Hearthstone Card Recognizer

Board recognizer for the game Hearthstone using image processing methods.

Authors

- Danilo de Moraes Costa - NUSP 8921972
- Guilherme Zanardo Borduchi - NUSP 8937458
- Lucas Silveira de Moura - NUSP 8937267

Project Objective

Hearthstone is a popular collectible card game, launched in 2014, with more than 70 million unique players. With more than 1000 playing cards, our project aims to process an image from the playing board, and try to detect which cards you are holding, which monsters and weapons are on board and how many cards your oponent has, from which point it will be possible to present various informations to the user, such as who is currently winning the game, what combination of cards can be played and how can the monsters interact.

Input Images

The input images are composed of screenshots of games being played, and also a large database containing every playable card in the game, to be able to detect them on the screenshots. It is important that the input screenshot of the game has a resolution of 1920x1080 pixels.

The screenshots were obtained by playing the game and capturing them in different situations (for example, when the board is full of monsters and also when it is empty). We didn't need a large number of screenshots, since the biggest challenge of this project relies on identifying which of the more than 1000 playable cards are in play.

The database of cards was created by using the Hearthstone API service, which provides a JSON with information about all the cards on the game. We then filtered the JSON so that we could fetch only the currently playable cards on the standard format, which excludes cards older than 2 years. Finally, we downloaded each of the cards on the filtered JSON to create our database.

Figure 1 is an example of the type of screenshots that were used. Note how the monsters can have different shapes and particle effects, and also how the cards in your hand get rotated and more grouped together as you draw more.

On Figure 2 we have an example of the images that compose our card database. They are matched against the screenshots previously shown. Also note how the scales of the images are very different, and how once a monster is played, only its picture is shown on the board (creating another challenge).

All the images used in our project are stored on the repository, either on the directories "Minion", "Spell" or "Weapon", which constitutes our database, or on the directory "Screenshot", which consists of the images to be analyzed.

Figure 1: The board of the game Hearthstone.

Figure 2: One of the playable cards in Hearthstone.

Steps and Methods

The first step in our method was the creation of the databases of the images. For that, we used the "fetcher.py" file, which returned a big JSON containing every card on the game, including old cards that have rotated out of the standard format, and also other miscellaneous stuff, like the heroes (playable characters) of the game.

Possessing this JSON, the next step was to then filter it. For that, the file "filter_downloader.py" was used. This script saved a new JSON, this time only allowing the playable cards of the standard format, which reduced our space from 3085 cards to 1148 cards. It is important to note that some of the links obtained from the Hearthstone API were broken. We manually fixed these links with the links from the Hearthpwn website, which returned the correct images.

This same file also downloaded every image on the filtered JSON, thus completing the creation part of our database of cards.

The next step was an optimization one, using the "descriptor_writer.py" file. Since we used the SIFT method to match the images on the database with the images on the scene, that involved calculating the keypoints and descriptors for every image on the database. We decided to save every descriptor (we didn't need the keypoints for the matching process) on it's own file, and include the reference to these file on the filtered JSON, generating yet another JSON file. This helped with the running times immensely, since the calculations were only done once and then loaded afterwards, instead of calculating them on the fly multiple times for each run.

The SIFT method was chosen because, since our project deals with matching images on a scale, color and rotation variant environment, it seemed like the best alternative amongst others like SURF or ORB to extract the relevant points on the image.

The final steps of the process consist on counting the number of cards in the player's hand and the number of minions at play on board, to match them individually using a FLANN-based matcher. These steps are all performed on the "matcher.py" file, which is the only file needed to run the project if you have already setup the previous steps.

In order to count the number of cards in the player's hand and the number of minions at play on board, we decided to use the board in our advantage. Each match of the game can take place in a set of several different boards, but the board is fixed for that specific match. In order to use the board, we took several screenshots, one of each different board in the standard format. As we are going to use the board to check how many minions are in play and how many cards in the hand, it was essential that the boards had those areas empty in the screenshots. The empty board images are available in the "Board" folder.

The first step the program takes as soon as it reads an input image is to find what board that match takes place on. A small window is cropped off from each of the boards as well as from the input image itself. Then the program compares them to know what is the most likely board in that match.

The process of determining how many cards are in the player's hand is given by a simple algorithm. First, a cropped and blurred space of the input image is subtracted from the board image of that game (cropped and blurred as well). That should give us really dark or really bright pixels where the intensities of both images were similar. Those pixels are probably not cards, because cards usually have different colors than the board behind them. That being said, we applied a threshold for really dark or bright pixels, making them black, while the remaining pixels were set to white. Figure 3 shows an example of this result. We then tested this image against every hand size mask (those premade masks can be found in the "Mask" folder). The mask that best described the result of the subtracted images tells us how many cards the player has in their hand.

To detect the minions currently on the board, some processing had to be done. The first step was

Figure 3: The player's hand after basic processing.

to subtract the empty board from the input image. Then, a very harsh threshold is applied, so that the minions are rendered white, while the rest of the image remains black. Then, some crops of the image are taken, on all the spots the minions can stay when they are played. These positions depend uniquely on if there are an even or odd number of minions at play.

Now an erosion is applied, with an elliptical kernel of size 100, to help distinguish the odd crops, when there are actually an even number of minions at play, and vice-versa for the even crops, when there are actually an odd number of minions at play. The central pixel of each crop is checked; if it is white, that means it is a match; it it is black, that means there is no minion on that crop.

After we got the crops for every card in the player's hand and every minion on board, they are matched against our database of images using the SIFT keypoints and detectors and the FLANN-based matcher. The best match for each of these crops is considered to be on the scene.

Results Obtained

We were really satisfied with the accuracy of the project. We obtained no mismatches across all the tested screenshots. The program is capable of determining what is the board of the game, how many cards the player currently has in their hand, how many minions are on the player's board as well as the opponent's. Above all, the program is able to tell exactly what is every card: all minions on the board and cards in player's hand.

That happens because we try and compare every card of the hand and board with all the cards of the database and select the one with the most correct matches. Unfortunately this leads to a very slow run time, therefore the program still cannot be used in real time while playing the game. It takes roughly 4 minutes to parse all the information described above.

Figures 4 and 5 are examples of the outputs of our project. It firstly informs the game board being played on, then the cards on the player's board, and lastly the cards on the opponent's board. All of the matches were accurate, with no mismatches.

Demo Code

To execute our project, please run the program with "python matcher.py scene", where "scene" is the path to the screenshot to be analyzed. There are several screenshots to be used in the directory "Screenshots/".

Additionally, out project includes a small file for Winduws users ("demo.bat") that executes the process for 2 inputs ("Screenshot/screenshot8.png" and "Screenshot/screenshot1.png"). Please note that this demo can take more than 10 minutes to run. The results are output on standard output.

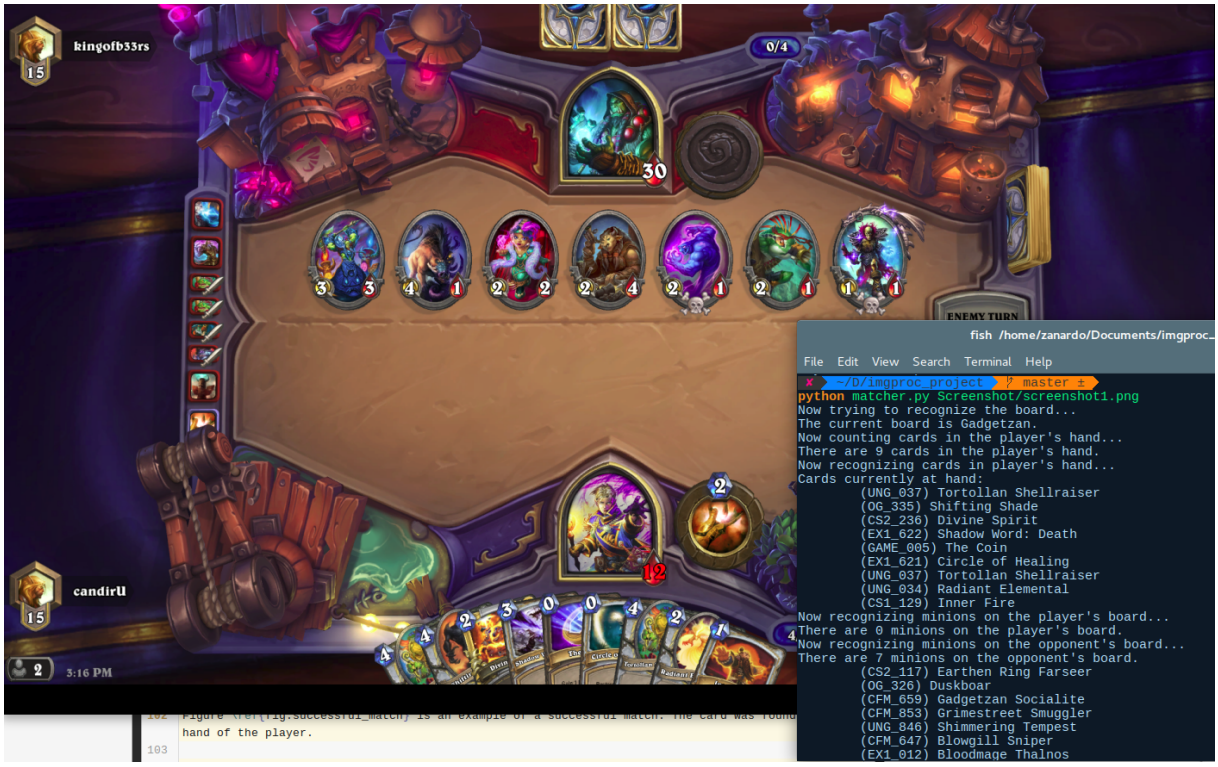

Figure 4: Output of our project.

Figure 5: Another output, this time with a lot of cards on the board and also on the player's hand.