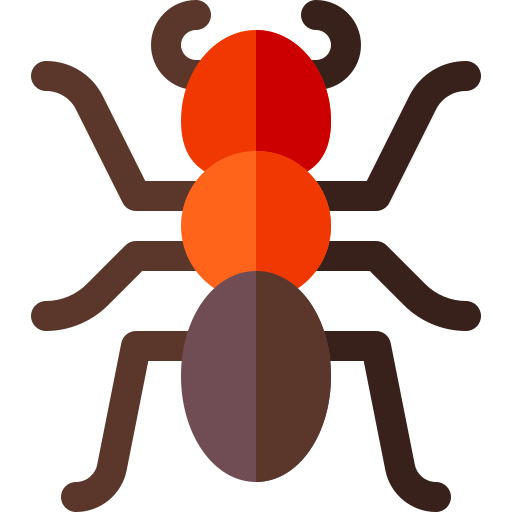
ŁUKASZ STANISZEWSKI

**SYMULATOR MRÓWKI LANGTONA**

OFICJALNA DOKUMENTACJA



**SPIS TREŚCI**

[ROZDZIAŁ I. OPIS PROJEKTU 3](#_Toc29895213)

[**WSTĘP** 3](#_Toc29895214)

[**ZASADY DZIAŁANIA** 3](#_Toc29895215)

[**WYMAGANIA PROJEKTU** 3](#_Toc29895216)

[ROZDZIAŁ II. STRUKTURA PROJEKTU 4](#_Toc29895217)

[**OGÓLNY PODZIAŁ** 4](#_Toc29895218)

[**SCHEMAT DZIAŁANIA** 4](#_Toc29895219)

[**NAJWAŻNIEJSZE UŻYTE BIBLIOTEKI** 5](#_Toc29895220)

[**STRUKTURA MODUŁÓW I KLAS** 5](#_Toc29895221)

[**NAJISTOTNIEJSZE FUNKCJE** 5](#_Toc29895222)

[**ALGORYTMIKA** 6](#_Toc29895223)

[FAZA WSTĘPNA 6](#_Toc29895224)

[FAZA SYMULACJI 6](#_Toc29895225)

[ROZDZIAŁ III. INSTRUKCJA UŻYTKOWNIKA 7](#_Toc29895226)

[**WSTĘP** 7](#_Toc29895227)

[**START** **KONSOLOWY** 7](#_Toc29895228)

[**START** **GUI** 8](#_Toc29895229)

[**OPCJA TWORZENIA PLIKU (1)** 8](#_Toc29895230)

[**OPCJA LADOWANIA PLIKU** 9](#_Toc29895231)

[**OPCJA TWORZENIA PLIKU Z PRAWDOPODOBIEŃSTWEM** 9](#_Toc29895232)

[**WYNIK SYMULACJI** 9](#_Toc29895233)

[ROZDZIAŁ IV. PRZEPROWADZONE TESTY 10](#_Toc29895234)

[**TESTY JEDNOSTKOWE** 10](#_Toc29895235)

[**WAŻNY PRZYKŁAD** 10](#_Toc29895236)

[**TEST SYMULACJI** 11](#_Toc29895237)

[ROZDZIAŁ V. EWENTUALNY ROZWÓJ 11](#_Toc29895238)

[ROZDZIAŁ VI. WYKORZYSTANE ZASOBY 11](#_Toc29895239)

# ROZDZIAŁ I. OPIS PROJEKTU

## **WSTĘP**

Projekt polega na implementacji automatu komórkowego, tzw. [**Mrówki Langtona**](https://pl.wikipedia.org/wiki/Mr%C3%B3wka_Langtona).

Rys 1 - trasa mrówki po wykonaniu 7000 kroków

## **ZASADY DZIAŁANIA**

1. Mrówka porusza się na planszy o określonych wymiarach, podzielonej na kwadratowe komórki (pola) w dwóch możliwych kolorach: czarnym i białym
2. Jeśli mrówka znajduje się na polu białym to obraca się w lewo (o kąt prosty), zmienia kolor pola na czarny i przechodzi na następną komórkę.
3. Jeśli mrówka znajduje się na polu czarnym to obraca się w prawo (o kąt prosty), zmienia kolor pola na biały i przechodzi na następną komórkę.
4. Jeśli mrówka doszła do końca planszy i próbuje za nią wyjść to przechodzi na losowo wybraną sąsiednią komórkę, która leży na planszy (np. jeśli mrówka znajduje się w dolnym lewym rogu planszy i próbuje przejść jedną komórkę w dół, to powinna wykonać losowy ruch w górę, lub w prawo).

## **WYMAGANIA PROJEKTU**

1. Mrówka porusza się po obrazie, którego wartość początkowa to jedna z następujących możliwości:
   1. biały obraz o wymiarach podanych przez użytkownika
   2. obraz ładowany przez użytkownika (powinien być czarno-biały)
   3. czarno-biała obraz, gdzie czarne piksele zostały losowo wygenerowane z zadanym prawdopodobieństwem
2. Wyjściem programu jest zapisana seria obrazów obrazująca wynik 'spaceru' mrówki - jeden obraz na każdy ruch. Liczba kroków określona jest przez użytkownika.

# ROZDZIAŁ II. STRUKTURA PROJEKTU

## **OGÓLNY PODZIAŁ**

*Strukturę omawianego projektu najwygodniej podzielić na kilka punktów:*

**DOKUMENTACJA** – folder *doc*

**TESTY** – folder *tests* i zawarte w nim testy (czyt. rozdział IV)

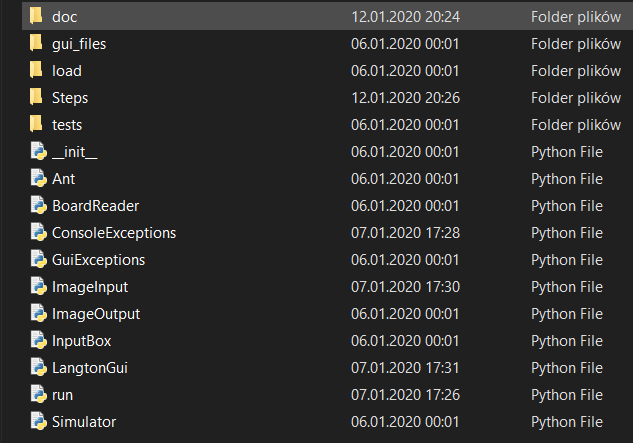
**WYJĄTKI** – pliki *ConsoleExceptions.py* i *GuiExceptions.py*

**KLASY** **ODPOWIEDZIALNE** ZA **SYMULACJE** – pliki *Ant*.*py*, *BoardReader.py* i *Simulator.py*

**FUNKCJE** **KONWERTUJACE** NA **LINII** **PROJEKT**-**OBRAZ** – pliki *ImageInput.py i ImageOutput.py*

**CZĘŚĆ** **INTERFEJSOWĄ** – folder *gui\_files,* plik *LangtonGui.py*

**FOLDER** **WYNIKOWY** – folder *Steps*



Rys 2 - końcowy wygląd folderu z projektem

## **SCHEMAT DZIAŁANIA**

**Schemat działania projektu obrazuje lista kroków:**

1. Uruchom projekt
2. Wybierz opcje
3. Wprowadź potrzebne dane
4. Program wykonuje symulacje
5. KONIEC

## **NAJWAŻNIEJSZE UŻYTE BIBLIOTEKI**

Użyte w projekcie biblioteki:

1. **Numpy** - korzystanie z klas do wygodnej pracy z wektorami dwuwymiarowymi
2. **Pillow** - biblioteka do obsługi obrazów
3. **Pygame** – biblioteka do tworzenia interfejsu graficznego
4. Dodatkowo częściej spotykane biblioteki typu **time, os, math** czy **sys**

## **STRUKTURA MODUŁÓW I KLAS**

## **NAJISTOTNIEJSZE FUNKCJE**

1. **Ant.go**(kierunek) – porusza mrowke w dany kierunek
2. **BoardReader.create\_board\_from\_picture**(obrazek) – tworzy tablice dwuwymiarowa z danego obrazka
3. **ImageOutput.create\_picture\_from\_board**(tablica) – tworzy obrazek z danej tablicy
4. **Simulator.take\_step**() – funkcja odpowiedzialna za pojedynczy krok symulacji

## **ALGORYTMIKA**

### FAZA WSTĘPNA

Omówmy teraz dokładniej całe działanie projektu. Użytkownik uruchamiając plik run.py włącza symulator. W zależności od parametru w pliku run.py może uruchomić grę za pomocą:

1. **KONSOLI**
2. **GRAFICZNEGO INTERFEJSU UŻYTKOWNIKA (GUI)**

Teraz ma do wyboru wybranie 1 z 3 opcji:

1. **STWORZENIA BIAŁEGO OBRAZKA** (podaje wysokość i szerokość oczekiwanej grafiki)
2. **ZAŁADOWANIA SWOJEGO OBRAZKA O MAX WIELKOSCI 300x300** (podaje ścieżkę do niego)
3. **STWORZENIA BIAŁEGO OBRAZKA Z PRAWDOPODOBIENESTWEM WYSTAPIENIA CZARNYCH PIXELI NA NIM** (zarówno wysokość i szerokość jak i prawdopodobieństwo jest podawane przez użytkownika)

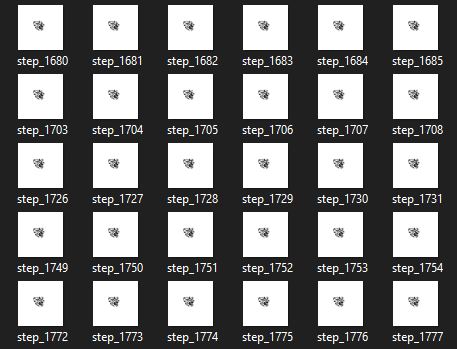
Dodatkowo w każdej z 3 tych opcji musi podać również liczbę kroków jaką ma wykonać mrówka (jeżeli poda więcej niż 30000, mrówka wykona 30000 kroków).

Następnie program sprawdza poprawność wprowadzonych danych – odpowiednie typy, wielkości itd.

Gdy nie wystąpi żaden wyjątek, przechodzimy do następnej fazy projektu.

### FAZA SYMULACJI

Fazę tę rozpoczyna program od skonwertowania wygenerowanego obrazka do tablicy dwuwymiarowej. Przedstawia ona plansze po jakiej się mrówka porusza – to właśnie w jej centrum zostaje umieszczona mrówka. W pętli wykonującej się N-krotnie (N – liczba kroków podana przez użytkownika we wcześniejszej fazie) mrówka robi kolejne kroki według algorytmu zawartego w I rozdziale dokumentacji. W międzyczasie każdy stan planszy jest przerabiany za pomocą odpowiedniej funkcji na czarno-biały obrazek, który jest zapisywany do folderu *Steps* z odpowiednia nazwa w postaci *step\_[nr kroku].*

Po wykonaniu pętli, następuje koniec programu.

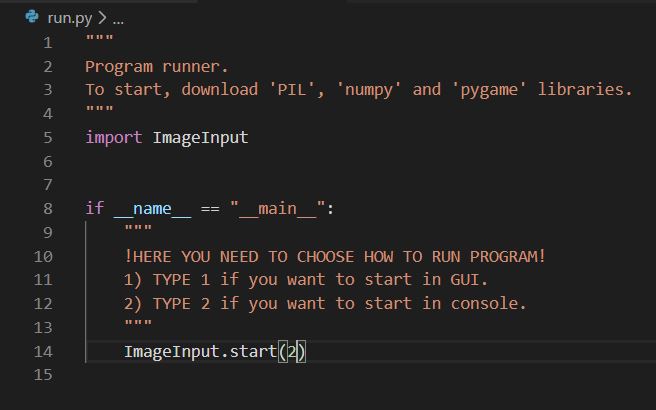
Rys 3 – fragment przykładowego wyglądu folderu Steps po zakończonym programie

# ROZDZIAŁ III. INSTRUKCJA UŻYTKOWNIKA

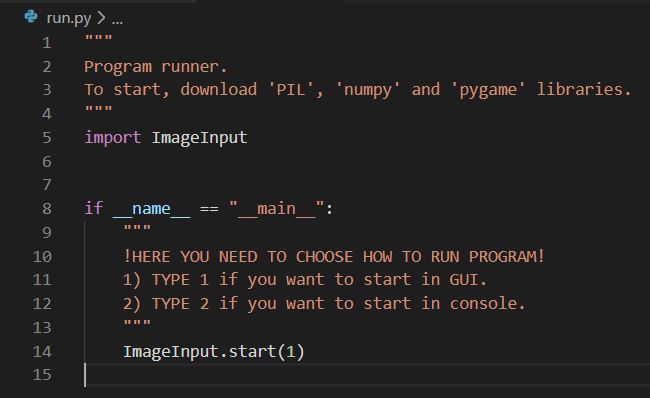
## WSTĘP

Aby uruchomić program, należy wejść do folderu z projektem, za pomocą edytora tekstowego otworzyć plik ***run*.*py*** i w funkcji start() wpisać odpowiednio **1** lub **2:**

**1 –** uruchomienie programu w gui

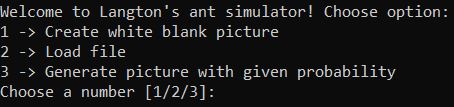
**2 –** uruchomienie programu w konsoli

Rys 4.2 - konfiguracja startu konsolowego



Rys 4.1 - konfiguracja startu gui

## START KONSOLOWY

Następnie po zapisaniu pliku ***run*.*py*** możemy go uruchomić. W konsoli powita nas ekran taki jak na zdjęciu.

Rys 5 - ekran startowy w wersji konsolowej

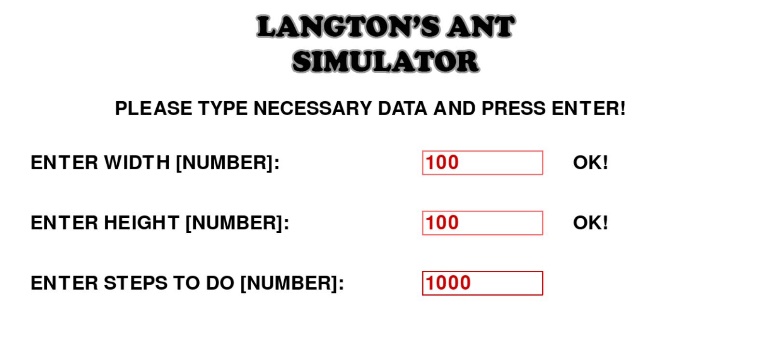
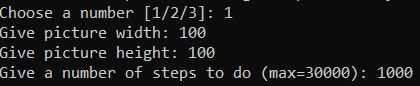
## START GUI

 Po uruchomieniu pliku ***run.py*** z parametrem **1,** pojawi nam się następujące okno.

Rys 6 - ekran startowy w wersji gui

## OPCJA TWORZENIA PLIKU (1)

Pierwszą z wybranych opcji, jest opcja tworzenia białego obrazka. Po wybraniu jej, należy wpisać odpowiednie dane, tak jak na obrazkach. Dane mają też swoje ograniczenia:

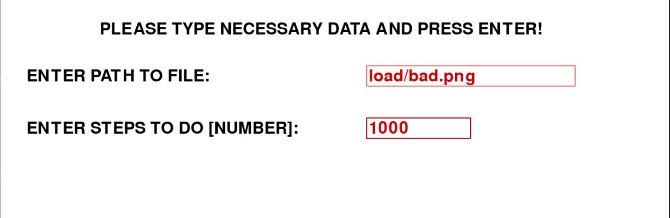
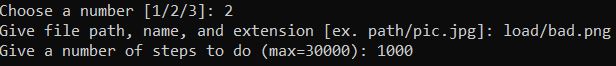
* 1. Wysokość [**height**] obrazka powinna być liczbą całkowitą z zakresu 0-300
  2. Szerokość [**width**] obrazka powinna być liczbą całkowitą z zakresu 0-300
  3. Liczba kroków [**steps**] powinna być liczbą z zakresu 1-30000

Rys 7.2 - ekran 1 opcji w konsoli

Rys 7.1 - ekran 1 opcji w gui

## OPCJA LADOWANIA PLIKU

Drugą z możliwych opcji do wybrania jest ładowanie własnego pliku. Ładowany plik warto umieścić w folderze ***load***, specjalnie do tego przeznaczonym. Nie ma wymogów co do kolorów na obrazku – zostanie on przekonwertowany na czarno-biały obiekt. Istotny jest jednak rozmiar obrazka. Ładowany obrazek nie może mieć szerokości lub wysokości większej niż 300 pixeli.

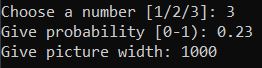


Rys 8.2 - opcja 2 w konsoli

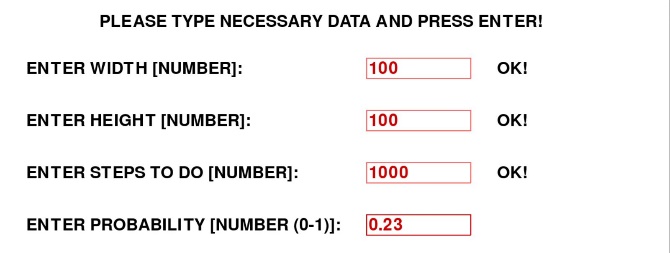
Rys 8.1 - 2 opcja w gui

## OPCJA TWORZENIA PLIKU Z PRAWDOPODOBIEŃSTWEM

Trzecią z możliwych opcji do wybrania jest tworzenie własnego pliku z prawdopodobieństwem wystąpienia czarnego pixela na nim. Poza atrybutami pojawiającymi się w opcji 1, występuje tu również atrybut **probability** – wymaga podania liczby rzeczywistej z zakresu 0-1.



Rys 9.2 - opcja 3 w konsoli



Rys 9.1 - opcja 3 w gui

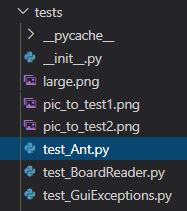
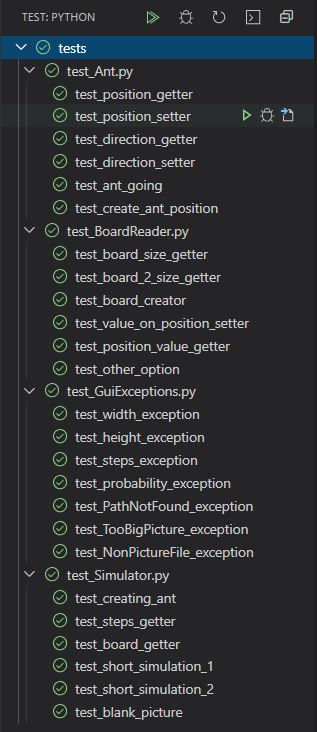
## WYNIK SYMULACJI

Po zakończeniu symulacji, należy wejść do folderu **steps**. Własnie tam znajduje się wynik całej symulacji w postaci serii obrazków, tak jak pokazane to zostało na **rys 3**.

# ROZDZIAŁ IV. PRZEPROWADZONE TESTY

## TESTY JEDNOSTKOWE

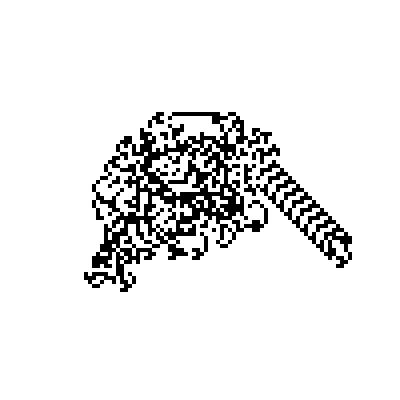
Projekt ten zawiera liczne testy, sprawdzające prawidłowe działanie zawartych w nim klas, metod, funkcji czy atrybutów. Zaczynając od sprawdzenia pozycji mrówki na planszy aż po całe jej załadowanie z pliku.



Rys 10 - wszystkie przeprowadzone testy (działają!)

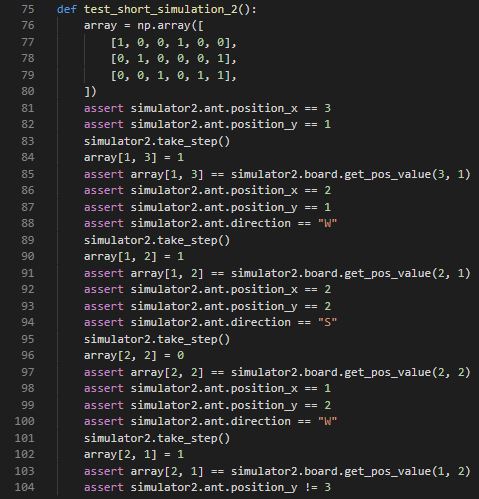
Rys 11 - pliki z testami

## WAŻNY PRZYKŁAD

 Istotną właściwością mrówki Langtona jest trasa, jaką wykonuje ona po zrobieniu 10000 kroków. Tworzy ona wtedy charakterystyczny kształt nazwany „*autostradą*”, nie inaczej jest w tym projekcie.

Rys 12 - droga wykonana przez mrówkę po 11000 krokach

## TEST SYMULACJI

 Podjąłem w projekcie próbę przeprowadzenia krótkiej symulacji na małym obrazku (np. o wymiarach 5x3), w której mogłem przewidzieć pierwsze 4 kroki mrówki. Następny krok program ze względu na mały obszar wybiera już losowo (stąd dalsze testy nie mają sensu – nie da się przewidzieć następnego kroku).

Rys 13 - test symulacji



Rys 14.5 - mrowka skreca w prawo, idzie na pixel w 2 kolumnie i 3 rzedzie

Rys 14.4 - mrowka skreca w lewo, idzie na pixel w 3 kolumnie i 3 wierszu

Rys 14.3 - mrowka skreca w lewo, idzie na pixel w 3 kolumnie i 2 wierszu

Rys 14.2 - zmiana pozycji srodkowego pixela

Rys 14.1 - załadowany obrazek jako tablica 0-1

Podobne symulacje przeprowadziłem również dla ładowanego obrazka 5x5 (4 kroki) i stworzonego białego obrazka 5x5 (10 kroków). Wszystkie symulacje wyszły prawidłowo.

# ROZDZIAŁ V. EWENTUALNY ROZWÓJ

Projekt można by było udoskonalić w możliwość pojawienia się kilku mrówek naraz na obrazie. O ile założenie te zostałoby sformułowane już w początkowej fazie projektowania (a tak naprawdę to w fazie planowania algorytmiki), zaimplementowanie pomysłu mogłoby dojść do skutku. Na obecnym etapie wdrożone algorytmy stworzone są z przeznaczeniem dla automatu jednej mrówki i ewentualna modyfikacja spowodowałaby zmianę struktury większości programu. Dlatego też kwestię tą postanowiłem umieścić w tym rozdziale dokumentacji.

# ROZDZIAŁ VI. WYKORZYSTANE ZASOBY

1. Ikona programu - [*https://www.flaticon.com/free-icon/ant\_2253561?term=ant&page=1&position=14*](https://www.flaticon.com/free-icon/ant_2253561?term=ant&page=1&position=14)