Pokémon Battle Screen Analyzer

EDBV WS 2019/2020: AG_A5

Rupert Ettrich (01129393)
Marko Zivojinovic (01633063)
Friedrich Decker (00625081)
Lena Dolinek (11808609)
Paul Nefischer (01613480)

6. Januar 2020

1 Gewählte Problemstellung

1.1 Ziel

Das Ziel des Projekts war es, ein Programm zu entwickeln, das alle verfügbaren Informationen aus einem Kampfbildschirm mit geöffneter Attackenauswahl aus dem Spiel Pokémon FireRed/LeafGreen für den GameBoy Advance auszulesen. Dieses Programm soll als Entscheidungshilfe dienen und kann dabei von unerfahrenen Spielern als Hilfe verwendet werden, oder aber auch als Entscheidungsgrundlage für eine AI dienen, die selbstständig Entscheidungen im Kampf treffen soll. Im Vordergrund stehen dabei die Methoden der Bildverarbeitung, die verwendet werden, um die Eingabebilder zu analysieren. Pokémon wurde vor allem deswegen als Spiel gewählt, weil es eines der größten Franchises in der Geschichte der Videospiele ist, und sich neue sowie alte Spiele aus dieser Reihe nach wie vor großer Beliebtheit erfreuen.

Abschließend sollte eine Abschätzung getroffen werden, ob sich das Projekt tatsächlich zum beschriebenen Einsatz eignet, oder ob die Performance oder die Genauigkeit dazu nicht ausreichen.

1.2 Eingabe

Die Eingabebilder sind Screenshots im .png - Format vom Spiel *Pokémon FireRed/LeafGreen Version*, welche von Emulatoren für das Handheld-System *GameBoy Advance* stammen und einen Kampfbildschirm zeigen, auf dem die auswählbaren Attacken des eigenen Pokémon zu sehen sind.

1.3 Ausgabe

In der Ausgabe werden folgende Informationen erfasst und beschrieben:

1. Die am Kampf teilnehmenden Pokémon inklusive Typinformationen und Basiswerten
2. Das Level beider Pokémon, um die Stats der Pokémon abschätzen zu können
3. Die HP-Leiste beider Pokémon in Prozent
4. Die Attacken des eigenen Pokémon, um Treffwahrscheinlichkeit, Effektivität und Schaden abschätzen zu können
5. Die Statuseigenschaften des Pokémon (paralysiert, schlafend, vergiftet, etc.)

1.4 Voraussetzungen und Bedingungen

Da, wie bei jedem Projekt, einige Dinge erst im Laufe des Entwicklungsprozesses klar werden, wollen wir hier noch einige zusätzliche Bedingungen erwähnen, die wir im ursprünglichen Konzept nicht berücksichtigt hatten.

1.4.1 Ursprüngliche Voraussetzungen

Die Eingabebilder müssen einen Battle-Screen zeigen, wobei die Attackenauswahl des eigenen Pokémon sichtbar sein muss, damit diese verarbeitet werden können. Die Bilder müssen digitale Screenshots in Originalfarbe sein, also mit irgendeiner Art von Screen Capturing - Software aufgezeichnet worden sein und keine Fotos. Die originalen Seitenverhältnisse sollen beibehalten werden, die Bilder sollen nicht rotiert werden. Die Skalierung der Bilder ist beliebig, wobei als Minimum die Originalauflösung des Systems 240 x 160 Pixel (3:2) festgelegt wird.
Da Pokémons mit anderen Farben als der Originalfarbe (sog. “Shinies“) nur selten vorkommen, werden diese aus dem Datensatz ausgeschlossen, um die Template-Database möglichst kompakt zu halten.

1.4.2 Erweiterte Voraussetzungen

Zusätzlich wäre es sinnvoll zu erwähnen, dass momentan nur Screenshots in englischer Sprache korrekt erkannt werden. Zwar wäre es auch möglich, andere Sprachen zu implementieren, allerdings müssen dann auch die unterliegenden Datenbanken und gegebenfalls die Alphabete für das Template-Matching erweitert werden, um korrekt zu funktionieren. Da dies nicht unbedingt etwas mit Bildverarbeitung zu tun hat, haben wir in diesem Projekt ausschließlich mit englischen Screenshots gearbeitet.
Aus Zeitgründen haben wir es nicht geschafft, Doppelkämpfe in das Programm zu integrieren. Bei Doppelkämpfen treten 4 Pokémon gleichzeitig in Teams von 2 gegeneinander an. Da diese nur einen sehr kleinen Prozentsatz der Kämpfe darstellen, haben wir uns entschieden, sie wegzulassen.

Ebenso gab es Probleme dabei, Pokémon zu erkennen, die ihr Sprite durch Attacken ändern können und somit ihre Farbe ändern. In der ursprünglichen Problemstellung haben wir aus dem selben Grund Shinies ausgeschlossen.

1.5 Methodik

Während der Entwicklung haben wir bemerkt, dass nicht unbedingt alle Methoden aus unserem ursprünglichen Konzept auch sinnvoll eingesetzt werden können, oder dass es andere Lösungen gibt. Daher haben wir zum Beispiel keine Bildpyramiden für das Template-Matching verwendet, da es wesentlich effizienter war, das Bild auf die Originalauflösung herunterzuskalieren und anschließend dort die Sprites in bekannter Auflösung zu matchen.
In der finalen Pipeline finden sich daher folgende Methoden:

1. Resizing, Segmentierung, Umwandeln in Graustufen- oder Binärbilder
2. Template-Matching
3. OCR
4. Thresholding

Im globalen Preprocessing wird das bild auf die Originalgröße skaliert. Das spezifische Preprocessing segmentiert das Bild in die für die jeweilige Aufgabe spezifischen Bildbereiche und wandelt den Teilbereich gegebenfalls in ein Graustufen- oder Binärbild um. Dann werden mittels Template-Matching die Pokémon Sprites gematched und die Statuseigenschaften überprüft. Bei den Pokémon Sprites werden die Hintergründe der Teilbilder, die die Sprites enthalten, zusätzlich mit Thresholding nach der Otsu-Methode entfernt. Als Optimierung werden hier die Seitenabstände zum Rand mit den Sprites in der Datenbank verglichen, um, falls der Unterschied zu groß ist, keine Zeit für das Template-Matching zu verschwenden, da sonst jedes Mal 840 Sprites gematched werden müssen (420 für nach vorne und 420 für nach hinten schauende Pokémon). Mittels OCR, das ebenfalls auf Template-Matching basiert, werden die Attacken und Levels der Pokémon gelesen. Für die HP-Leisten wird der Prozentsatz der farbigen Pixel innerhalb der Leiste im Vergleich zur Gesamtlänge zurückgegeben.

1.6 Evaluierungsfragen

Fragen die sich bei unserem Program im Bereich Evaluierung stellen sind unter anderem

- Wie lange dauert die Auswertung eines Input Bildes im Durschnitt und wie genau ist diese in Einbezug der unterschiedlichen Auflösungen?
- Wie sieht unser Datensatz aus, wie wurde dieser angelegt und ist er Ausreichend ?
- Wodurch kommt es, wenn überhaupt, zu Erkennungsproblemen unterschiedlicher Pokemon Sprites und könnten diese behoben werden?

- Gibt es problematische Bereiche bezüglich Level-, Lebens- und Attackenauswertung und gibt es für diese Lösungsansätze?

1.7 Zeitplan

Meilenstein	abgeschlossen am		Arbeitsaufwand in h	
	geplant	tatsächlich	geplant	tatsächlich
Datensatz, Datenbanken, Grundgerüst erstellen	11.11.2019	1.12.2019	30h	40h
Preprocessing / Segmentierung	18.11.2019	18.11.2019	30h	5h
Prototyp mit GUI	25.11.2019	18.11.2019	40h	15h
Erstversion Pokémon erkennen	02.12.2019	25.11.2019	25h	25h
Erstversion Attacken erkennen	02.12.2019	10.11.2019	25h	10h
Erstversion HP-Leiste lesen	02.12.2019	10.11.2019	20h	4h
Erstversion Statusveränderungen erkennen	02.12.2019	10.11.2019	20h	10h
Erstversion Level erkennen	02.12.2019	2.12.2019	20h	8h
Testumgebung / Evaluierung	09.12.2019	6.1.2020	30h	35h
Finale Version	23.12.2019	5.1.2020	60h	120h
Erstellung der Testdatensätze	-	5.1.2020	-	15h
Abgabedokument	-	6.1.2020	-	20h

Zusätzlich zu den genannten Meilensteinen kommt noch der Zeitaufwand für Meetings, Besprechungen, gruppeninterne Kommunikation und Präsentationsvorbereitungen. Unter *Erstversion* ist der Aufwand für die Prototypen-Implementation mit Matlab-Funktionen zusammengefasst, unter *Finale Version* ist die Ergänzung durch die eigenen Implementationen von Toolbox-Funktionen, das Zusammenfügen der Teilaufgaben und diverse Bugfixes zusammengefasst, was deutlich mehr Zeit beansprucht hat als ursprünglich gedacht.

2 Arbeitsteilung

Name	Tätigkeiten
Marko Zivojinovic	Matlab-Funktion: preprocessing_healthbar, imageprocessing_healthbar, read_moves script und CreateDatabase + zugehörige moves Datei (Skript zum Erstellen unserer Datenbank und befüllt diese mit Moves) Bericht: Abschnitt 4 (Healthbar), Abschnitt 5 (Evaluierung), Abschnitt 6 (Schlusswort)
Paul Nefischer	Matlab-Funktion: preprocessing_level, imageprocessing_level, mequal Bericht: Abschnitt 5 (Evaluierung), Abschnitt 6 (Schlusswort)
Rupert Ettrich	Matlab-Funktionen: determineWhitespace.m, matchTemplate.m, otsuThreshold.m, imageprocessing_pokemon_sprites.m, preprocessing_all.m, preprocessing_pokemon_sprites.m, Matlab-Klassen: Database.m, Pokemon.m, PokemonSprite.m, TestData.m, Matlab-Skripts: loadSprites.m, createDatabase.m, populateDatabase.m, testScript.m Skript zur Erstellung der Pokemon in der Datenbank Bericht: Abschnitt 1, Abschnitt 3 (Template Matching und Thresholding), Abschnitt 4 (Pokemon Sprites) Abschnitt 5
Lena Dolinek	Matlab-Funktionen: preprocessing_status.m, imageprocessing_status.m Bericht: Abschnitt 4 (Status)
Friedrich Decker	Matlab-Funktionen: preprocessing_attacks, imageprocessing_attacks, normxcorr2PA, spatialCrossCorr, imresizePA, imcropPA, rgb2grayPA, sizePAgray, sizePA, diverse Hilfsfunktionen, ... Matlab-Klassen: Attack, AttckDTO, AttckPreprocessingDTO GUI.mlapp (Grafische Benutzerschnittstelle mit App Designer) Bericht: Abschnitt 3 (OCR), Abschnitt 4 (Attacken, Allgemeines)

3 Methodik

3.1 OCR

Zur Erkennung der Attacken wird für jedes Zeichen des Alphabets die normalisierte räumliche Kreuzkorrelation (normxcorr2) des jeweiligen Buchstabens (in Form eines Graustufen-Templates, Abbildung 2) an jeder infrage kommenden Position des Battle Screens (ebenfalls Graustufenbild, Abbildung 1) berechnet und somit gewissermaßen eine Heatmap (Abbildung 3) erstellt: Die Werte liegen zwischen -1 (perfekte negative Korrelation) und 1 (perfekte positive Korrelation). 0 bedeutet, dass keine räumliche Korrelation vorliegt. Wir sind an hohen Werten nahe 1 (etwa ab 0.95) interessiert. An diesen Stellen entspricht der Bildausschnitt in der Größe des Templates vom Originalbild

(="Battle Screen") dem Template mit dem jeweiligen Buchstaben selbst (zumindest zu weiten Teilen).

Abbildung 1: Originalbild

Abbildung 2: Template für Buchstaben E

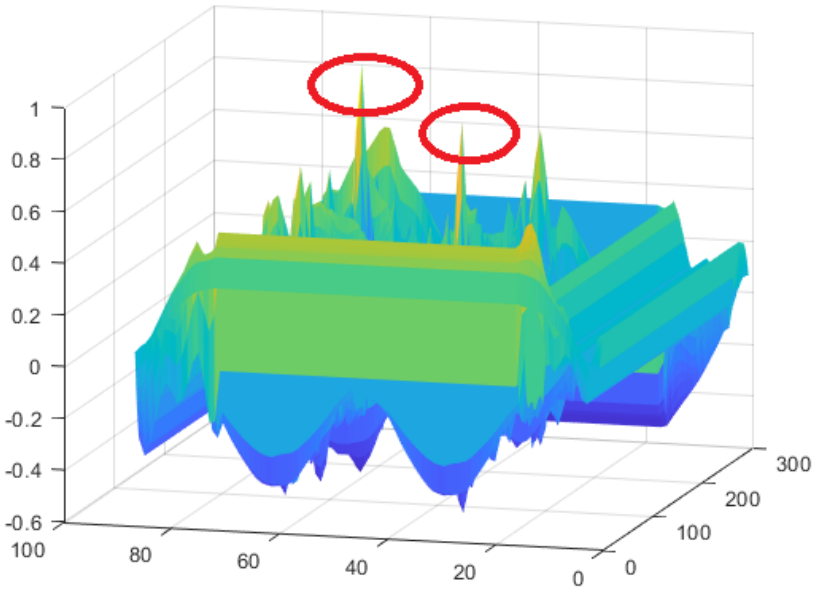


Abbildung 3: Die durch normxcorr2 Generierte Heatmap mit zwei Gipfelregionen über 0.95 – diese entsprechen den Positionen des Templates im Originalbild (durch die gewählte Perspektive wirken sie niedriger).

Template Matching mithilfe räumlicher Kreuzkorrelation ist eng mit dem Konzept der euklidischen Distanz verwandt [2]. Dieses Vorgehen ist in vertretbarer Zeit möglich, da ein fixer Font in einer fixen Größe verwendet wird. Kommen viele verschiedene Schriftarten und/oder Größen infrage, würde dieses Verfahren zu lange dauern oder eine zu große Unsicherheit bzw. Fehlerquote mit sich bringen, und man müsste wohl eine andere

Methode wählen (z.B. neuronale Netze, ...).
Zur Berechnung der räumlichen Kreuzkorrelation wird folgende Formel verwendet:

$$\gamma(u,v) = \frac{\sum_{x,y}[f(x,y) - \overline{f}_{u,v}][t(x-u,y-v) - \overline{t}]}{\{\sum_{x,y}[f(x,y) - \overline{f}_{u,v}]^2 \sum x,y[t(x-u,y-v) - \overline{t}]^2\}^{0.5}}$$

wobei gilt, dass:

- f ist das Originalbild
- $\overline{t}$ ist der Durchschnitt unter dem Template
- $\overline{f}_{u,v}$ ist der Durchschnitt von $f(x,y)$ in der entsprechenden Region unter dem Template
- Quelle: https://de.mathworks.com/help/images/ref/normxcorr2.html (6.1.2020)

Die Berechnung der Durchschnittswerte muss lediglich einmal erfolgen und kann ins Preprocessing verlagert werden. Die errechneten Werte werden gespeichert und im Processingteil verwendet, wodurch ein effizienter Einsatz der Methode möglich ist.

3.2 Template Matching

Beim Template Matching wird versucht, das Vorkommen und die Position eines Templates innerhalb eines Eingabebildes zu ermitteln. Dabei gibt es unterschiedliche Ansätze, wie zum Beispiel das Feature-basierte Matching, bei dem versucht wird, die Bilder mittels stark ausgeprägter Features oder Control Points wie Kurven oder Kanten zu matchen. Eine andere Herangehensweise ist das naive Template-Matching, das auch in diesem Projekt verwendet wird. Da das Input-Bild klar definiert ist und frei von Verzerrungen, Transformationen (bis auf uniforme Skalierung) oder sonstigen Veränderungen ist und die Größe des Templates innerhalb des Eingabebilds im Vorhinein bekannt ist, kann hier theoretisch ein 1:1 Match erzielt werden zwischen Template und Originalbild. Dabei werden die Möglichen Positionen des Templates im Originalbild ausprobiert und die Unterschiede der Farbwerte verglichen. Eine häufig verwendete Metrik zur Bestimmung der Position ist die *Sum of squared Differences/Distances*. Dabei wird der Unterschied zwischen den Farbwerten des Templates und des Eingabebildes quadriert und aufsummiert. Außerdem wird ein gewisser Schwellenwert festgelegt, wenn dieser unterschritten wird, ist ein Match an der Position wahrscheinlich. Da in unserem Projekt das Template bei den Pokemon Sprites immer genau einmal gematched werden kann, wird kein Schwellenwert verwendet, sondern das beste Ergebnis wird verwendet. Die mathematische Definition für die *Sum Of Squared Distances* ist $\sum_{x,y}(f(x,y) - t(x-u,y-v))^2$, wobei f(x,y) für den Farbwert im Originalbild und t(x-u, y-u) für den Wert im Template steht, das um u und v verschoben wurde [1].

3.3 Thresholding

Als Thresholding werden Methoden bezeichnet, bei denen versucht wird, einen einzelnen Grenzwert zu finden, der einem Farbwert oder einer Intensität im Bild entspricht. Mit diesem Wert soll ein Binärbild aus dem Eingabebild erzeugt werden, dass sich möglichst gut in Vordergrund und Hintergrund auftrennen lässt. Beim Thresholding wird außerdem unterschieden zwischen adaptiven und globalen Methoden. Adaptive Methoden eignen sich, wenn zum Beispiel ein Farb- oder Helligkeitsverlauf im Bild stattfindet, da der Threshold nicht für das gesamte Bild, sondern immer nur für eine festgelegte Umgebung bestimmt wird.
Für dieses Projekt wurde allerdings ein globales Verfahren gewählt, nämlich die Otsu-Methode. Es wird beim Template-Matching für die Pokemon - Sprites verwendet, um Hintergründe zu entfernen. Der Grund dafür, warum diese Methode gewählt wurde, ist, dass kein Farbverlauf stattfindet und die meisten Hintergründe im Spiel relativ schwache Farbintensität im Vergleich zu den Pokemon Sprites im Vordergrund haben.
Beim Otsu-Thresholding wird als erstes das Histogramm des Eingabebildes erstellt. Aus diesem lassen sich die Häufigkeiten für die verschiedenen Intensitäten, die im Bild vorkommen, ablesen. Im Anschluss werden alle möglichen Werte für den Threshold durchprobiert - bei einem 8-bit-Graustufenbild sind das 256 - und der beste Wert wird als Threshold bestimmt. Dabei gibt es zwei Ansätze, die beide zum gleichen Ergebnis führen. Entweder kann versucht werden, die intra-class-variance, also die Varianz zwischen den Farbwerten einer Klasse (Vorder- oder Hintergrund) zu minimieren, oder es kann versucht werden, die inter-class-variance, also die Varianz zwischen Vorder- und Hintergrund zu maximieren [1]. Beides führt zum selben Ergebnis, wegen der einfacheren Berechnung wurde jedoch bei der Implementation die 2. Variante gewählt, also die Maximierungsvariante.

4 Implementierung

4.1 Status

Die Status der beiden Pokemon sind in fixen Bildbereichen zu sehen, zum Bestimmen wird im Preprocessing das Eingabebild auf 160x240 skaliert und die ensprechenden Bildbereiche werden ausgeschnitten. Im Processing wird zunächst zwischen zwei Fällen unterschieden, ob es einen Status gibt oder nicht, dazu wird ein *Template-Matching* durchgeführt. Für das Matching wird das Eingabebild in Graustufen umgewandelt. Es gibt fünf verschiedenen Status, die sich in Graustufen sehr ähnlich sehen. Beim Versuch fünf verschiedene Templates zu verwenden kam es oft zu falschen Matchings, deshalb wird nur mit einem einzigen Template, welches ein Durchschnitt der fünf Status ist, gematcht. Falls das entsprechende Pokemon keine Statusveränderung besitzt, ist der Bildbereich einfärbig oder ein Pokeball ist zu sehen. Andernfalls gibt es einen Status, hierbei haben die fünf verschiedenen Möglichkeiten unterschiedliche Farben. Um Festzustellen um

[1] http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MORSE/threshold.pdf

welche Farbe und somit um welchen Status es sich handelt, werden die RGB-Farbwerte eines Pixels des Eingabebilds mittels *euklidischer Distanz*[2] mit den tatsächlich im Spiel verwendeten Farben der Status verglichen.

4.2 Level

Um die Level der InputImages auszulesen, muss das InputImage erst in ein 160x240 Schwarz-Weiß Bild umgewandelt werden. Aus diesem kann man dann 2 fixierte Bildbereiche um die Zahlen herauscroppen. Diese werden dann abermals aufgeteilt in insgesamt 4 Matrizen, die jeweils eine Ziffer enthalten. Um mit der 1Pixel Varianz des Bildes klarzukommen kann man jedoch die Ziffern nicht direkt herausschneiden, sondern muss die Position der Ziffer im ausgeschnittenen Bildbereich errechnen. Sobald man diese hat, kann man jede Ziffer in eine 7x4 Matrix abspeichern. Diese Matrizen werden dann mit den vorbereiteten Ziffern-Templates abgeglichen und geben dann die übereinstimmende Zahl zurück. Falls das Level eines Pokemons einstellig sein sollte wird durch das Errechnen der entsprechenden linken Ziffer-Position erkannt, dass es sich nicht um eine Ziffer handeln kann desweiteren als Null gehandhabt. Sollte ein dreistelliges Level, also "100", vorkommen so wird dies durch das erkennen von 2 Nullern in einem Level klar.

4.3 Pokemon Sprites

Um die Sprites zu matchen, wird im Preprocessing das Bild auf 160x240 skaliert und für die weitere Verwendung in ein Graustufenbild umgewandelt. Außerdem werden die ungefähren Bildbereiche aus dem Bild geschnitten, wo das eigene und das gegnerische Pokemon zu erwarten sind (preprocessing_all.m und preprocessing_pokemon_sprites.m). Dabei ist die horizontale Position eindeutig festgelegt, die vertikale Position ist jedoch variabel. Außerdem können vom eigenen Pokemon einige Pixelzeilen verdeckt sein durch die Attackenauswahlbox. Beim Startup der Applikation wird für das Template Matching einmal das gesamte Sprite-Template-Dataset eingelesen (loadSprites.m), was aus 420 Sprites für eigene und 420 für gegnerische Pokemon besteht. Die Sprites der Pokemon sind Farbbilder der Größe 64x64, dabei werden die Abstände vom Rand der Sprites bestimmt und von jedem Sprite Versionen in unterschiedlichen Größen erstellt (48x48 und 32x32) und im Arbeitsspeicher abgelegt, um nicht jedes Mal von der Festplatte lesen zu müssen. Dies braucht beim Startup natürlich einige Sekunden, dafür wird im Nachhinein viel Zeit gespart, wenn mehrere Bilder bearbeitet werden. Beim eigentlichen Template-Matching (imageprocessing_pokemon_sprites.m) wird dann der Otsu-Threshold der vom Preprocessing übergebenen Teilbilder bestimmt, um den Hintergrund zu entfernen und den seitlichen Abstand zur Bounding Box des im Bild vorhandenen Templates ungefähr bestimmen zu können - die horizontale Position ist ja eindeutig fixiert. Dann werden alle verfügbaren Sprites durchgetestet und der beste Match wird als Ergebnis hergenommen. Dabei werden Templates geskipped, deren seitliche Abstände stark vom gefundenen Sprite abweichen, um Zeit zu sparen. Dies funktioniert aber nicht mit allen Hintergründen

[2] https://uwspace.uwaterloo.ca/bitstream/handle/10012/937/swesolko1999.pdf

gleich gut, da manchmal Teile des Hintergrunds auch nach dem Thresholding noch da sind. Als Template Matching Methode wird *Sum of Absolute Distances* verwendet, da diese Methode gegenüber Sum of Squared Distances ein wenig effizienter ist, weil das quadrieren weggelassen wird, aber beim Testen mit den Samples die gleiche Genauigkeit aufweist.

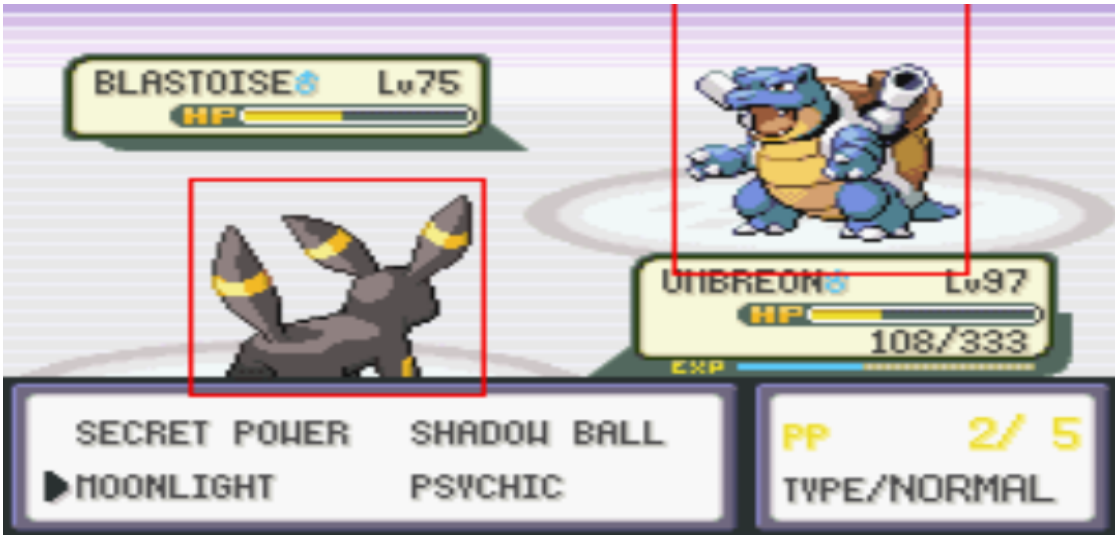


Abbildung 4: Die Positionen der Pokemon sind horizontal fixiert, vertikal aber leicht variabel.

4.4 Healthbar

Da es sich bei den Lebensleisten um fixe Bereiche auf unserem Bildschirm handelt, können hier die gewünschten Koordinaten manuell überprüft und mittels imcrop heraus geschnitten werden. Da der Lebensbalken mehr als 3 Pixel breit ist, kann hier, wenn genau die mitte erfasst wird, die 1 Pixel-Verschiebung, welche beim User vorkommt, ignoriert werden. Dieser Bereich wird mit einer Höhe von 1 Pixel gecropped und anschließend noch von RGB in schwarz und weiß umgewandelt. Hier muss auch noch darauf geachtet werden, dass zwei Fälle eintretten für jeden Screenshot, nämlich für den Gegner und für den User. Anschließend wird in der Processing Methode lediglich jedes Pixel in der 1 Pixel hohen Matrix überprüft ob dieses noch weiß ist und anschließend durch die Gesamtlänge dividiert. Das ist unser angenehrter HP Wert.

4.5 Attacken

Zunächst wird der Bildausschnitt, der die Attacken enthält, auf die minimale Größe (240x160 Pixel) skaliert und dann extrahiert und in ein Grauwertbild umgewandelt. Im nächsten Schritt wird der Bildausschnitt weiter in 4 Teile unterteilt, die jeweils eine Attacke enthalten. Diese Teile können dann parallel abgearbeitet werden, falls der zum Einsatz kommende Prozessor über zumindest 4 Threads verfügt. Nun werden für jeden

Buchstaben aus dem Alphabet (dieses liegt in Grauwertbildern als Templates der Größen 5x8 bzw. 4x8 vor) mithilfe einer selbst implementierten Funktion, welche die normalisierte räumliche Kreuzkorrelation an jeder Stelle berechnet (Einzelheiten hierzu: siehe 3.1 Methodik, OCR), die Orte bestimmt, an denen er auftritt. An jenen Stellen, wo die Werte der Korrelation 0.95 übersteigen, wird der zum Template gehörende Buchstabe angenommen. Die Buchstaben werden in der Reihenfolge, in der sie auftreten, gespeichert. Die Reihenfolge ergibt sich aus dem Vergleich der jeweiligen Koordinaten der gefunden „Gipfelregionen". Damit werden die Strings für alle 4 Attacken sukzessive aufgebaut. In einem letzten Schritt werden sie von Großbuchstaben auf das von der/dem UserIn gewohnte Leseformat (Beginn Großbuchstabe, dann Kleinbuchstaben) umgewandelt.

4.6 Allgemeines zur Programmausführung

Für den gesamten Programmablauf steht eine GUI zur Verfügung. Diese kann vom MATLAB App Designer aus gestartet werden (Doppelklick auf Datei: src/GUI.mlapp). Innerhalb dieser GUI können Bilder ausgewählt und analysiert sowie automatisch getestet werden. Dem/der UserIn steht es dabei frei, einen zufälligen Battle-Screen generieren zu lassen (also aus vorhandenen Ressourcen zufällig einen auswählen lassen) oder den File-Picker zu benutzen, um eine Datei anzugeben, die sich auf dem Rechner der/des UserIn befindet. Das automatische Testen kann ebenfalls über die GUI durchgeführt werden. Dazu wechselt man in den Tab „Testing". Es kann ein Ordner angegeben werden, der die Screenshots enthält, wobei default der Ressourcen-Folder aus den Programm Assets gewählt ist. Mit Klick auf „Run Full Test" werden alle Screenshots aus dem angegebenen Ordner analysiert und das jeweilige Ergebnis mit der Testdaten-Datenbank verglichen. Nach Ende des Tests werden eine Statistik sowie Details zum Test ausgegeben. Weitere Informationen und benötigte Add-Ons sind auch dem Readme-PDF zu entnehmen!
Das Setzen der Parameter erfolgt folgendermaßen: Der/die UserIn hat den Speicherort des zu analysierenden Screenshots auszuwählen bzw. einen zufälligen auswählen zu lassen und dann auf den Button „Analyze Battlescreen" zu klicken, um die Ausführung zu starten. Beim „vollen Testen" eines gesamten Verzeichnisses mit Daten (Screenshots) kann der Pfad des Verzeichnisses bei Bedarf geändert werden (z.B. um einen schnellen Test mit einem Teil der Daten, die dann in ein anderes Verzeichnis zu kopieren sind, durchzuführen).

5 Evaluierung

Als Input für unser Programm verwenden wir einen Datensatz an selbst erstellten Screenshots, die den von uns angegebenen Spezifikationen entsprechen. Dieser Datensatz enspricht einer größe von 313 Images, unterschiedlicher Auflösungen. Die Verarbeitung eines dieser Images beträgt im Durchschnitt etwa 1.5s. Getestet wurde dies auf einem Intel i2500 (3.3Ghz) mit 8GB DDR3 RAM, dabei wird für die einzelnen Teilaufgaben folgende Genauigkeit erzielt:

Eigene Pokemon Sprites	301/313	96%
Gegnerische Pokemon Sprites	310/313	99%
Eigene HP	305/313	97%
Gegnerische HP	288/313	92%
Eigener Status	309/313	98%
Gegnerischer Status	306/313	98%
Eigenes Level	307/313	98%
Gegnerisches Level	307/313	98%
Attacken (4 pro Datum)	1223/1252	98%

Da überall eine Genauigkeit von mindestens 92% erzielt werden konnte, sind wir mit dem Ergebnis durchaus zufrieden. Allerdings ist die Abdeckung des Datensatzes natürlich nicht vollständig, daher ist das keine absolute Genauigkeit. Außerdem gibt es bestimmte Voraussetzungen, unter denen die Datensätze falsch analysiert werden, dazu gibt es Genaueres im Abschnitt 5.1. Zudem ist noch zu beachten, dass die gegnerischen HP nur wage abgeschätzt werden können. Ein tatsächlicher Vergleich für die HP Funktion ist bei den eigenen HP zu sehen, welche eine Genauigkeit von 97% erzielen.

Wir verwenden eine Datenbank mit drei Datensätzen.
Im ersten Datensatz werden 326 Attackenname als auch deren Typ, Schaden, Effekt, Treffgenauigkeit, Statusänderungen und deren Chance einzutreffen, verzeichnet.
Hier war das erhalten der notwendigen Informationen eine größere Hürde, da zum Zeitpunkt des Erstellens kein offizieller und vollständiger Datensatz vorhanden war.Es wurde als Ansatz folgende Liste verwendet [3] , kopiert und bearbeitet. Hierfür wurde ein Skript geschrieben, welches die vorhande Liste einliest und für jede Attacke eine Zeile in der Datenbank mit den vorhandenen Informationen füllt.

Im zweiten Datensatz befinden sich alle 386 Pokemon und deren relevante Information wie Nummer, Name, Typ und durchschnittliche Werte.[4] . Diese Information wurde anschließend im JSON Format von dieser Website bezogen, bearbeitet und in der Datenbank gespeichert.

Des weiteren befindet sich ein dritter Datensatz in unserer Datenbank für unsere Testfälle. Diese wurden zu jeden erstellten Screenshot manuell erstellt. In diesem befinden sich alle relevante Informationen für das Abgleichen unserer Auswertungen.

Zusätzlich wird ein seperater Image Datensatz für das bearbeiten von Leveln im Ordner dataset gespeichert. Bei diesen Bildern handelt es sich um 10, 4x7 große Templates, welche dem Abgleichen der Pokemon Level dienen.

5.1 Probleme und Verbesserungsvorschläge

Folgende Problematiken und entsprechende mögliche Lösungen sind uns im Verlauf dieses Projektes untergekommen:

[3] https://gamefaqs.gamespot.com/gba/918915-pokemon-firered-version/faqs/33491
[4] https://pokemondb.net/pokedex/all

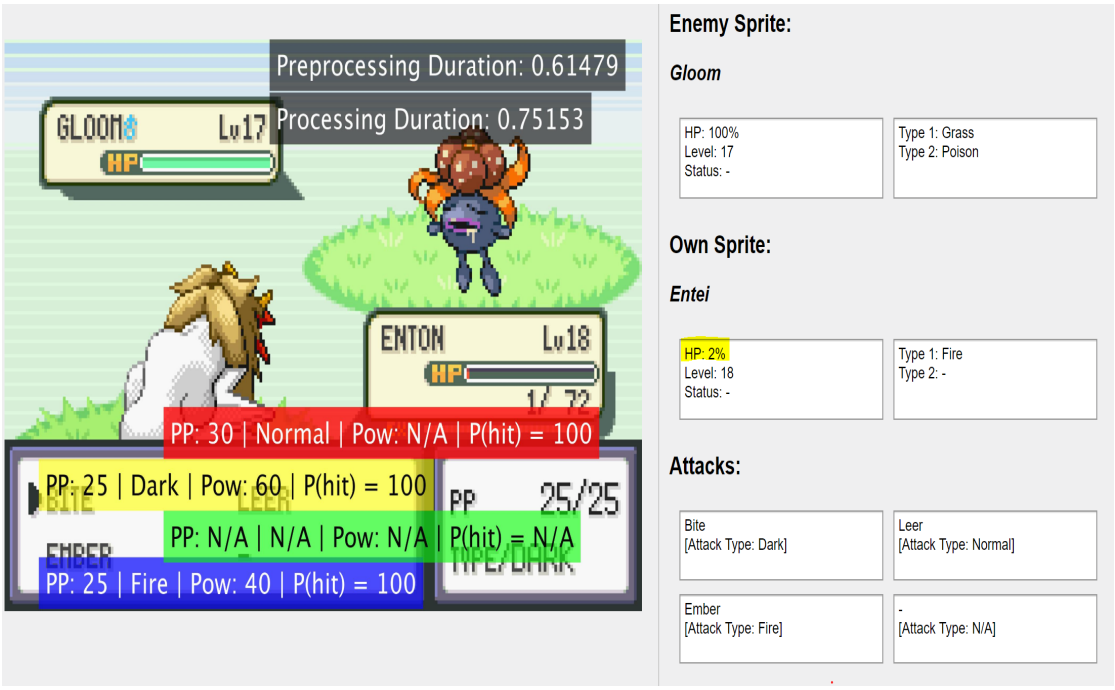
- Attacken, die die Sprites der Pokémon verändern, führen dazu, dass das Matching nicht funktioniert. Ein Beispiel hierfür ist in Abbildung 5 vorzufinden. Die triviale Lösung dafür wäre es, den Template-Datensatz um die Sprites mit veränderter Farbe zu erweitern. Dies würde natürlich die Laufzeit erhöhen.
  Folgendes Bild zeigt z.B das Pokemon MEW nachdem es die Attacke Transform eingesetzt hat und damit dasselbe Sprite wie der Gegner annimmt und somit nicht mehr erkannt werden kann.

Abbildung 5: Durch die Attacke 'Transform' geht das ursprüngliches Sprite verloren.

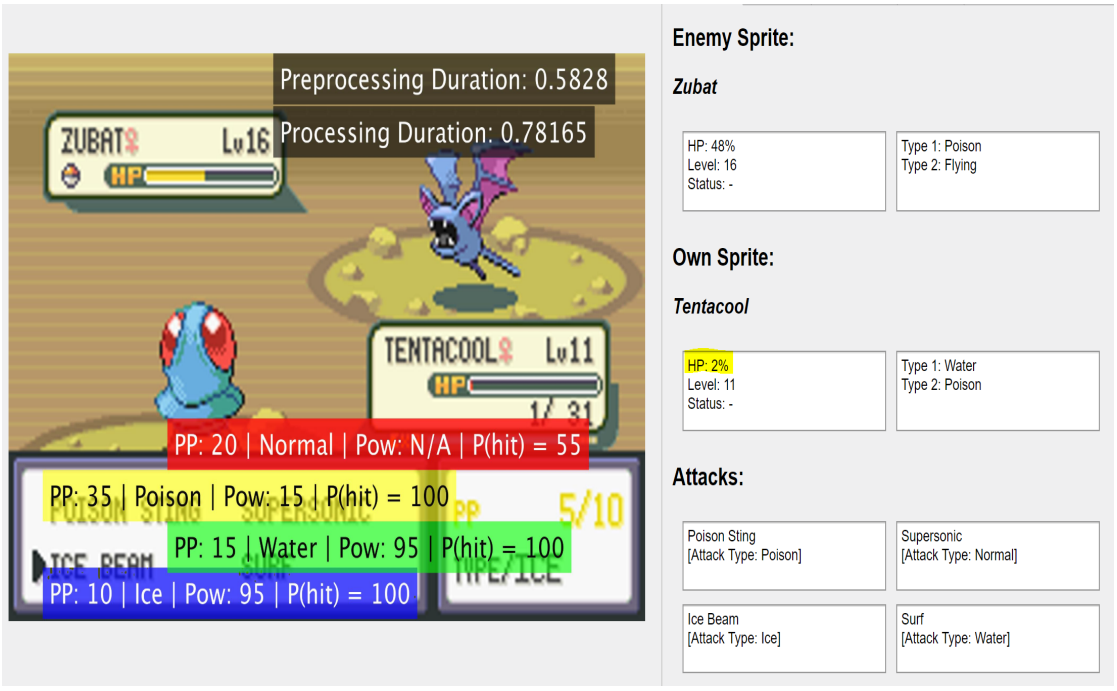
- Der Testdatensatz deckt nur einen Teil der möglichen Daten ab: 59/386 eigene Pokémon und 141/386 gegnerische Pokémon, ähnlich verhält es sich bei den Attacken. Ein vollständiges Testen ist theoretisch möglich, hätte aber den zeitlichen Rahmen des Projekts innerhalb dieser Lehrveranstaltung gesprengt, da der Datensatz manuell angelegt werden musste.

- Doppelkämpfe wurden nicht berücksichtigt. Um ein Programm zu entwickeln, das für alle möglichen Kämpfe funktioniert, muss dies ebenfalls integriert werden.

- Bezüglich der HP Auslese kann zum derzeitigen Zeitpunkt kein perfektes Ergebnis erzielt werden, da die Darstellung der HP aus zu wenig Pixel besteht und dementsprechend ein ungenauer Wert zurückgegeben wird. Z.B wird bei folgenden Bildern jeweils 2 % als HP Wert zurückgegeben.



(a) 1/72HP = 2%



(b) 1/31HP = 2%

Abbildung 6: Ungenaue Prozentangabe in anbetracht der eigentlichen HP

- Attacken, die die Sprites der Pokémon verschwinden lassen, führen zu einem falschen Match. In diesem Fall ist es aber auch nicht möglich, ein Sprite zu matchen. Das Programm sollte aber in zukünftigen Versionen zumindest erkennen, dass sich kein Pokémon am Feld befindet.

6 Schlusswort

Das Ziel des Projektes war es, ein Program zu schreiben, das als Entscheidungsgrundlage für Pokemonkämpfe dienen kann. Sowohl das Erkennen der Sprites sowie das Herauslesen von Level, Leben und Attacken funktioniert relativ akkurat, wobei beim Auslesen der HP-Leiste leichte Schwankungen bestehen, die auf die niedrige Pixelanzahl des Ursprünglichen Formates zurück zu führen sind.
Momentan gibt es außerdem noch Probleme mit Grenzfällen, die größtenteils durch das Erweitern von Datensätzen gelöst werden können. Unter anderem müssten Sonderformen wie sogenannte Shinies hinzugefügt werden, und Attacken, die die Farbe des Pokemon Sprites ändern (Transform) oder diesen gänzlich verschwinden lassen (Fly und Dig) müssten noch in Betracht gezogen werden.
Allerdings würde dies zu weitaus höheren Laufzeiten führen, wodurch der eigentliche Verwendungszwecks des Programmes teilweise eingeschränkt wird. Dementsprechend liefert unser Programm einen guten Ansatz, welcher mit Verbesserungen im Bereich der Laufzeit auf ein von uns gewünschtes Ziel kommen könnte.

Literatur

[1] Nazanin Sadat Hashemi, Roya Babaie Aghdam, Atieh Sadat Bayat Ghiasi, and Parastoo Fatemi. Template matching advances and applications in image analysis, 2016.

[2] J. P. Lewis. Fast normalized cross-correlation, 1995.