# wallprime の自動化と最適化

tax_free

## 1　概要

wallprime でより高いスコアを安定して獲得できるプログラムを開発した．その過程で，OCR 処理の正答率を向上させる前処理や multiprocessing を用いた並列処理による高速化について学習し，それらを実装した．

## 2　実行環境とソースコード

表 1: 実行環境

| | |
|---|---|
| OS | MacOS Big Sur 11.6 |
| CPU | Core i5-8259U |
| メモリ | 8GB LPDDR3 |
| Python version | Python 3.8.9 |
| Bluestacks version | Version 4.270.1 (2803) |
| 仮想環境 | venv |

ソースコードは，GitHub で確認できる [1]．以下，本文ではこれらのコード群を auto_wallprime と書く．

## 3　はじめに

### 3.1　モチベーション

SNS で wallprime を遊んでいる動画を見て興味を持ち，自分で遊んでみたが素因数分解が苦手で高いスコアを獲得できなかった．そこで，Python を使って wallprime を自動化し，最適化してより高いスコアを獲得できるプログラムを作ろうと思った．また，琉球大学 GSC で模型自動車の自動運転に関する画像分析について研究していたので，その知識を応用したいと思った．

### 3.2　wallprime とは

wallprime は壁に書かれた数字を素因数分解するゲームである．制限時間が存在し，制限時間は，下の方法に従って計算される．

制限時間の計算方法 [2]

(情報保護のため削除)

wallprime のスコア (単位:pt) は，制限時間内でのみ獲得できる．よって，直接的にスコアを最大化するのではなく，制限時間を最大化し，結果としてスコアを最大化するプログラムを開発した．

### 3.3　作成方針

最初の操作を除いて人間が操作せず，自動で wallprime をプレイでき，30000pt 以上を獲得できるプログラムを開発すること．

## 4　OCR 処理の説明と使用した主なライブラリ

### 4.1　OCR 処理の説明

OCR は，Optical Character Reader(または Recognition) の略で，画像データのテキスト部分を認識し，文字データに変換する光学文字認識機能のことである．今回は壁に書かれた数字を読み取るために使用した．本文では，OCR 処理は，後述の read_num の OCR 処理を指す．

### 4.2　使用した主なライブラリ

以下のライブラリを主に用いた．詳細なバージョンは，requirements.txt[1] に記載した．

- PyOCR : スクリーンショットを OCR 処理するために，PyOCR の Tesseract を使用した．
- PyAutoGUI : スクリーンショットの取得と，ゲーム画面への入力のために使用した．
- opencv-python : スクリーンショットを前処理して，OCR 処理の正答率を向上するために使用した．

## 5　auto_wallprime の概要

auto_wallprime の核となる solver.py について説明する．

solver.py の本体は Solver クラスであり，Solver クラスは下の 5 つのメソッドを持つ．

- run(引数:なし，返り値:なし):
  Solver のインスタンスが作成されると，実行される．各メソッド間の値の受け渡しを行う．
- read_num(引数:なし，返り値:非負整数とその素因数):
  スクリーンショットを OCR 処理し，その数字を素因数分解して返す．特定の数字で詰むことを防ぐために，異なった 2 つの Tesseract のモード (eng, snum) で冗長化した．eng は英語に，snum はシリアルナンバーの読み取りに特化している．
- check(引数:read_num の返り値，返り値:引数の適当な方の素因数，または 0):
  read_num の 2 つの返り値から適当な 1 つを選び，その素因数を返す．どちらも不適当である場合は，0 を返す．
- click(引数:入力する素因数，返り値:なし):
  素因数をゲーム画面に入力する．乱数によって入力座標をずらすことで，入力が判定されないケースを防いでいる．
- pfactorization(引数:自然数， 戻り値:引数の素因数):
  2 以上 53 以下の素数で素因数分解を行い，OCR 処理の間違いを発見できるようにしている．

## 6 目標達成の課題と解決方法

### 6.1 課題

制限時間の計算方法より，一発クリアは制限時間を最大化するために重要である．一発クリアの割合を大きくするために解決する必要があった課題を，OCR 処理と処理速度の 2 つの点で考えた．

OCR 処理について

- OCR 処理の正答率が 40% 程度と低い．
- check の選択基準が最適化されていない．

処理速度について

- run の処理が遅い
- 処理と処理の間隔が最適化されていない

### 6.2 解決方法

まず，OCR 処理に関する課題の解決方法 (以下，OCR 処理最適化) を考えた．下の画像は，スクリーンショットの例である．スクリーンショットには，下の 3 つの正答率を下げている要素 (obs) があると考えた．


- obs_1: 画像の底辺に対して，数字が傾いている．
- obs_2: 黒色の背景に白色の数字が書かれている．
- obs_3: 左下にゲーム内キャラクター (ランナー) の腕が描画されている．

これらの要素を取り除き，OCR 処理の正答率を上げるために，スクリーンショットに前処理をした．行った前処理 (method) は，下の 5 つである．

行った前処理

- method_1: スクリーンショットを回転する．
- method_2: スクリーンショットを二値化し，色を反転する．
- method_3: スクリーンショットにぼかし処理をする．
- method_4: スクリーンショット内の輪郭を抽出する．
- method_5: スクリーンショット内のランナーの腕を隠す．

それぞれの method について説明する．

method_1 は，obs_1 を解決するために作成した． cv2.warpAffine と cv2.getRotationMatrix2D を使い，スクリーンショットを OCR 処理に最適な角度に回転した．

method_2 は，obs_2 と obs_3 を解決するために作成した．cv2.threshold と cv2.THRESH_BINARY_INV によってスクリーンショットを二値化し，背景を白色に，数字を黒色に変換した．cv2.threshold の閾値がある値より大きくなると，ランナーの腕は背景と同じ白色になった．

method_3 は，obs_3 を解決するために作成した．cv2.GaussianBlur によって画像をぼかし，ランナーの腕を数字の一部と認識しないようにした．

method_4 は，obs_2 と obs_3 を解決するために作成した．cv2.Canny と cv2.findContours を使い，スクリーンショット内の輪郭を抽出した．cv2.Canny でランナーの腕を取り除き，白色の背景に黒色で輪郭を描画した．

method_5 は, obs_3 を解決するために作成した．ランナーの腕は決まった範囲に描画される．その範囲を黒色に置換し，ランナーの腕を消去した．

read_num の 2 つの結果を eng と snum の正答率を考慮して選択するよう変更した．$P_{eng}$ と $P_{snum}$ を eng と snum の正答率としたとき，eng の結果を選ぶ確率 $w_e$ と snum の結果を選ぶ確率 $w_s$ は，

$$w_e = \frac{P_{eng}}{P_{eng} + P_{snum}}, \quad w_s = \frac{P_{snum}}{P_{eng} + P_{snum}}$$

で与えられる．

次に，処理速度に関する課題の解決方法 (以下，処理速度最適化) を考えた．

run の中で処理に時間がかかっているメソッドを特定するために，メソッドごとの処理時間を計った．その結果が表 2 である．全て，112 回測定した．

表 2: メソッドごとの実行時間

| 関数名 | 平均値 | 分散 |
|---|---|---|
| read_num | 0.91098 | 0.00116 |
| check | 3e-05 | 0.000 |
| click | 1.1349 | 0.107 |
| pfactorization | 0.00000 | 0.000 |

表 2 から，read_num と click の処理にかかる時間が大きいことが分かる．read_num の処理速度と click を呼び出す間隔，処理と処理の間隔を最適化するために，下の 3 つの方法 (solver) を調べた．

解決方法

- solver_1: multiprocessing による OCR 処理の並列化
- solver_2: click を呼び出す間隔 ($t_{in}$) の調整
- solver_3: スクリーンショット取得間隔 ($t_{sp}$) の調整

それぞれの solver について説明する．

solver_1 は，multiprocess モジュールの pool オブジェクトによって，OCR 処理を並列化した．pool オブジェクトは作成コストが大きいので，ループごとに使い捨てるのではなくループと切り離して使用した．

solver_2 は，$t_{in}$ を最適化するために作成した．$t_{in}$ が非常に小さいとき，入力を重複して判定され，正しく素因数が入

力されない．一発クリアには，正しく素因数を入力することが必要である．

solver_3 は，$t_{sp}$ を最適化するために作成した．$t_{sp}$ を小さくすれば，当然処理は早く実行される．しかし，スクリーンショットが壁の数字を全て含まず，一発クリアでない可能性が大きくなる．

### 6.3 実験と評価方法

OCR 処理最適化と処理速度最適化の評価は，下の評価方法に従った．評価に使用する dataset は，表 3 である．

- OCR 処理最適化：parameter.txt[1] のパラメータを用いた各 method ごとに，3 つの dataset を OCR 処理したスコアを計算し，それらを合計した値で評価した．スコアはゲーム内スコアの計算方法 [2] に従い，read_num の eng と snum の結果が異なる場合は，$w_e$ と $w_s$ のうち高い方を用いた期待値をそのスクリーンショットのスコアとした．
- 処理速度最適化：solver_1 と OCR 処理を並列化していないプログラムが 3 つの dataset を OCR 処理したときにかかる時間を合計した値で評価した．pool に 6 コアを割り当て，交互に 26 回行った．solver_2 は，parameter.txt のパラメータを試し，入力が正しく判定されるかを調べた．solver_3 は，parameter.txt のパラメータを試し，OCR 処理への影響を調べた．$t_{in}$ は 0.01 秒，$t_{sp}$ は 0.1 秒ごとに試した．

表 3: 3 つの dataset の特徴

| | |
|---|---|
| dataset_1 | 1 から 3 桁の数字から成る (360 枚) |
| dataset_2 | 4 から 6 桁の数字から成る (991 枚) |
| dataset_3 | 7 桁以上の数字から成る (178 枚) |

dataset を数字の大きさで分割した理由は，数字の大きさと OCR 処理の正答率に関係があるかを調べるためである．

OCR 処理最適化の評価は，正答率ではなくスコアを用いた．正答率だけでは，dataset ごとに正答率が異なるが，合計した正答率が同じになった組を評価できないからである．

各 method を評価した結果から，高いスコアが期待できる method の組み合わせを method_6 とした．

処理速度最適化を評価したとき，OCR 処理は method_6 を含む評価した method のなかで，最もスコアが高いものを用いた．

## 7 結果

表 4 は，OCR 処理最適化の評価の結果である．それぞれの method の中で最もスコアが高くなったパラメータのときの値を掲載した．表 4 の default は，前処理なしで OCR 処理したときの値である．method_6 は，method_5 と method_2 をこの順番で組み合わせた．method_2 が表 4 のスコアを獲得したとき，前処理されたスクリーンショット内にランナーの腕は見られなかった．

表 5 は，各 method のスコアが最も高くなったパラメータを用いたときの $w_e$ と $w_s$ である．

表 6 は，solver_1 と OCR 処理を並列化していないプログラムが 3 つの dataset を OCR 処理するのにかかった時間である．default が OCR 処理を並列化していないプログラムで，どちらも method_2 を用いた．

表 7 より $t_{in}$ は，0.07 秒が最適となった．

表 8 より $t_{sp}$ は，3.4 秒が最適となった．

それぞれの詳細な値は，evaluation_method, evaluation_solver[1] に記載した．

## 8 考察

まず，OCR 処理最適化の結果について考察する．

method_2 と method_5 が method_1,3,4 と比べて高いスコアを獲得し，method_1,3,4 の中でも obs_3 を解決するために作成した method_3,4 は method_1 に比べて高いスコアを獲得したことが，表 4 から分かる．これらから，スクリーンショットに含まれるランナーの腕が OCR 処理の正答率を下げていた一番の要因であったと推測される．

また，method_2 と method_5 を比較したとき，dataset_1,2 のスコアは，method_5 が method_2 を上回った．しかし，dataset_3 では，method_2 のスコアが method_5 より高い．これは，桁が大きい数字で構成された dataset_3 では，method_5 が黒色に置換する範囲に数字の一部が含まれたことが原因だと考えられる．

ある detaset で $w\_e > w\_s$ となった method は白黒反転処理を含む method_2,4 だけであったことが，表 5 から分かる．よって，eng の学習データは，多くの場合で白色の背景に黒色の文字が書かれている紙やデジタル文書であると推測できる．

次に，処理速度最適化の結果について考察する．OCR 処理を並列化することで，188% 高速化した．表 6 の default の値と 3 つの dataset に含まれるスクリーンショットの枚数から，default が一枚の OCR 処理に必要な時間を計算できる．その時間は 0.33 秒となるが，この値は表 2 の read_num の $\frac{1}{3}$ 倍程度である．よって，スクリーンショットを取得する時間や関数の呼び出しに約 0.6 秒かかっていたことが分かる．

ゲーム内で壁をパンチするモーションにかかる時間は，入力した素因数の数によって決まり，5 個のとき，約 1.7 秒である．また，壁を壊して新しい壁に移動するまでの時間は約 1.8 秒であり，この間制限時間は消費されず，これは壁の数字に依らない．ここで，3 つの dataset の数の桁の平均は 4.7 桁，素因数の個数の平均は 5.0 個である．これらの値と制限時間の計算方法から，長い目で見ると制限時間が減らない $(t_{in}, t_{sp})$ の条件を計算できる．ゲーム内で登場する数は 3 つの dataset と同じ分布であると仮定し，一発クリアの割

表 4: 各 method のスコア

| | default | method_1 | method_2 | method_3 | method_4 | method_5 | method_6 | 満点 |
|---|---|---|---|---|---|---|---|---|
| dataset_1 | 3004 | 6193 | 15969 | 15132 | 9265 | 16264 | 16359 | 17066 |
| dataset_2 | 31390 | 67211 | 102701 | 77208 | 74185 | 103179 | 103247 | 107798 |
| dataset_3 | 7113 | 10981 | 34357 | 7496 | 3949 | 32345 | 32084 | 36102 |
| sum | 41507 | 84385 | 153027 | 99836 | 94059 | 151788 | 151690 | 160966 |

表 5: 各 method の $w_e$ と $w_s(w_e, w_s)$

| | default | method_1 | method_2 | method_3 | method_4 | method_5 | method_6 |
|---|---|---|---|---|---|---|---|
| dataset_1 | (0.42, 0.58) | (0.23, 0.77) | (0.12, 0.88) | (0.02, 0.98) | (0.26, 0.74) | (0.0, 1.0) | (0.04, 0.96) |
| dataset_2 | (0.49, 0.51) | (0.05, 0.95) | (0.68, 0.32) | (0.10, 0.90) | (0.83, 0.17) | (0.0, 1.0) | (0.22, 0.78) |
| dataset_3 | (0.30, 0.70) | (0.16, 0.84) | (0.19, 0.81) | (0.27, 0.73) | (0.82, 0.18) | (0.0, 1.0) | (0.21, 0.79) |

表 6: OCR 処理にかかった時間

| | default | solver_1 |
|---|---|---|
| dataset_1 | 118.9 | 63.20 |
| dataset_2 | 327.2 | 173.2 |
| dataset_3 | 59.02 | 31.90 |
| sum | 505.16 | 268.28 |

表 7: ゲーム画面への入力間隔 ($t_{in}$)

| $t_{in}$ の値 | 誤入力の発生頻度 |
|---|---|
| $t_{in} \leq 0.03$ | ほとんど毎回 |
| $0.04 \leq t_{in} \leq 0.06$ | 無視できない程度 |
| $0.07 \leq t_{in}$ | ほとんどない |

表 8: スクリーンショット取得の間隔 ($t_{sp}$)

| $t_{sp}$ の値 | スクリーンショットへの影響 |
|---|---|
| $t_{sp} \leq 2.9$ | 全ての桁を含まない場合が多い |
| $3.0 \leq t_{sp} \leq 3.3$ | 全ての桁を含まない場合がある |
| $3.4 \leq t_{sp}$ | 全ての桁を含まない場合が少ない |

合を 80% とした.

$$0.33 + 5.0 \times t_{in} + 1.7 + (t_{sp} - 1.7 - 1.8) < 4.7 \times 3.7 \times 0.1 \times 1.8$$
$$\therefore 5.0 t_{in} + t_{sp} < 4.6 \qquad \text{(条件 } A\text{)}$$

$(t_{in},\ t_{sp})$ が条件 A を満たすとき，長い目で見れば制限時間は減少しない.

表 8 と表 7 の $(t_{in},\ t_{sp} = 0.07,\ 3.4)$ は $5.0 t_{in} + t_{sp} = 3.75$ であるから，条件 A を満たす．また，$4.6 - 3.75 = 0.85$ であるからある程度のバッファーを持っている．よって，$(t_{in},\ t_{sp} = 0.07,\ 3.4)$ は実際のゲームでも，有用な値であるといえる.

## 9 最後に

wallprime の自動化と最適化に必要な課題を考え，それを改善した．その過程で OCR 処理の正答率を向上させる前処理や OCR 処理の並列化，処理と処理の間隔ついて実験し考察した．その結果，OCR 処理の正答率は 2 倍以上向上し，処理速度は 1.5 倍以上早くなった．そして，wallprime で 37450pt を取ることができた.

また，機械学習を用いれば，より正答率を上げることが可能だと考えている．加えて，学習データに一部が欠けたスクリーンショットを含むことで安定性の向上が期待される.

## 10 単独か否か

開発は単独で行ないましたが、活動実績報告書は (プライバシー保護のため削除) さん協力の元、文章構成についてアドバイスを頂きました。

## 参考文献


[1] `https://github.com/taxfree-python/auto_wallprime`

[2] 開発元の QuizKnock 様より，学業利用を条件に公開していただきました.