Artificial Intelligence and Computer Vision Project Report

Chimelie Nzelibe (257283), Krzysztof Hoszowski (259771)

11-2-2023

Game Control Using Hand Gesture Detection with Python/OpenCV

Contents

1 Abstract

The aim of this project was to enable webcam input as controls for the Snake video game (or, by extension, any other application). Two hand gestures were chosen to represent turning clockwise or anticlockwise in the game. The whole project is implemented in software, in the Python 3 programming language. This includes the Snake game, image processing and machine learning. The main libraries used for operations were *numpy*, *pandas* and *cv2* (OpenCV), as well as *pygame* for the Snake game itself.

First in this report, the Computer Vision and Artificial Intelligence parts of the project (from henceforth abbreviated as CV and AI, respectively) are discussed in two following chapters. Then, the combined implementation is presented and results are commented on. The Snake header file and other auxilliary programs can be found in the Appendix.

2 Computer Vision

While the AI part does not strictly require any preprocessing, it is a practical necessity. It greatly facilitates machine learning, and is required for smooth and responsive game experience. In our case, we take a picture with personal computer's webcam and wish to extract from it the human hand. We considered various approaches to this problem, including using a high-pass filter, or filtering based on higher red content in the RGB representation of the human hand. We chose the latter solution, due to its simplicity and reliability.

Below is the image preprocessing program, and its results on a number of randomly selected pictures. It times the transformation of each image and utilizes a number of optimizations, including downsizing the image, using a numpy array as mask, and converting unsigned integers to signed. Surprisingly, the latter significantly shortens runtime, despite the additional three operations. It also makes the program more flexible, as it is easy to overflow an 8-bit integer. Additionally, it was noted that the algorithm tended to produce noise near the borders of the mask, therefore they are omitted by the width of a filter variable.

```python
"""
   Performs binary thresholding on an image based on R
   content in RGB image, aiming to extract the human hand
   from it for further processing.

   Co-authored by Chimelie Nzelibe and Krzysztof Hoszowski.
   February 2023.
"""

from time import time
import cv2 as cv
from numpy import zeros


def extract_hand(img):
    """Returns binary mask of human body from picture."""

    # Dimension of image (square)
    shape = 256

    # Downscale the image for faster processing
    img = cv.resize(img, (shape, shape))

    # Create a binary mask
    msk = zeros((shape, shape, 1), dtype="u1")

    # Filter variable
    fil = 30

    # Loop over the pixels in the image
    for y in range(fil, img.shape[0] - fil):
        for x in range(fil, img.shape[1] - fil):
            # OpenCV uses BGR representation instead of RGB
            b, g, r = img[y, x]

            # Optimization: convert unsigned int into signed
   int.
            r = int(r)
            g = int(g)
            b = int(b)

            # Alternative algorithm
            # if r > 80 and g > 30 and b > 20 and r > g and r
    > b and abs(r-g) > 15:

            # Compare the RGB values
            if r > g + fil and r > b + fil:
                # Keep the pixel
                msk[y, x] = 255
    return msk


# Test the extracting on 8 examples
if __name__ == "__main__":
    for i in range(1, 9):

        # Timing the program
        start_time = time()

        # Load the image
        image = cv.imread("".join([str(i), ".jpg"]), cv.
   IMREAD_COLOR)

        mask = extract_hand(image)

        print(f"-- It took {time() - start_time} seconds --")

```

```python
        # Show the binary image
        cv.imshow("binary mask", mask)
        cv.imwrite("".join([str(i), ".png"]), mask)
        cv.waitKey(0)
        cv.destroyAllWindows()
```

Listing 2.1: Hand-extraction algorithm.

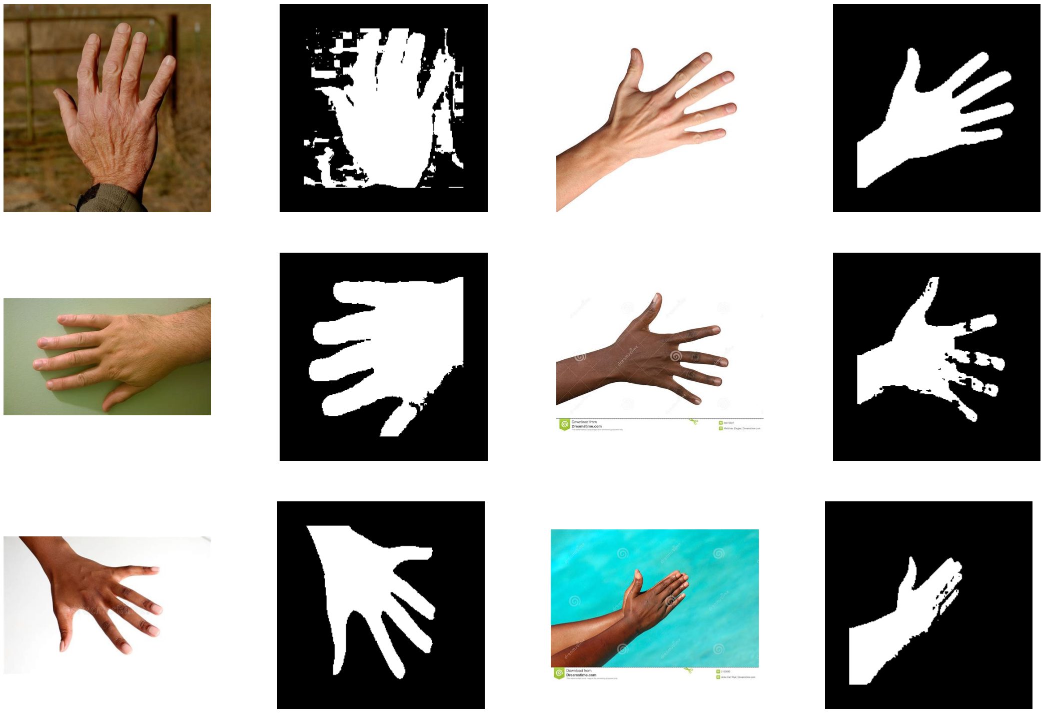

Figure 2.1: The results of image processing.

As visible, the algorithm works very well for a variety of pictures in good lighting. Nevertheless, it has its flaws, as is visible by the image with the field in the background. Perhaps refining it or combining it with another form of filtering, such as high-pass filter, would make it more robust. On average, the images take just under 100 milliseconds to process (approx. 93 ms). That theoretically allows 10 Frames Per Second, which is enough for the Snake game.

3 Artificial Intelligence

The purpose of the AI is to recognize whether its input contains one of two special gestures: turn right or turn left. These gestures are showing either the side of the hand, or the back of the hand, respectively. Any other gesture, such as clenched fist, means the program should not accept any player input. In addition to this, a third prediction from the AI model was trained, whereby the hands are not shown at all. This was done to simplify the final program, avoiding the use of probabilities and using the prediction directly.

Below is the program code. The neural network is trained using Supervised Learning approach from *sklearn* library, and the time it takes is recorded. The images are read as *pandas* dataframes using a simple algorithm based on *Pd2Img* library[1]. The same algorithm is used in the final program to process webcam images on the fly. The model is prepared using Random Forest Classifier, which tends to yield the best results in comparison with other methods. At the end, the model is generated a classification report and confusion matrix.

```python
"""
   Training and testing the hand gesture detection algorithm.

   Made by Krzysztof Hoszowski, February 2023.
"""

# Performance timer
from time import perf_counter

# Data processing
from pandas import concat
from image_to_dataframe import im2df

# Training the AI model
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Assessing the quality of the AI model
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

# Saving the classifier
```

```python
from joblib import dump


tic = perf_counter()  # First reading

# Reading the dataframes
dfn, dfc, dfa = im2df()

# Adding labels to dataframes
d = {"label": 0}
dfn = dfn.assign(**d)

d = {"label": 1}
dfc = dfc.assign(**d)

d = {"label": 2}
dfa = dfa.assign(**d)

# Concatenating the dataframes
df = concat([dfn, dfc, dfa], axis=0)


x = df.iloc[:, 0:3]  # Parameter columns
y = df.iloc[:, -1]  # Label column

# Splitting 75% of data into training set, 25% into test set
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.25, random_state=1
)

tac = perf_counter()  # Second reading


# Training the model using Random Forest Classifier
lr_grid = {"max_depth": [4, 8, 16], "criterion": ["entropy", "gini"]}

clf = RandomForestClassifier(n_estimators=12, max_features="sqrt", random_state=1)

# Fitting the model. Grid Search is used to optimize hyper-parameters
gs = GridSearchCV(estimator=clf, param_grid=lr_grid, cv=5)

gs.fit(x_train, y_train)
```

```python


# Making the prediction
y_pred = gs.predict(x_test)
gs.best_params_

# Obtaining classification report and confusion matrix
print("\nClassification Report: \n", classification_report(y_test, y_pred))
print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))

toc = perf_counter()  # Third reading


print(f"\nTime taken preparing data: {tac - tic:0.2f} seconds")
print(f"Time taken training and testing: {toc - tac:0.2f} seconds\n")


#########################
# SAVE using joblib #
#########################

dump(gs, "model.pkl")
```

Listing 3.1: Machine learning algorithm.

The following is output of the program. For each gesture, 20 pictures were used. Unfortunately, producing a robust estimator that works in variety of environments and lighting conditions is extremely difficult. Because our team did not have sufficient resources nor motivation to overcome this, it is assumed that the game is played only in the specific place and lighting which was used for training. Last thing to note is that specific codes have been assigned to the different gestures: 0 – no turn; 1 – clockwise turn; 2 – anticlockwise turn. Natural numbers were chosen because these labels are used in the game program for logical conditions, and because using characters or strings makes the AI training take multiple times longer.

```
krzysztofh@ip11:~/Documents/artificial-intelligence-and-computer-vision/Project$ python3 ai_extract_gesture.py

Classification Report:
              precision    recall  f1-score   support

           0       0.42      0.10      0.16    327021
           1       0.40      0.10      0.16    328077
           2       0.35      0.87      0.49    327942

    accuracy                           0.36    983040
   macro avg       0.39      0.36      0.27    983040
weighted avg       0.39      0.36      0.27    983040

Confusion Matrix:
 [[ 32998  27761 266262]
 [ 23005  32571 272501]
 [ 22517  21360 284065]]

Time taken preparing data: 3.50 seconds
Time taken training and testing: 748.81 seconds
```

Listing 3.2: Output of the AI training program.

4 Results

This chapter will present the final software and discuss the results.

4.1 The Game

Below is the ”Gestured Snake” program that combines the original simple game with all the components necessary for gesture control. Figures further down show the game and an example mask used while playing.

```
"""
   Snake terminal game controlled using hand gestures.

   Made by Krzysztof Hoszowski, February 2023.
"""

# Calculating results of AI model
from statistics import mode

# Loading AI model
from joblib import load

# Getting images from webcam
from cv2 import VideoCapture, imwrite

# Obtaining dataframes from images
from pd2img import Pd2Img

# Preprocessing images
from extract_hand import extract_hand

# Basic variables and functions
from snake import *


# Preparing webcam
WEBCAM_PORT = 0
webcam = VideoCapture(WEBCAM_PORT)

# Adjusting game speed to gesture controls
SNAKE_SPEED = 5

```

```python
# Loading AI classifier
clf = load("model.pkl")

# Controls variable
prediction = -1

# Limiting controls speed
counter = 0


### Main Program Loop ###
while True:
    counter += 1

    if counter == 2:
        # Resetting the timer
        counter = 0

        # Reading input using the camera
        result, image = webcam.read()

        if result:
            # Extracting the hand
            mask = extract_hand(image)
            imwrite("mask.png", mask)

            # Convert mask to dataframe
            df = Pd2Img("mask.png")

            # Recognizing the gesture
            predictions = clf.predict(df.df.iloc[:, 0:3])
            prediction = mode(predictions)

            print(prediction)

            if prediction == 1:
                dir_index += 1
                dir_index %= 4
            elif prediction == 2:
                dir_index -= 1
                dir_index %= 4
        else:
            print("No image detected. Please check your camera or settings.")

    # Alternative to webcam: Handling key events
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP:
                dir_index += 1
                dir_index %= 4
            if event.key == pygame.K_DOWN:
                dir_index -= 1
                dir_index %= 4

    # Moving the snake
    if directions[dir_index] == "UP":
        snake_position[1] -= 10
    if directions[dir_index] == "DOWN":
        snake_position[1] += 10
    if directions[dir_index] == "LEFT":
        snake_position[0] -= 10
    if directions[dir_index] == "RIGHT":
        snake_position[0] += 10

    # Snake body growing mechanism
    # If fruit and snake collide then score will be
   incremented by 10
    snake_body.insert(0, list(snake_position))
    if (
        snake_position[0] == fruit_position[0]
        and snake_position[1] == fruit_position[1]
    ):
        score += 10
        fruit_spawn = False
    else:
        snake_body.pop()

    if not fruit_spawn:
        fruit_position = [
            randrange(1, (WINDOW_X // 10)) * 10,
            randrange(1, (WINDOW_Y // 10)) * 10,
        ]

    fruit_spawn = True
    game_window.fill(BLACK)

    for pos in snake_body:
        pygame.draw.rect(game_window, GREEN, pygame.Rect(pos
   [0], pos[1], 10, 10))
```

```
    pygame.draw.rect(
        game_window, WHITE, pygame.Rect(fruit_position[0], fruit_position[1], 10, 10)
    )

    # Game Over conditions
    if snake_position[0] < 0 or snake_position[0] > WINDOW_X - 10:
        game_over(score)
    if snake_position[1] < 0 or snake_position[1] > WINDOW_Y - 10:
        game_over(score)

    # Touching the snake body
    for block in snake_body[1:]:
        if snake_position[0] == block[0] and snake_position[1] == block[1]:
            game_over(score)

    # Displaying score
    show_score(score, WHITE, "calibri", 20)

    # Refresh game screen
    pygame.display.update()

    # Frame Per Second / Refresh Rate
    fps.tick(SNAKE_SPEED)
```

Listing 4.1: End-user terminal game program.

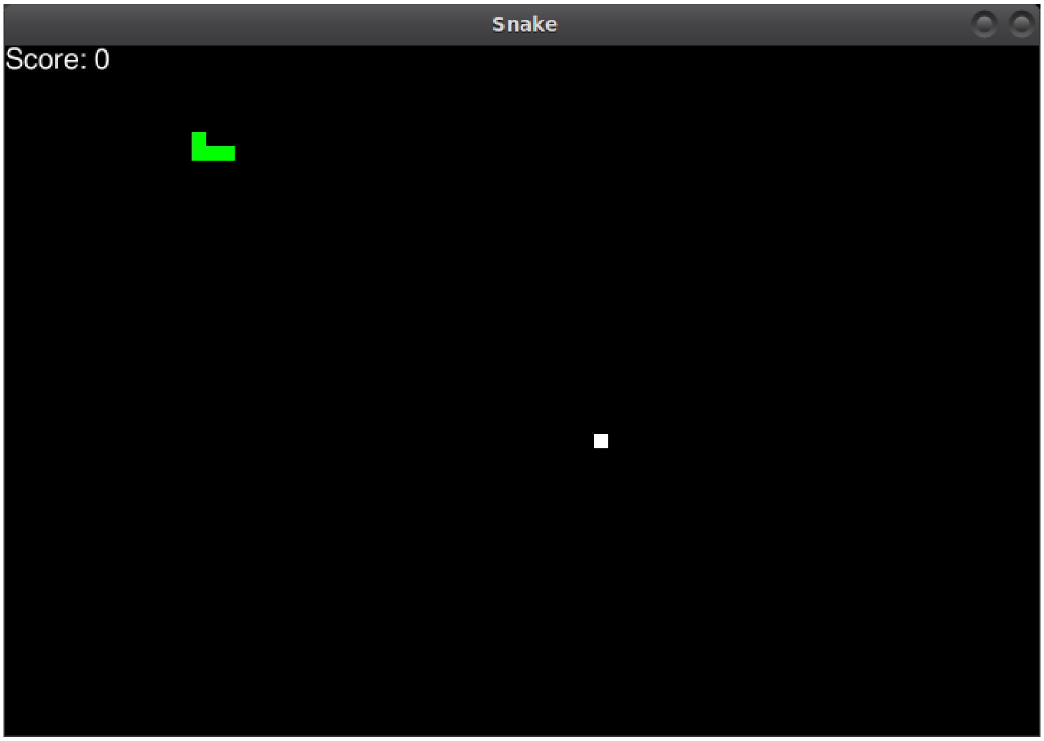

Figure 4.1: The Snake game screen. The snake is green, the white pixel is the fruit, and the snake mustn't touch itself or the window borders to keep playing.

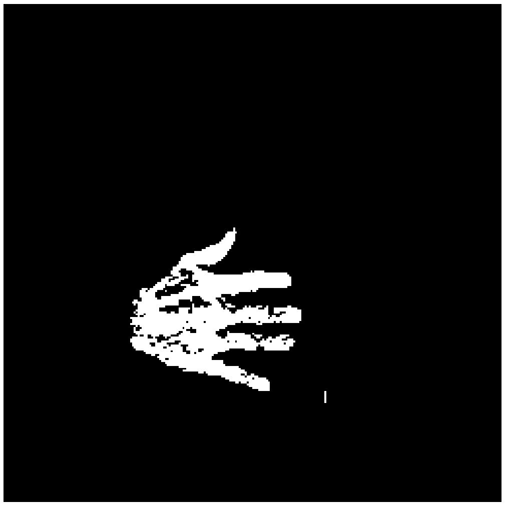

Figure 4.2: The processed image.

4.2 Conclusions

The main objective of the project was accomplished – the computer game responds to gestures made before its camera, which are preprocessed and fed through a neural network. All the individual parts work efficiently and as expected in the Python 3 framework. The one yet key problem is the image interpretation itself. For the purposes of a video game, it has to be not only accurate and robust, but also quite fast. It is really difficult to optimize one parameter without sacrificing the other. Yet, subjectively, our implementation is playable. Due to delays between gesturing and actual turn, the objective of the game may be changed. For example, it could be to avoid colliding as long as possible. Alternatively, the fruit could be made bigger. All in all, the project was a success. It greatly enhanced our knowledge of artificial intelligence and computer vision, including supervised machine learning. It also furthered our skills at teamwork and division of duties. C. Nzelibe focused on the CV part, and K. Hoszowski on the AI part.

5 Appendix

This chapter includes listings of other Python programs which were not presented in previous parts.

5.1 Simple Photo-taking

```python
"""
   Taking a number of photos (for AI training).

   Made by Krzysztof Hoszowski, February 2023.
"""

# Getting images from webcam
from cv2 import VideoCapture, imwrite

# Processing images
from extract_hand import extract_hand


# Preparing webcam
WEBCAM_PORT = 0
webcam = VideoCapture(WEBCAM_PORT)


for i in range(20):
    # Reading input using the camera
    result, image = webcam.read()

    if result:
        # Extracting the hand
        mask = extract_hand(image)

        # Save mask
        imwrite("mask.png", mask)  # Checking the images
        imwrite("".join([str(i), ".png"]), mask)  # Actual save
```

Listing 5.1: Simple script to take pictures for machine learning purposes.

5.2 Images to Dataframes Conversion

```python
"""
   Converting images into pandas dataframes and CSV files.

   Made by Krzysztof Hoszowski, February 2023.
"""

# Loading filepaths into Python in bulk
from glob import glob
# Converting images into pandas dataframes
from pd2img import Pd2Img
# Concatenating dataframes
from pandas import concat


# Loading the images
paths = [
    glob("images/input_clockwise/*.png"),
    glob("images/input_anticlockwise/*.png"),
    glob("images/input_nothing/*.png")]

# Output of function
dataframes = []


def im2df():
    """Take all images from given paths and return
    a dataframe for each path in a list."""

    for path in paths:
        # First run to add headers
        dframe = Pd2Img(path[0]).df

        # The remaining images
        for i in range(1, len(path)):
            dfi = Pd2Img(path[i]).df

            # Concatenating the dataframes
            dframe = concat([dframe, dfi])

        dataframes.append(dframe)
    return dataframes
```

Listing 5.2: Hard-coded software to convert images into pandas dataframe format.

5.3 Snake Core

```python
"""
   Simple Snake terminal game.

   Made by Krzysztof Hoszowski, February 2023.
"""

import sys
from time import sleep
from random import randrange
import pygame

## Function definitions
def show_score(score, color, font, size):
    """Displaying score obtained so far in the game."""

    # Creating font object score_font
    score_font = pygame.font.SysFont(font, size)

    # Create the display surface Score_surface
    score_surface = score_font.render("Score: " + str(score), True, color)

    # Create a rectangular object for the text surface object
    score_rect = score_surface.get_rect()

    # Displaying text
    game_window.blit(score_surface, score_rect)


def game_over(score):
    """\"Game over\" message with final score."""

    # Creating font object my_font
    my_font = pygame.font.SysFont("calibri", 60)

    # Creating a text surface on which text will be drawn
    game_over_surface = my_font.render("Final score: " + str(score), True, RED)

    # Create a rectangular object for the text surface object
    game_over_rect = game_over_surface.get_rect()

    # Setting position of the text
    game_over_rect.midtop = (WINDOW_X / 2, WINDOW_Y / 4)

    # Draw the text on screen
    game_window.blit(game_over_surface, game_over_rect)
    pygame.display.flip()

    # Quit the game after 3 seconds
    sleep(3)

    # Quitting
    pygame.quit()
    sys.exit(0)


### Initializing the game ###
# Constant snake speed / FPS
SNAKE_SPEED = 15

# Window size
WINDOW_X = 720
WINDOW_Y = 480

# Defining colors
BLACK = pygame.Color(0, 0, 0)
RED = pygame.Color(255, 0, 0)
GREEN = pygame.Color(0, 255, 0)
BLUE = pygame.Color(0, 0, 255)
WHITE = pygame.Color(255, 255, 255)

# Initializing pygame
pygame.init()

# Initialize game window
pygame.display.set_caption("Snake")
game_window = pygame.display.set_mode((WINDOW_X, WINDOW_Y))

# FPS (frames per second) controller
fps = pygame.time.Clock()

# Defining snake default position
snake_position = [140, 70]

# Defining initial 4 blocks of snake body
snake_body = [[130, 70], [120, 70], [110, 70], [100, 70]]

```

```
# Fruit position
fruit_position = [
    randrange(1, (WINDOW_X // 10)) * 10,
    randrange(1, (WINDOW_Y // 10)) * 10,
]

# Whether to spawn more fruit
fruit_spawn = True

# Setting default snake direction towards right
directions = ("UP", "RIGHT", "DOWN", "LEFT")
dir_index = 1

# Initial score
score = 0


### Main Program Loop ###
if __name__ == "__main__":
    while True:
        # Handling key events
        for event in pygame.event.get():
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_UP:
                    dir_index += 1
                    dir_index %= 4
                if event.key == pygame.K_DOWN:
                    dir_index -= 1
                    dir_index %= 4

        # Moving the snake
        if directions[dir_index] == "UP":
            snake_position[1] -= 10
        if directions[dir_index] == "DOWN":
            snake_position[1] += 10
        if directions[dir_index] == "LEFT":
            snake_position[0] -= 10
        if directions[dir_index] == "RIGHT":
            snake_position[0] += 10

        # Snake body growing mechanism
        # If fruit and snake collide then scores will be incremented by 10
        snake_body.insert(0, list(snake_position))
        if (
```

```python
            snake_position[0] == fruit_position[0]
            and snake_position[1] == fruit_position[1]
    ):
            score += 10
            fruit_spawn = False
    else:
            snake_body.pop()

    if not fruit_spawn:
            fruit_position = [
                randrange(1, (WINDOW_X // 10)) * 10,
                randrange(1, (WINDOW_Y // 10)) * 10,
            ]

    fruit_spawn = True
    game_window.fill(BLACK)

    for pos in snake_body:
            pygame.draw.rect(game_window, GREEN, pygame.Rect(pos[0], pos[1], 10, 10))
    pygame.draw.rect(
            game_window,
            WHITE,
            pygame.Rect(fruit_position[0], fruit_position[1], 10, 10),
    )

    # Game Over conditions
    if snake_position[0] < 0 or snake_position[0] > WINDOW_X - 10:
            game_over(score)
    if snake_position[1] < 0 or snake_position[1] > WINDOW_Y - 10:
            game_over(score)

    # Touching the snake body
    for block in snake_body[1:]:
            if snake_position[0] == block[0] and snake_position[1] == block[1]:
                game_over(score)

    # Displaying score
    show_score(score, WHITE, "calibri", 20)

    # Refresh game screen
```

```
        pygame.display.update()

        # Frame Per Second /Refresh Rate
        fps.tick(SNAKE_SPEED)
```

Listing 5.3: The basic Snake terminal game. It acts as a header for the final product.

Bibliography

[1] Hans Alemao *pd2img 0.0.3*. Python Package Index, released: Apr 23, 2022. Source: https://pypi.org/project/pd2img/