






Visual Analysis of Computer Game Output Video Stream for Gameplay Metrics

Kamil Kozłowski¹, Marcin Korytkowski² , and Dominik Szajerman¹  

¹ Institute of Information Technology, Lodz University of Technology,
ul. Wólczajska 215, 90-924 Łódź, Poland

dominik.szajerman@p.lodz.pl

² Częstochowa University of Technology, Al. Armii Krajowej 36,
42-200 Częstochowa, Poland

marcin.korytkowski@pcz.pl

<http://kisi.pcz.pl>, <http://it.p.lodz.pl>

Abstract. This work contains a solution for game metrics analysis based on a visual data stream dedicated for the player. The solution does not require interference in the programming code of the analyzed game and it is only based on image processing. It is possible to analyze several aspects of the game simultaneously, for example health/energy bars, current weapon used, number of objects worn (aid kits, ammunition). There have been presented methods using cascading classifiers and their training to detect the desired objects on the screen and to prepare data for other stages of processing, e.g. OCR. The effect of the methods is a gameplay chart that allows a thorough analysis of the player's actions in the game world and his or her advancement. The solution is fast enough that it can be used not only in previously recorded gameplay analysis, but also in real time during simultaneous gameplay.

Keywords: Gameplay metrics · HUD · Image processing

1 Introduction

Gameplay metrics are important parts of testing and analyzing computer games. They rely on the interpretation of raw data from the game, which may contain information about the player's current activities [10] – including, but not limited to, moving around the world, performing defined operations, or interacting with the graphical interface [4]. Acquiring data for the gameplay metrics process may require some access to the game's source code, whether by modding or compiling open source projects. Another way may be to use specialized equipment such as e.g. eyetracker to analyze the player's behavior, but for obvious reasons it is limited to laboratory conditions [9]. However, a lot of information could be obtained from the game result streams alone – image and sound. This is very

important, because it allows to analyze games at which access to the source code is impossible, such as those recently released – and do it very effectively [6].

The methods presented here can also be used in a broader scope, such as testing player behavior (including attempts to cheat in online games), or elements of automated tests, including tests of games produced by third-parties. The data obtained here can be a subset of the inputs of a bot’s program, which tests the game by building a map and navigating within it [3] or acting in other way [8].

The paper presents a solution consisting of three methods. The first combines known HUD¹ analysis with shape detection based on a cascade classifier to obtain a robust method independent of user screen configuration. The second expands the game analysis presented in the literature with a new algorithm. It involves recognizing shapes using cascade classifiers based on Haar-like [12] and Local Binary Patterns (LBP) [2] features to recognize moving technical objects – in this case, weapons in first-person shooter game (FPS). The third method allows to analyse the HUD detected by the cascade classifiers to recognize the text it contains, and interpret the strings obtained in this way. This allows to read secondary information about the game, such as the number of first aid kits or ammunition.

2 Related Work

Usually game metrics are understood as telemetry, i.e. remote collection of game-play data for many players for analysis and statistics [5]. Only game producers can do this because they have access to data collected on their game servers. The data can be various: equipped weapon, play time, difficulty level, current mission, avatar location, enemies’/NPCs’ parameters, save/load actions, player drop-off per mission, custom use of game mechanics. Applications other than play testing are, for example, bug tracking and QA. Game metrics results are often presented as graphs of measured value versus time or heat-maps showing the intensity of various measurements collected in 3D game space [5].

Audiovisual analysis of the game through postprocessing does not need to access the game code. Thus can be an effective variation of gameplay metrics for third-parties. It is based on audiovisual analysis of the recorded gameplay. It was presented in [6]. Key elements of communication with the player in the analyzed game, i.e. “Bioshock 2” [1] (e.g. health bar status, occurrence of the HUD on the screen, sounds of shots) can be precisely extracted from the game result streams. Then, these data can be presented as a graph illustrating several dozen minutes of gameplay. Such graph is a clear way to distinguish between the way a beginner player plays and an advanced one. The work [6] presents algorithms that allow to segment and index the gameplay – they are referred to as “Algorithm 1” and “Algorithm 2”. Our work references to them the same way.

¹ Heads-Up Display is the graphics part of the game user interface by which information about the state of the game is visually presented to the player.

“Algorithm 1” checks the occurrence of HUD on the screen (during non-interactive scenes the HUD is hidden, which excludes them from performance analysis) through the static logo detection algorithm on the screen, depending on the mask created earlier. “Algorithm 2” checks the player’s health and energy level as the appropriate colors constituting the health and energy bars in the areas of the screen determined by static masks.

As a field for further analysis [6] proposes to include a method of detecting objects on the screen, extracting text, and recognizing enemies by face detection.

3 Method

The solution is based on visual analysis of gameplay recordings in the game “Bioshock 2” [1], using the tools built into the OpenCV 3.3.1, and cascade classifiers that have been prepared by hand. In addition, the Tesseract text recognition system was used.

In order to get data for the analysis the gameplay recordings were made. To reduce the calculation time, the frames of the recordings for analysis should be averages of several seconds of gameplay [6]. Each image analyzed is an average of 30 frames. Averaging frames blur all the moving elements on the screen. The most static element of the world – a weapon held by the player – is also blurred, but not enough to prevent its analysis.

A certain problem in creating classifiers [7, 13] is the preparation of appropriate image samples: several hundred positive, containing the sought shape, and several thousand negative, not containing it. Although a thousand image samples can mean a huge amount of data and a long time to obtain them, it is also the number of frames from just 33 s of gameplay recording with a frequency of 30 frames per second. Samples were therefore obtained by recording a short gameplay and appropriate processing, described below.

Negative samples for learning the classifier were the entire screenshots of gameplay. To ensure the right variety of screenshots two rules were applied. The first was to quickly change the position and rotation of the camera throughout the entire recording. The second involves making recordings in several places – at least one dark and at least one bright. All classifiers were created on the basis of several uninterrupted gameplays recorded in two different chapters of “Bioshock 2” – no more than 30 s each – made separately for each weapon.

Positive samples were obtained in the same way, however, additional cropping was necessary so that they contained only the shape sought, with only small background areas at the edges. To increase the diversity of samples, a random jitter was introduced – for each frame a position offset and frame size are randomly chosen, equal to at most 5% of its width or height. To save calculation time when training the classifier, the n -th frame, instead of each were used.

The classifiers were created with the help of the *trainscascade* program included in the OpenCV package. Each classifier for this project was based on 200–900 positive samples, and 1000–2000 negative ones. The maximum false

alarm rate² as well as the specific number of samples were manually chosen to get the best results while keeping the training time below an hour. The exact values are presented in Table 1. For all classifiers the Gentle AdaBoost training algorithm was used, 19 was the maximum number of training epochs achieved. The classifiers were created in two variants, based on the same data sets – for Haar-like and LBP features.

Table 1. Training configurations of the different classifiers used in the method.

	Haar1/LBP1	Haar2/LBP2	Haar3	LBP3	Haar4/LBP4	Haar HUD
Negative samples	479	263	397	397	246	903
Positive samples	1501	990	1921	1921	1098	826
Max false alarm	40%	40%	40%	33%	40%	40%

3.1 Algorithm A. HUD Detection and Analysis

HUD detection is performed using a properly trained cascade classifier based on Haar-like features. The classifier reliably detects a screen section with the same shape and the same HUD fragment (the cropping differences between subsequent detections are small). It is therefore possible to train the classifier so that it detects one very characteristic fragment of the HUD and then enlarges the marked area (e.g. in percentage) so that it contains the entire HUD (Fig. 1).

The detected screen area – if the detection occurred – is intended for further processing. As in “Algorithm 2” [6], the intensity of colors was measured in two areas of the examined image (health bar and energy bar). It was based on masking the image and counting pixels with an appropriate shade relative to all non-zero pixels in the mask. In our approach, on the other hand, pixel filtering for shade and brightness in the HSV³ color space was used, and detection of rectangular contours describing the pixels detected in this way (bar filling). The width of the bright red rectangle in the HUD area determines the amount of health of the player, and the width of the bright blue rectangle means the amount of energy of the player.

The difference from “Algorithm 2” [6] is that the operation is local for the HUD, not for the entire screen, so the transformations made on the HUD are not relevant to it. As a result, it is possible to detect the presence of HUD on the screen and advanced analysis of its content regardless of its current location. In addition, it is not necessary to prepare an appropriate mask before conducting the tests: it is enough to prepare information about the desired colors. Optionally, additional cropping of the detected area can be used to reduce the chance of incorrect color detection in the background game world.

² False positive detection percentage allowed at each stage of training the cascade classifier.

³ hue, saturation, value.



Fig. 1. The cascade classifier detects and marks the screen area containing the HUD (processed frame from a screencast recorded from “Bioshock 2” [1]).

An intermediate “Algorithm A” has also been implemented, detecting the HUD area with a cascade classifier, and then analyzing it locally using masks, as described by “Algorithm 2”.

In addition, a separate implementation was made that works exactly according to Algorithms “1” and “2” of [6] to compare the effectiveness of HUD detection.

3.2 Algorithm B. Identification of Held Weapons

In “Bioshock 2”, as in the vast majority of FPS, the weapon held by the player’s character is always visible in the lower right corner of the screen as a model in the game world, making small movements both when moving the camera and the character, as well as when idle. In the averaged frame, the model is therefore blurred, but still retains a reproducible set of characteristics (Fig. 2).

The image analysis was carried out in the initial stages of the plot of the game “Bioshock 2”, where four weapons are available. Detections are carried out on each analyzed frame using four different classifiers, each trained to detect a different weapon: “Drill” (1), “Rivet gun” (2), “Hack tool” (3) and “Machine gun” (4). In order to optimize detection and reduce false positive errors, the analysis is performed only on a fragment of the image – specifically its right half. What counts is the fact of detecting the shape; whether the area found by the classifier contains all or only part of the weapon does not matter (Fig. 2). There is also no reason to create a general classifier (whether any weapons are kept) as part of the optimization, since the weapons are kept only when the HUD is displayed. And this is detected by the “Algorithm A”.

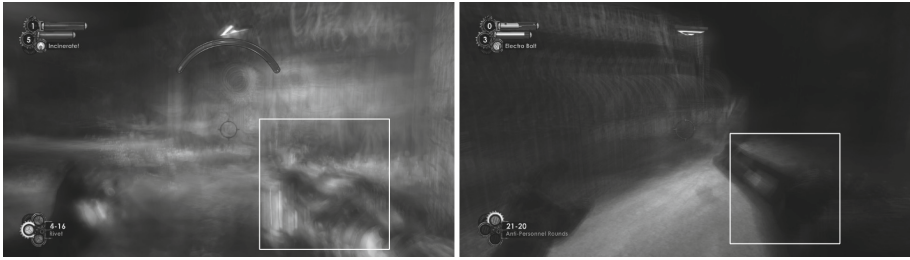


Fig. 2. The effects of the “Algorithm B”. Two different weapons are effectively detected and marked on the screen (processed frames from a screencast recorded from “Bioshock 2” [1]).

Based on which classifier in the given frame detected the weapon on the screen, it is determined which weapon is currently visible.

3.3 Algorithm C. Eliminating Weapon Identification Error

Due to problems such as invisibility of the weapon at some moments, caused by environmental factors (“Bioshock 2” takes place largely in very dark locations, so a large part of the features can be visible to a very limited extent), or the possibility of incorrect detection of weapons by “Algorithm B” (no weapons, or several weapons at the same time), it is necessary to use the method of mitigating the effects of errors. Our solution to the problem is to use fuzzy logic by creating a vector of floating point values that are stored throughout the analysis. Each value in the vector corresponds to the “truth” of the occurrence of a given weapon in a given frame based on the results of detection by cascade classifiers in the current frame and previous frame. Thus, with four detected weapons, the vector has four components. Each vector field has a specific maximum (here 3.0) and minimum (0.0) value. If any weapon is recognized in a given frame, the fields corresponding to each of the detected weapons are incremented by a certain value (here 1.0), and the non-detected weapon fields are decremented by the same amount. If no weapon is detected in the frame, the values of all fields are reduced by some other value (here 0.3). The highest value in the corresponding field of the vector gives current weapon. In this way, a single incorrect detection in one frame (as well as the detection of more than one weapon in a single frame), when the weapon is detected correctly in the previous and next frame, does not affect the result. If no weapon is detected, the stored values will be accepted as current for the next few frames (maximum of 10 frames for example values). However, the detection of a correct weapon change is delayed by only two frames.

This method can be effectively applied in the vast majority of modern FPS games – it is enough to prepare other classifiers for them, for weapons present in a given game.

3.4 Algorithm D. Reading Text in the Graphical Interface

The main HUD in the game “Bioshock 2”, shown in Fig. 3, in addition to two status bars also contains two numbers. The first of them means the number of first aid kits (supplementing the character’s health), and the second – the number of portions of “Eve”, supplementing energy. The lower HUD of the game, in turn, presents information about the quantity and type of ammunition for the currently selected weapon. These values have a huge impact on the player’s tactics and tell a lot about his style.



Fig. 3. Algorithm for detecting the value of the energy bar. From the left: The input area of the image detected by the classifier (additionally automatically cropped to improve processing); the image with detected pixels of proper shade and brightness (light blue); the output image with the outline of the energy bar marked, whose width is information about the energy possessed (fragments of processed frames from a screen-cast recorded from “Bioshock 2” [1]). (Color figure online)

After detecting these HUDs, the cropped image fragments are processed using the method described in “Algorithm A”. It is based on masking the interface so that only fragments that may contain text are visible, and on thresholding the image according to brightness to make the text clearer and remove unwanted shapes and gradients from the background. In the image prepared in this way, the text recognition process is carried out using the Tesseract-OCR system. The detection is based on default text recognition results for English, made available by the creators in a public repository [11].

If the pre-processing was successful, processing the detection results is very simple. Tesseract-OCR provides a text string of several lines whose analysis strictly depends on the analyzed HUD. For example, upper HUD analysis provides three lines of text: the first two are integers, the third is a simple string with the name of the plasmid⁴ used.

To support text recognition, the range of characters recognized by Tesseract can be limited, e.g. to numbers only. Particularly undesirable are simple punctuation marks, such as a period and comma, which may be misdetected on the basis of individual pixels.

4 Results

A series of tests of the methods was carried out on four different recordings of the game from the “Bioshock 2” game. It was done with the use of frame averaging

⁴ Characteristic of “Bioshock 2” gameplay object.

(30 in one) and without it. The recordings contained sequences of exploration, fighting, non-interactive story scenes (without HUD) and an open pause menu, recorded in various game locations, including well-lit and darker places, additionally differing in the colors of the environment. In total, over 11 min of various gameplay were tested, averaged into 699 samples. The recordings were made at a resolution of 1280×720 pixels, with a 30 frames per second (thus during the analysis one average frame contained data from exactly one second of the recording). Weapon classifiers operated with a scale factor of 1.02^5 , and searched for areas between 250×200 and 600×600 pixels.

For averaged frames, the average accuracy (ACC) of HUD detection was 92.13%, with a precision of 99.35% (PPV). Details of the effectiveness of the HUD detection classifier are presented in Table 2 (where it was designated “Haar HUD”). The standard deviation of the reading of the values presented in the HUD at a constant level of health and energy, using “Algorithm A” was 1.07 for both bars (with values given on a scale of 0–100), respectively. Standard deviations for the values read by the implemented “Algorithm 2” were 4.03 and 2.25, respectively, for health and energy. In the case of the indirect algorithm (“Algorithm 2” operating locally in the area cut out using the cascade classifier) 4.38 and 2.22, respectively. In addition, the implemented “Algorithm 1” confirmed its 100% HUD detection efficiency mentioned in [6].

Table 2. Table of statistics on the operation of various classifiers in the application. The rows “Total Haar” and “Total LBP” contain the sums of the TP, FP, TN and FN columns of all Haar and LBP classifiers, respectively, and the average of the other columns. Numbering 1–4 next to the classifier names defines the weapons as in the Subsect. 3.2. Haar HUD is a classifier used to detect HUD in the “Algorithm A”.

Classifier	TP	TN	FP	FN	Precision (PPV)	Sensitivity (TPR)	Specificity (SPC)	Accuracy (ACC)
Haar 1	124	445	107	24	53.68%	83.78%	80.62%	81.29%
Haar 2	92	495	8	105	92.00%	46.70 %	98.41%	83.86%
Haar 3	115	493	71	21	61.83%	84.56%	87.41%	86.86%
Haar 4	105	464	47	84	69.08%	55.56%	90.80%	81.29%
Total Haar	436	1897	233	234	69.15%	67.65%	89.31%	83.32%
LBP 1	132	532	20	16	86.84%	89.19%	96.38%	94.86%
LBP 2	161	466	25	37	86.56%	81.31%	95.02%	91.14%
LBP 3	94	562	2	42	97.92%	69.12%	99.65%	93.71%
LBP 4	121	431	79	69	60.50%	63.69%	84.51%	78.86 %
Total LBP	508	2002	126	164	82.95%	75.83%	93.89%	89.64%
Haar HUD	618	26	4	51	99.35%	92.37%	86.66%	92.13%

⁵ Defines the step size when the classifier scales the searched shape. The smaller the scale factor, the more calculations the classifier makes and the greater its accuracy.

When testing the “Algorithm B”, the average effectiveness of identification of the weapon held (compliance of the identified weapon with the actual value) was 58.08% for “Algorithm B” alone using LBP classifiers, 55.36% for Algorithm B” alone using Haar-like classifiers, and 66.95% for “Algorithm B” using LBP features and using “Algorithm C”. Details are presented in Table 2 and Table 3.

Table 3. Weapon identification efficiency for three different detection systems, for four different gameplay recordings.

Recording number	1	2	3	4
RAW LBP	75.00%	44.64%	74.83%	76.32%
RAW Haar-like	69.57%	46.43%	73.78%	70.30%
LBP with “Algorithm C”	77.17%	62.50%	82.17%	92.48%

For non-averaged frames, both the detection of weapons and HUDs increased the occurrence of false positive error, which reduced the effectiveness of both “Algorithm B” and “Algorithm A”, due to much greater information noise.

Using the presented algorithms, it was possible to create charts showing the states of health, energy and weapons used (Detected Weapon) in subsequent time samples for two (of four) chosen recordings – visible in Fig. 4. The algorithms “B” and “C” were tested independently of the “Algorithm A”.

On average, full analysis took 19.692 ms, of which 9.118 ms was HUD and 10.573 ms was weapon analysis. On this scale, it should also be mentioned that loading and averaging 30 frames of the recording took an average of 188.416 ms using an HDD. Tests were carried out on a computer with an Intel Core i7-4720HQ @ 2.60 GHz processor. The created system did not contain multi-threaded elements.

Training of classifiers with Haar-like features took 20–40 minutes on the above described equipment with 5 threads. Training of LBP classifiers on the same data in all cases lasted from 50 s to 18 min (where only LBP 3 was created for more than 5 min).

HUD analysis using the “Algorithm D” in “Bioshock 2” was performed on the same recordings as the detection of weapons. The upper HUD (first aid kit, eve) and lower HUD (ammunition) analysis were tested. The average effectiveness of text analysis in frames with correctly detected areas of text was 99.36% and 98.79%, respectively. The effectiveness of text analysis in all frames in which it should be carried out (the frames in which detection of screen areas was ineffective were included) was 93.25% and 95.90%, respectively. There was no false positive error in any of the 701 frames tested (there was never an accidental successful analysis of an area that did not contain the appropriate text). Details are presented in Table 4. The numbers in the upper part of the table indicate the number of frames meeting the condition shown in the left column.

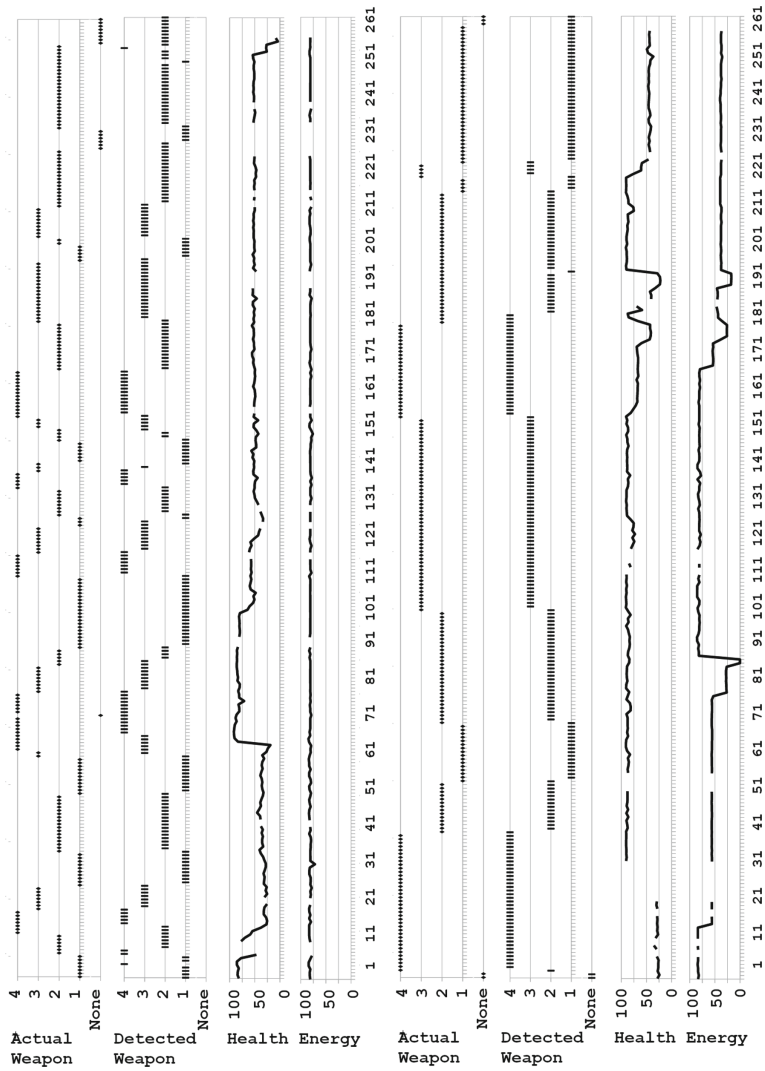


Fig. 4. Left: the first choosen gameplay recording chart. Right: the second one.

Table 4. Statistics on the analysis of two different HUD parts.

	Top HUD (aid kits, eve)	Bottom HUD (ammunition)
Correct detection of the area and its correct analysis result (A)	622	492
Correct detection of the area and incorrect analysis result (B)	4	6
Incorrect detection of visible area (FN)	41	15
Incorrect detection and full analysis of the invisible area (FP)	0	0
Correctly not detecting an invisible area (TN)	34	188
Total analysed frames	701	
Effectiveness of the analysis in the detected areas ($\frac{A}{A+B}$)	99.36%	98.80%
The effectiveness of the analysis in all frames that should be analyzed ($\frac{A}{A+B+FN}$)	93.25%	95.91%

5 Discussion

HUD detection using the Haar-like classifier brought noticeably less accuracy than the logo detection method of “Algorithm 1” (a decrease of about 8%). This, combined with the high accuracy achieved (a small number of FP errors) means that “Algorithm A” receives less data for analysis, but only a small part of them is incorrect (HUD analysis occurs rarely when it is not on the screen).

Random deviations of the detected position and HUD area by the cascade classifier should have a negative impact on the effectiveness of HUD analysis. However, there are negligible differences in the standard deviation values between the implementation of “Algorithm 2” in the full-screen version (described in [6]) and the indirect, classifier-based. This suggests that the deviations of the values are mainly caused by errors in detecting colored pixels in subsequent frames, rather than shifting areas of the masks. At the same time, the new method of analyzing the value of health and energy bars, finally described in “Algorithm A” brought much more stable results than both versions of “Algorithm 2” (with more than four times less standard deviation).

Weapon identification based on both Haar-like and LBP features has similar effectiveness – in some cases Haar-like gives better results, in others worse than LBP. The difference is always at most few percent (at most 6% points of difference). This is despite the fact that the LBP classifiers have on average clearly higher efficiency and precision than those based on Haar-like features.

For each set of samples, the use of the “Algorithm C” increased the effectiveness of weapon identification. The effectiveness of the “Algorithm C” decreases with frequent weapon changes, reaching the lowest value at first recording (Fig. 4 left). In the second recording (Fig. 4 right), where the effectiveness of identification with the “Algorithm C” is the highest of all cases, weapon changes occur least often (only 11 times in 4 min).

The number of samples that should be forwarded to the classifier learning process (and its maximum false alarm) to obtain satisfactory results is strictly dependent on the appearance of the specific weapon. For example, the most difficult classifiers to create were Haar 3 and LBP 3, detecting the “Hack Tool” (shown in Fig. 5). The reason was the distinctive appearance of the weapon, which contained two circles. This caused a false positive to occur frequently.



Fig. 5. Visual effect of the methods result: image from the average frame, the detected HUD area subjected to further analysis, and the identifier of the detected weapon in the lower right corner (processed frame from a screencast recorded from “Bioshock 2” [1]).

The analysis of the HUD content in terms of text content (“Algorithm D”) proved to be very effective for all recordings from the game, with a particularly low (below 1.3%) percentage of erroneous full analyzes, and a complete lack of False Positive errors. Their higher frequency could lead to undetectable analysis errors, distorting the generated picture of the course of the game.

Figure 4 shows the results of analyzing two different game recordings. They presented types of weapons appearing in subsequent samples, marked manually (Actual Weapon chart line), and types detected by “Algorithm B”, using LBP classifiers, supported by “Algorithm C” (Detected Weapon chart line). The

results of Actual Weapon and Detected Weapon are similar to each other, which means that the identification of weapons carried out by the application is quite effective. It is clearly seen that in Fig. 4 on the left side there is a greater discrepancy between the actual weapons and weapons detected by the algorithms than on the right. The reason is the player frequently changing weapons, which reduces the effectiveness of the “Algorithm C”.

Health and energy values presented in Fig. 4 were detected using the “Algorithm A”. It can be seen many breaks in them, caused by the algorithm not detecting the GUI due to an error or because the GUI was hidden in the game. For example, the pause in Fig. 4 on the left side, near the sample (seconds) 246, is the result of the player entering the shopping menu. However, the gap in Fig. 4 on the right side, at sample 26, is caused by a false negative error of the classifier, extending over a dozen samples. Despite the missing data at some moments, the charts can be used to determine differences in the player’s actions. According to Fig. 4 on the left, the player lost health during the fight around 11 s (samples) and healed 60 s later. At the same time he at no time used special skills that would cause loss of energy – its state was constant all the time. In Fig. 4 on the right, the player repeatedly used his skill, and in the 157th second he started a 40-s battle that caused health levels to wave over that period. Thanks to the weapon chart, it can be said that most of the aforementioned fight took place using a “Machine gun” (weapon 4), which at the end (about 181 s) was changed to a “Rivet gun” (weapon 2).

6 Conclusions

The use of cascade classifiers in the presented methods has significantly reduced its creation time. This is due to the possibility of using a small number of samples and the use of a small number of learning stages. Despite this, the classifiers achieved a precision of 75%.

Using the presented methods, it was possible to distinguish a set of four classes of objects (weapons) with an efficiency of over 66%, which turned out to be sufficient for analyzing the game. The presented algorithm for minimalization identification errors turned out to be positive in all test cases.

LBP classifiers present in this application – i.e. detecting different types of weapons – show noticeably higher efficiency than Haar-like classifiers. However, this does not translate into clear gains in the effectiveness of weapon identification without the use of an algorithm that eliminates identification errors. Much shorter learning time makes the process of improving and testing LBP classifiers in an iterative way much more efficient than with Haar-like classifiers.

The time results of the algorithms clearly show that the method can be effectively used in real time – not only as postprocessing, but also as an analysis of ongoing gameplay.

The presented method of HUD detection and analysis is effective to a degree similar to the literature method [6], however, it allows you to analyze the interface regardless of its location on the screen, so it is suitable for analyzing moving interface elements.

Further work may include the preparation of more precise classifiers, which should significantly increase the effectiveness of weapon identification. The system has the potential to significantly parallelize calculations, because of all the methods presented only the “Algorithm C” has to be executed sequentially. The presented graphical interface analysis method can be used in other games to analyze static and moving interface elements. Weapon identification algorithms can be used in most FPS games.

The algorithms presented here allowed to make an effective analysis of the player’s state of change over time in three aspects: energy level, health level and the type of weapon held. The obtained statistics allow to determine the style of playing and to detect and analyze the most important moments from the recorded game (e.g. the course of the fight) based on the generated graph itself.

Of course, the methods presented have their limitations. While the analysis of GUI elements can also be useful for other game genres, the specific weapon detection algorithm will only be useful for FPS games. The method based on postprocessing and classification of graphic elements would always need to be adjusted if the type of game analyzed changes. But other game metrics methods are not without this drawback, where the collected data itself is closely related to a specific genre or even a specific game.

References

1. 2K Games: Bioshock 2 (2010). <https://store.steampowered.com/app/8850/>
2. Ahonen, T., Hadid, A., Pietikäinen, M.: Face recognition with local binary patterns. In: Pajdla, T., Matas, J. (eds.) ECCV 2004. LNCS, vol. 3021, pp. 469–481. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24670-1_36
3. Daszuta, M., Wróbel, F., Rynkiewicz, F., Szajerman, D., Napieralski, P.: Affective pathfinding in video games. *J. Appl. Comput. Sci.* **26**(2), 23–29 (2018)
4. El-Nasr, M.S., Drachen, A., Canossa, A. (eds.): Game Analytics. Springer, Heidelberg (2013). https://www.ebook.de/de/product/20778057/game_analytics.html
5. El-Nasr, M.S., Drachen, A., Canossa, A. (eds.): Game Analytics. Springer, London (2013). <https://doi.org/10.1007/978-1-4471-4769-5>
6. Marczak, R., Schott, G., Hanna, P.: Postprocessing gameplay metrics for gameplay performance segmentation based on audiovisual analysis. *IEEE Trans. Comput. Intell. AI Games* **7**(3), 279–291 (2015). <https://doi.org/10.1109/tciaig.2014.2382718>
7. Opalka, S., Stasiak, B., Szajerman, D., Wojciechowski, A.: Multi-channel convolutional neural networks architecture feeding for effective EEG mental tasks classification. *Sensors* **18**(10), 3451 (2018)
8. Rogalski, J., Szajerman, D.: A memory model for emotional decision-making agent in a game. *J. Appl. Comput. Sci.* **26**(2), 161–186 (2018)
9. Szajerman, D., Napieralski, P., Lecoite, J.P.: Joint analysis of simultaneous EEG and eye tracking data for video images. *COMPEL Int. J. Comput. Math. Electr. Electron. Eng.* **37**(5), 1870–1884 (2018). <https://doi.org/10.1108/compel-07-2018-0281>

10. Szajerman, D., Warycha, M., Antonik, A., Wojciechowski, A.: Popular brain computer interfaces for game mechanics control. In: Zgrzywa, A., Choroś, K., Siemiński, A. (eds.) *Multimedia and Network Information Systems. AISC*, vol. 506, pp. 123–134. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-43982-2_11
11. Tesseract Data: `tesseract-ocr/tessdata`. <https://github.com/tesseract-ocr/tessdata/>
12. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2001*. IEEE Computer Society (2001). <https://doi.org/10.1109/cvpr.2001.990517>
13. Walczak, J., Poreda, T., Wojciechowski, A.: Effective planar cluster detection in point clouds using histogram-driven KD-like partition and shifted mahalanobis distance based regression. *Remote Sens.* **11**(21), 2465 (2019)