s ability to learn meaningful policies, highlighting the value of robust vision-only pipelines in environments where engine-level telemetry is unavailable.

5.6 Future Work

Future work will focus on improving the robustness and generalization of the vision pipeline and expanding its analytical capabilities. A key direction is the fine-tuning of the OCR component to better recognize HUD elements in Quake III and OpenArena. This will involve constructing a dedicated training corpus by exporting cropped HUD digits and manually annotating their ground-truth values. Given that the HUD consists primarily of numeric characters, a domain-specific OCR model is expected to substantially reduce misclassification and improve temporal stability.

In addition, we intend to explore the use of multimodal models to support higher-level interpretation of gameplay. Although not included in the current system, future extensions may incorporate fine-tuned large language models deployed via Ollama [14] (for example Llama-3.2 model), supported by a Retrieval-Augmented Generation (RAG) database constructed from textual logs, extracted vision features, game documentation, and data dictionaries. By

grounding language-model reasoning in vision-derived state information, such an approach could help bridge remaining gaps in gameplay telemetry by providing contextual explanations that are not explicitly captured in either visual or log-based data alone. This component could be used to generate performance summaries, explain player decisions, or offer strategic insights based on reconstructed game states.

Beyond visual information, integrating additional data modalities presents a promising research direction. Incorporating audio cues, such as footsteps, weapon fire, or environmental sound, could provide richer context for understanding player decision-making. Similarly, collecting and analyzing input telemetry (keyboard and mouse actions) would enable investigation into reaction times and action selection, potentially supporting more detailed behavioral modeling and recommendations.

Lastly, we plan to extend the Q3Vision framework to other FPS and action titles with similar visual structures, evaluating its generalizability across different game engines, HUD styles, and gameplay dynamics.

6 Conclusion

This paper presented Q3Vision, a vision-based framework for reconstructing game-state information in Quake III Arena using object detection and OCR-based HUD parsing. By extracting both spatial and numerical cues directly from screen-captured footage, the system demonstrates that fine-grained player status and environmental entities can be recovered without access to internal game telemetry. Evaluation across diverse gameplay scenarios confirms the feasibility of this approach while also highlighting the challenges posed by rapid motion, visual noise, and stylized HUD elements.

Despite its effectiveness, Q3Vision currently faces limitations, particularly in the OCR component. The use of an untuned EasyOCR model leads to errors when interpreting low-resolution, low-contrast, or stylized HUD digits, reducing the stability of frame-by-frame numerical state estimates. These shortcomings underscore the need for domain-specific OCR fine-tuning and enhanced pre-processing tailored to game fonts and rendering artifacts.

Looking forward, several avenues for improvement and expansion remain. Future work will focus on fine-tuning OCR models using curated datasets of HUD elements, integrating temporal smoothing to improve the consistency of state trajectories, and incorporating additional modalities such as audio cues and player input telemetry to enrich behavioral analysis. Future work will also explore integrating vision-derived telemetry with existing log-based data streams to provide a more complete and temporally aligned representation of gameplay state. We also aim to develop interactive interfaces for real-time playback analysis and to explore the use of the reconstructed state as a training resource for perception-based reinforcement learning agents. Finally, extending Q3Vision to other FPS and action titles will further evaluate the generalizability of vision-only state reconstruction across diverse gaming environments.

Acknowledgments

The authors acknowledge support from the Telecom Research Unit (TRU), University of Technology Sydney (UTS), Sydney, Australia.

References

- [1] Jaied AI. 2024. EasyOCR. <https://github.com/JaiedAI/EasyOCR>. Accessed: 2025-12-08.
- [2] Gary Bradski. 2000. The OpenCV Library. *Dr. Dobbs's Journal of Software Tools* (2000).
- [3] Meng-Tzu Cheng, Louisa Rosenheck, Chen-Yen Lin, and Eric Klopfer. 2017. Analyzing gameplay data to inform feedback loops in The Radix Endeavor. *Computers & Education* 111 (2017), 60–73.
- [4] Yilun Cheng. 2023. Character Detection in First Person Shooter Game Scenes Using YOLO-v5 and YOLO-v7 Networks. In *2023 2nd International Conference on Data Analytics, Computing and Artificial Intelligence (ICDAICAI)*. IEEE, 825–831.
- [5] Derek Foo, Jianhao Tan, and Yap-Peng Tan. 2021. A Survey of Computer Vision Applications in Esports Analytics. *IEEE Transactions on Games* 13, 3 (2021), 253–264. doi:10.1109/TG.2020.3011398
- [6] Diego Rodriguez Garcia, Xinrui Yu, and Jafar Saniie. 2024. Basketball Video Analysis for Automated Game Data Acquisition Deep Learning. In *2024 IEEE International Conference on Electro Information Technology (eIT)*. IEEE, 173–178.
- [7] Reza Hadi Mogavi, Chao Deng, Jennifer Hoffman, Ehsan-Ul Haq, Sujit Gujar, Antonio Bucchiarone, and Pan Hui. 2023. Your favorite gameplay speaks volumes about you: Predicting user behavior and hexad type. In *International Conference on Human-Computer Interaction*. Springer, 210–228.
- [8] Thanapong Intharah and Gabriel J. Brostow. 2018. DeepLogger: Extracting User Input Logs From 2D Gameplay Videos. In *Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play* (Melbourne, VIC, Australia) (*CHI PLAY '18*). Association for Computing Machinery, New York, NY, USA, 221–230. doi:10.1145/3242671.3242674
- [9] Peiyuan Jiang, Daji Ergu, Fangyao Liu, Ying Cai, and Bo Ma. 2022. A Review of Yolo algorithm developments. *Procedia computer science* 199 (2022), 1066–1073.
- [10] Sheng Jiang and Erik Learned-Miller. 2019. Object Detection Under Extreme Motion: A Study of Fast-Moving First-Person Video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2223–2232. doi:10.1109/ICCV.2019.00231
- [11] Huang Jiayu and Liu Aofan. 2024. FPS Killer-A Moving Target Detection Algorithm in FPS Game Based on Yolov3 Frame and Gradient Descent Algorithm. In *2024 International Conference on Advancements in Smart, Secure and Intelligent Computing (ASSIC)*. IEEE, 1–6.
- [12] Shunji Mori. 1994. Historical review of theory and practice of handwritten character recognition. In *Fundamentals in Handwriting Recognition*. Springer, 43–69.
- [13] Raka Wirabuana Ninagan and Rinaldi Munir. 2023. Computer Vision Based Application Development for Screen Text Translation. *Technical Report, School of Electrical Engineering and Informatics, Bandung Institute of Technology* (2023).
- [14] Ollama Team. 2025. Ollama: Get up and running with large language models. <https://github.com/ollama/ollama>. Accessed: 2025-12-12.
- [15] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 779–788.
- [16] Tom Schultz and Marc Wieland. 2020. Understanding Player Behavior in First-Person Shooters Using Computer Vision Techniques. *Entertainment Computing* 33 (2020), 100–110. doi:10.1016/j.entcom.2019.100110
- [17] Baoguang Shi, Xiang Bai, and Cong Yao. 2016. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE transactions on pattern analysis and machine intelligence* 39, 11 (2016), 2298–2304.
- [18] MakeSense AI Team. 2025. Make Sense. <https://www.makesense.ai/>. Accessed: 2025-12-08.
- [19] Zi Ye, Dong Li, and Xiaoping Yang. 2020. GameVision: A Review of Computer Vision Techniques for Video Game Analysis. In *Proceedings of the 2020 International Conference on Computer Vision Systems*. Springer, 123–134. doi:10.1007/978-3-030-40605-9_12
- [20] Njayou Youssouf. 2022. Traffic sign classification using CNN and detection using faster-RCNN and YOLOV4. *Heliyon* 8, 12 (2022).