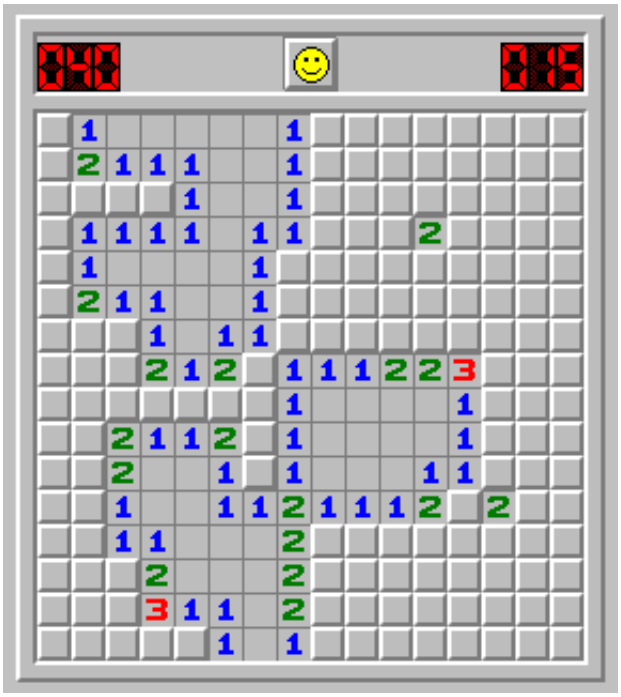
Gra Saper w Pygame - Projekt zaliczeniowy

Michał Pomarański

Styczeń 2014

1. Wprowadzenie

1.1. Saper



Rysunek 1: Przykładowa implementacja sapera

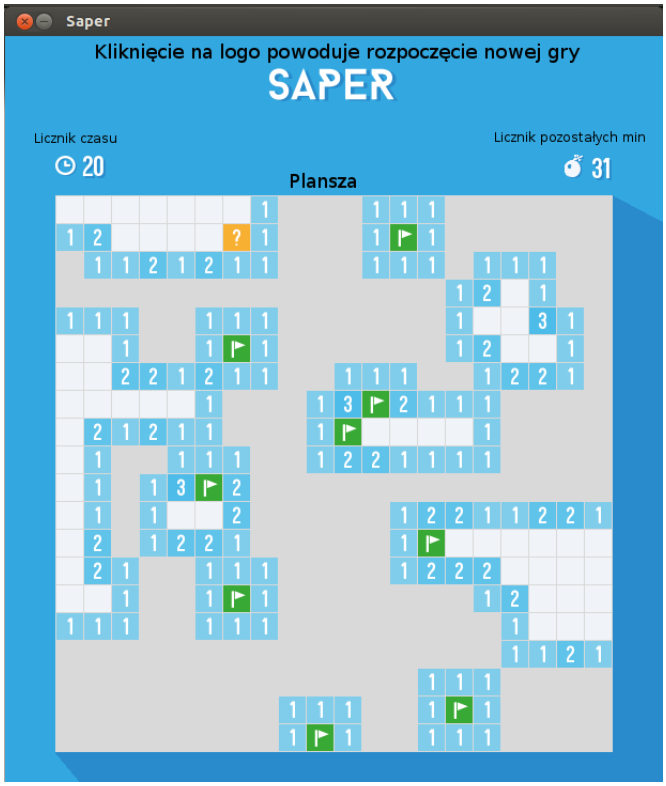
Opis (tytuł oryginalny Minesweeper) – klasyczna jednoosobowa gra komputerowa napisana w 1981 roku przez Roberta Donnera, dostępna jako akcesorium w każdym systemie Microsoft Windows do wersji 7. Od wersji 8 i RT dostępne do ściągnięcia w sklepie Windows (istnieją też wersje dla innych systemów operacyjnych). Gra polega na odkrywaniu na planszy poszczególnych pól w taki sposób, aby nie natrafić na minę. Na każdym z odkrytych pól napisana jest liczba min, które bezpośrednio stykają się z danym polem (od zera do ośmiu). Jeśli oznaczymy dane pole flagą, jest ono zabezpieczone przed odsłonięciem, dzięki czemu przez przypadek nie odsłonimy miny.

Strategia Zależy ona głównie od wielkości pola i liczby min. Im mniejsze pola, tym mniej czasu należy poświęcać pojedynczym polom. Ze wzrostem wielkości pola więcej czasu należy poświęcić na poszczególne przypadki, które są bardziej złożone. Ponieważ układ min jest losowy, powtórzenie układu praktycznie nie powinno się zdarzyć. Jednakże możliwa jest sytuacja, w której plansza jest niejednoznacznie zdefiniowana, tzn. nie można określić, w którym dokładnie miejscu znajduje się mina. Sytuacje takie zdarzają się m.in. w końcowych etapach rozgrywki (pozostają 2 pola do odsłonięcia i 1 mina do wstawienia). Niektóre z wersji gry nie dopuszczają do takiej sytuacji generując ponownie układ planszy.

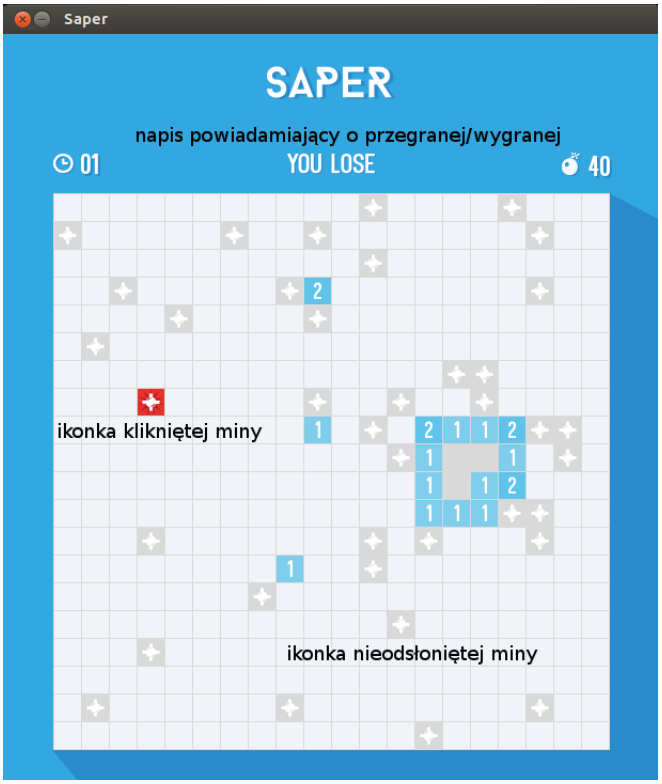
1.2. Pygame

Opis Pygame – to stworzona przez Pete Shinnersa biblioteka graficzna przeznaczona do tworzenia gier komputerowych oraz aplikacji multimedialnych w języku Python. Do działania wymaga biblioteki SDL, przy wykorzystaniu której dostarcza modułów pozwalających na wyświetlanie grafiki, odtwarzanie dźwięków, śledzenie czasu, obsługę myszy i joysticka, obsługę CD, czy renderowanie czcionek TTF. Pygame jako nakładka na SDL jest wieloplatformowa i umożliwia pracę na różnych systemach operacyjnych m.in. na Windows, Linux, MacOS. Biblioteka Pygame stanowi wolne oprogramowanie i jest dystrybuowana na zasadach licencji LGPL.

2. Opis interfejsu



- Szare pole oznaczają odsłonięte pola
- Białe pola oznaczają nieodsłonięte pola
- Pola z cyframi oznaczają ile min znajduje się dookoła danego pola
- Kiedy spodziewamy się, że za którymś polem znajduje się mina, oznaczamy to pole flagą (zielone pola z flagą)
- Kiedy nie jesteśmy pewni czy na danym polu znajduje się mina, możemy oznaczyć to pole pytajnikiem (pomarańczowe pole z pytajnikiem)



- Czerwone pole oznacza minę którą kliknęliśmy (przegrywając przy tym grę)
- Szare pola z minami oznaczają pozostałe nieodkryte miny

3. Uwagi na temat implementacji

3.1. Main.py

Jest to główny plik obsługujący grę. Na początku zostaje zainicjalizowane okno programu, jego wielkość, nazwa, a następnie jego tło.
Znajduje się tutaj główna pętla programu, w której pobieramy wszystkie zdarzenia (np. kliknięcie myszką, kliknięcie przycisku na klawiaturze itd.) oraz przesyłamy je do obsługujących je funkcji.

```
# -*- coding: utf-8 -*-

import pygame
import os
import sys
from pygame.locals import *

from minefield import *
from colors import *
from game import *
from difficulty import *
from globals import *


def start_new_game(difficulty):
    """ funkcja rozpoczynająca nową grę """
    game = Game(Minefield(difficulty))
    game.new_game()
    return game


def initialize_screen():
    """ funkcja ustawiająca okno oraz tło"""
    # menu
    background.fill(colors.backgroundColor)
    background.blit(saper_logo, logo_pos)


def main():
    """ główna funkcja programu """
    pygame.font.init()
    pygame.init()
    pygame.display.set_caption('Saper')

    initialize_screen()

    # ustawianie planszy
    easy = Difficulty(20, 20, 40, 25, "easy")

    pygame.display.flip()
```

```
    # rozpoczynanie gry
    game = start_new_game(easy)

    # glowna petla
    while True:
        for event in pygame.event.get():
                # obsługa kliknięcia przycisku zamykania
                    programu
            if event.type == QUIT:
                pygame.quit()
                sys.exit()
            # obsługa klawiszy
            if event.type == KEYDOWN:
                # obsługa przycisku ESC
                if event.key == K_ESCAPE:
                    pygame.quit()
                    sys.exit()
                # obsługa przycisku 'n'
                if event.key == K_n:
                    game = start_new_game(easy)

            if event.type == MOUSEBUTTONDOWN:  # handler
                poruszania myszka
                # jeżeli lewe kliknięcie myszki
                if event.button == 1:
                    # sprawdzamy czy gracz kliknął napis
                        SAPER
                    if logo_pos.collidepoint(event.pos):
                        game = start_new_game(easy)

                    # obsługa kliknięcia na pola
                    game.left_click(event.pos)
                # obsługa prawego klikniecią myszki
                if event.button == 3:
                    game.right_click(event.pos)
            # obsługa zegara
            if event.type == UPDATECLOCKEVENT:
                game.update_clock()

            # update obrazków
            screen.blit(background, (0, 0))

            pygame.display.flip()

if __name__ == '__main__':
    main()
```

3.2. Game.py

Jest to drugi najważniejszy plik programu. Klasa game obsługuje całą grę. Na początku inicjalizuje obiekt planszy, zegara oraz licznika min. Następnie obiekt gry czeka na wydarzenia przesłane z głównej pętli programu i obsługuje je.

```
# -*- coding: utf-8 -*-
""" główna klasa obsługująca całą grę"""

from Block import Block
from counter import Counter
import pygame
import os
import globals
from clock import *
import colors

# DEBUG MODE
DEBUG = False


class Game:

    def __init__(self, minefield):
        self.clock = 0
        self.minefield = minefield
        self.clickable = True
        self.clock = None
        self.first_click = True

        # ustawienia licznika
        self.counter = None

        # napis wygranej/przegranej
        self.text = None

    def init_counter(self):
        """ inicjalizacja licznika"""
        rect = self.minefield.game_area_pos
        size = 27
        pos = pygame.Rect(rect)
        pos.x = rect.right - 20
        pos.y -= (10 + size)
        self.counter = Counter(self.minefield.difficulty.
            mines_number, pos)

    def init_clock(self):
        """ inicjalizacja zegara"""
        rect = self.minefield.game_area_pos
        size = 27
        # pozycja zegara
```

```
        pos = pygame.Rect(rect)
        pos.x = rect.left
        pos.y -= (10 + size)
        # ustawianie sprite'a zegara
        return Clock(pos)

    def find_collide_rect(self, pos):
        """ metoda sprawdzająca które pole zostało klikni
            ęte """
        for b in self.minefield.get_blocks():
            if b.rect.collidepoint(pos):
                return b

    def left_click(self, pos):
        """ metoda obsługująca lewe kliknięcie myszki """
        if self.clickable:

            # znajdujemy kliniete pole
            b = self.find_collide_rect(pos)

            # jesli pierwsze klikniecie, to rozpoczynamy
                odliczenie zegara
            if self.first_click:
                self.first_click = False
                self.minefield.set_mines(b)
                self.clock.start_clock()

            if b:
                if b.covered and not b.flagged and not b.
                    question:
                    if b.mines_surrounding == 0:
                        self.minefield.ripple_effect(b)
                    else:
                        b.uncover()
                    # print(b.rect)
                    # print(pos)

            if self.is_game_over():
                self.end_game(b)
            self.minefield.update()

    def right_click(self, pos):
        """ metoda obsługująca prawe kliknięcie myszki
            """
        if self.clickable:
            b = self.find_collide_rect(pos)
            if b:
                if b.covered:
                    if not b.flagged and not b.question
                        and self.counter.mines > 0:
```

```
                        b.flag()
                        # update counter
                        self.counter.mines -= 1
                        self.counter.update()
                    elif b.flagged:
                        Block.question(b)
                        # update counter
                        self.counter.mines += 1
                        self.counter.update()
                    elif b.question:
                        b.cover()
                # print(b.rect)
                # print(pos)

            if self.is_game_over():
                self.end_game()
            self.minefield.update()

    def is_game_over(self):
        """ metoda sprawdzająca czy gra została
            zakonczona (wygrana lub przegrana gracza)"""
        # jesli miny jeszcze nie sa ustawione
        if not self.minefield.are_mines_set:
            return False
        # jesli trafimy na mine
        if any(not b.covered and b.mined for b in self.
            minefield.get_blocks()):
            return True
        # jesli oflagujemy wszystkie miny
        # wszystkie pola ktore sa zaminowane musza byc
            oflagowane
        if all(b.flagged for b in self.minefield.
            get_blocks() if b.mined):
            return True
        # jesli odkryjemy wszystkie niezaminowane pola
        if all(not b.covered for b in self.minefield.
            get_blocks() if not b.mined):
            return True

        return False

    def end_game(self, clicked_block):
        """ metoda kończąca grę (pokazanie wszystkich min
            , pokazanie napisu wygranej/przegranej,
            zatrzymanie zegara)"""
        # wygrana
        if self.check_win():
            self.text = pygame.image.load(os.path.join('
                ikonki', 'won.png'))
        # przegrana
```

```
        else:
            self.text = pygame.image.load(os.path.join('
                ikonki', 'lose.png'))
        # odkrywanie wszystkich min
        self.minefield.uncover_mines(clicked_block)
        # umiejscowienie napisu
        textpos = self.text.get_rect()
        textpos.centerx = self.minefield.game_area_pos.
            centerx
        textpos.y = self.minefield.game_area_pos.top - 27
            - 10
        globals.background.blit(self.text, textpos)
        # wylaczenie mozliwosci odkrywania pol
        self.clickable = False
        # wylaczenie zegara
        self.clock.stop_clock()

    def new_game(self):
        """ Metoda rozpoczynająca nową grę. """
        self.minefield.draw()
        if not self.clock:
            self.clock = self.init_clock()
        # czyszczenie grafik
        self.clock.clear_clock()
        self.clock.stop_clock()
        self.clock.show_clock()
        self.init_counter()
        self.counter.clear_counter()
        self.counter.show_counter()
        self.clear_win_lose()
        if DEBUG:
            self.minefield.debug()

    def check_win(self):
        """ Metoda sprawdzająca czy gracz wygrał """
        # jesli trafimy na mine - przegrana
        if any(b.mined and not b.covered for b in self.
            minefield.get_blocks()):
            return False
        return True

    def update_clock(self):
        self.minefield.update()
        self.clock.update()

    def clear_win_lose(self):
        """ metoda usuwająca napis wygranej/przegranej
            """
        tmp = pygame.Surface((110, 30))
        tmp.fill(colors.backgroundColor)
```

```
            tmp_pos = tmp.get_rect()
            tmp_pos.centerx = self.minefield.game_area_pos.
                centerx
            tmp_pos.y = self.minefield.game_area_pos.top - 27
                - 10
            globals.background.blit(tmp, tmp_pos)


def main():
    pass

if __name__ == "__main__":
    main()
```

3.3. Globals.py

W pliku globals.py znajdują się globalne wartości wykorzystywane przez wiele innych plików.

```
# -*- coding: utf-8 -*-
""" Moduł zawierający globalne zmienne używane przez
    wszystkie klasy """
import pygame
import os
import colors

screen_height = 700
screen_width = 615
# ustawianie wielkosci okna
global screen
screen = pygame.display.set_mode([screen_width,
    screen_height])

# czcionka
counter_and_clock_font = os.path.join('ikonki', "Kenzo.
    otf")

# ustawienia tla
global background
background = pygame.Surface(screen.get_size())
background = background.convert()

saper_logo = pygame.image.load(os.path.join('ikonki', '
    saper_logo.png'))
logo_pos = saper_logo.get_rect()
logo_pos.centerx = background.get_rect().centerx
logo_pos.y += 30

# event odliczania czasu
UPDATECLOCKEVENT = pygame.USEREVENT + 1


def main():
    pass

if __name__ == "__main__":
    main()
```

3.4. Colors.py

W tym pliku znajdują się definicje różnych kolorów używanych w programie.

```
# -*- coding: utf-8 -*-
""" moduł ze wszystkimi kolorami użytymi w programie """
import pygame
import os
# kolory
whiteColor = pygame.Color(250, 250, 250)
redColor = pygame.Color(255, 0, 0)
game_area_color = pygame.Color("#d9d9d9")
orangeColor = pygame.Color(255, 201, 0)
uncoveredColor = pygame.Color(0, 147, 191)
coveredColor = pygame.Color(41, 156, 0)
questionColor = pygame.Color(0, 255, 153)
bombColor = pygame.Color(200, 0, 255)
bombhint = pygame.Color(250, 192, 106)
backgroundColor = pygame.Color("#33A8E0")
napisyColor = pygame.Color("#FFFFFF")
shadowColor = pygame.Color("#298bcb")


def main():
    pass

if __name__ == "__main__":
    main()
```

3.5. Minefield.py

Klasa minefield zajmuje się obsługą planszy tzn. Zmienianiem ikonek na planszy, wyszukiwanie klikniętego pola, ustawianie min na planszy, ustawianiem wielkości planszy itd.

```
# -*- coding: utf-8 -*-
""" Jedna z głównych klas obsługująca planszę """

import pygame
import os
import colors
from Block import *
import random
import globals
import itertools
import time

# DEBUG MODE
DEBUG = True
calls = 0


class Minefield:

    def __init__(self, difficulty):
        self.mines_left = difficulty.mines_number
        self.difficulty = difficulty
        self.blocks = []
        self.game_area = None
        self.are_mines_set = False
        self.rows = 0
        self.columns = 0

    def get_blocks(self):
        """ Zwraca jednowymiarową tablicę wszystkich pól
            """
        return list(itertools.chain.from_iterable(self.
            blocks))

    def set_mines(self, block):
        """ Losuje pola, które mają być zaminowane oraz
            ustawia na nich miny """
        random.seed()
        for _ in range(self.difficulty.mines_number):
            blocks = self.get_blocks()
            selected_block = blocks[random.randint(0, len
                (blocks) - 1)]
            if block != selected_block:
                selected_block.mine()
```

```
        self.set_mines_surrounding()
        self.are_mines_set = True

    def set_mines_surrounding(self):
        for i in range(len(self.blocks)):
            for j in range(len(self.blocks[i])):
                if self.blocks[i][j].mined:
                    self.set_mines_around(i, j)

    def set_mines_around(self, i, j):
        """ Metoda pomocnicza sprawdzająca ile min jest
            ustawionych dookoła każdego pola """

        for n in range(i - 1, i + 2):
            for m in range(j - 1, j + 2):
                if n < self.rows and m < self.columns and
                    n >= 0 and m >= 0 and self.blocks[n][
                    m]:
                    self.blocks[n][m].mines_surrounding
                        += 1

    def ripple_effect(self, block):
        """ Metoda odkrywa pola rekurencyjnie dopóki
            natrafia na pola nieotoczone minami """
        if not block.covered:
            return
        if block.mined or block.flagged or block.question
            :
            return
        block.uncover()
        if block.mines_surrounding != 0:
            return

        row_clicked, column_clicked = self.find(block)
        blocks = self.blocks
        for i in range(3):
            for j in range(3):
                current_row = row_clicked + i - 1
                current_column = column_clicked + j - 1
                try:
                    if current_row >= self.rows or
                        current_column >= self.columns:
                        continue
                    if current_row < 0 or current_column
                        < 0:
                        continue
                    if blocks[current_row][current_column
                        ].covered:
                        self.ripple_effect(blocks[
                            current_row][current_column])
```

```
                except IndexError as e:
                    #print(current_row, current_column)
                    pass
        return

    def find(self, elem):
        """ metoda znajdująca obiekt danego pola"""
        for row, i in enumerate(self.blocks):
            try:
                column = i.index(elem)
            except ValueError:
                continue
            return row, column
        return -1

    def draw(self):
        """ metoda wyświetlająca planszę """
        self.init_game_area()
        self.init_blocks()
        self.update()

    def update(self):
        """ metoda wyświetlająca planszę po kliknięciu
            myszki """
        for b in self.get_blocks():
            self.game_area.blit(b.image, (b.posx, b.posy)
                )
        globals.background.blit(self.game_area, (self.
            game_area_pos))

    def init_blocks(self):
        """ inicjalizacja planszy """
        size = 25
        padding = 1
        posx, posy = padding, padding
        tmp_block = []
        i, j = 0, 0
        while posy + size < self.difficulty.height:
            j += 1
            block = self.create_block(posx, posy, size, i
                , j)
            tmp_block.append(block)
            posx += size + padding
            # nowy rzad
            if posx + size > self.difficulty.width:
                i += 1
                j = 0
                posx = padding
                posy += size + padding
                self.blocks.append(tmp_block)
```

```
            tmp_block = []
        self.rows = len(self.blocks)
        self.columns = len(self.blocks[0])

    def create_block(self, posx, posy, size, i, j):
        """ Metoda tworzy nowe pole """
        block = Block(posx, posy, size, i, j)
        offset = self.game_area_pos.topleft
        block.rect.topleft = (posx + offset[0], posy +
            offset[1])
        return block

    def init_game_area(self):
        """ Metoda tworząca planszę """
        size = self.difficulty.width, self.difficulty.
            height
        self.game_area = pygame.Surface(size)
        self.game_area = self.game_area.convert()
        self.game_area.fill(colors.game_area_color)
        self.game_area_pos = self.game_area.get_rect()
        self.game_area_pos.centerx = globals.background.
            get_rect().centerx
        self.game_area_pos.bottom = globals.background.
            get_rect().height - 30

        # ustawianie cienia planszy
        left_bottom = self.game_area_pos.bottomleft
        bottom_right = self.game_area_pos.bottomright
        top_right = self.game_area_pos.topright
        screen_right = (globals.screen.get_rect().right,
            top_right[1] + 25)
        screen_rightbottom = globals.screen.get_rect().
            bottomright
        screen_leftbottom = (
            left_bottom[0] + 25, globals.screen.get_rect
                ().bottom)

        print(left_bottom)
        print(bottom_right)
        print(top_right)
        print(screen_right)

        pygame.draw.polygon(background, colors.
            shadowColor, [
            left_bottom,
            bottom_right,
            top_right,
            screen_right,
            screen_rightbottom,
            screen_leftbottom,
```

```
            left_bottom])

    def debug(self):
        """ Metoda która pokazuje gdzie ukryte są miny (
            tylko dla trybu debugowania) """
        for b in self.get_blocks():
            if b.mined:
                b.image.fill(colors.bombhint)

    def uncover_mines(self, clicked_block):
        """ Metoda odsłaniająca wszystkie pola """
        for b in self.get_blocks():
            if b.mined and b != clicked_block:
                b.uncover()
            if b.mined and b == clicked_block:
                b.uncover(exploded=True)


def main():
    pass

if __name__ == "__main__":
    main()
```

3.6. Difficulty.py

Klasa difficulty opisuje liczbę min na planszy oraz wielkość planszy.

```
# -*- coding: utf-8 -*-
""" klasa odzwierciedlająca poziom trudności. Zawiera
    takie dane jak: wielkość planszy, ilość min na planszy
    , wielkość pola """


class Difficulty:

    def __init__(self, width, height, mines_number,
        field_size, name):
        self.name = name
        self.mines_number = mines_number
        self.height = height * field_size + (height + 1)
        self.width = width * field_size + (width + 1)


def main():
    pass

if __name__ == "__main__":
    main()
```

3.7. Counter.py

Klasa counter odzwierciedla licznik min.

```
# -*- coding: utf-8 -*-
""" klasas obsługująca licznik pozostałych
    nieznalezionych jeszcze min"""
import globals
import colors
import pygame
import os


class Counter:

    def __init__(self, mines, pos):
        self.mines = mines
        self.pos = pos
        self.show_counter()

    def show_counter(self):
        """ metoda wyświetlająca licznik """
        ikona_bomby = pygame.image.load(os.path.join('
            ikonki', 'bomba.png'))
        counter_font = pygame.font.Font(globals.
            counter_and_clock_font, 27)
        text = counter_font.render(
            str(self.mines).zfill(2), 1, colors.
                napisyColor)

        # ustawianie cienia
        counter_shadow = counter_font.render(
            str(self.mines).zfill(2), 1, colors.
                shadowColor)
        shadow_pos = pygame.Rect(self.pos)
        shadow_pos.x += 2
        shadow_pos.y += 2

        globals.background.blit(counter_shadow,
            shadow_pos)
        globals.background.blit(text, self.pos)
        pos2 = pygame.Rect(self.pos)
        pos2.right -= 25
        pos2.y -= 1

        globals.background.blit(ikona_bomby, pos2)

    def update(self):
        """ metoda zmieniająca licznik """
        # jesli gra sie zakonczyla, zatrzymaj zegar
        if self.mines < 0:
```

```
            return
        self.clear_counter()
        self.show_counter()

    def clear_counter(self):
        """ metoda usuwająca licznik """
        tmp = pygame.Surface((3 * 36 + 2, 30))
        tmp.fill(colors.backgroundColor)
        globals.background.blit(tmp, self.pos)


def main():
    pass

if __name__ == "__main__":
    main()
```

3.8. Clock.py

Klasa clock odzwierciedla licznik czasu oraz zajmuje się rozpoczynaniem/kończeniem odliczania czasu.

```
# -*- coding: utf-8 -*-
""" klasas obsługująca zegarek """

import globals
import colors
import pygame
import os


class Clock:

    def __init__(self, pos):
        """ początkowe wartości"""
        self.time = 0
        self.pos = pos
        self.running = True

    def show_clock(self):
        """ metoda wyświetlająca zegar """
        ikona_zegara = pygame.image.load(os.path.join('
            ikonki', 'zegarek.png'))
        clock_font = pygame.font.Font(globals.
            counter_and_clock_font, 27)
        text = clock_font.render(
            str(self.time).zfill(2), 1, colors.
                napisyColor)
        globals.background.blit(ikona_zegara, self.pos)
        # ustawianie pozycji
        pos2 = pygame.Rect(self.pos)
        pos2.x += 25
        pos2.y -= 2
        # ustawianie cienia
        clock_shadow = clock_font.render(
            str(self.time).zfill(2), 1, colors.
                shadowColor)
        shadow_pos = pygame.Rect(pos2)
        shadow_pos.x += 2
        shadow_pos.y += 2

        globals.background.blit(clock_shadow, shadow_pos)
        globals.background.blit(text, pos2)

    def update(self):
        """ metoda zwiększająca czas """
        # jesli gra sie zakonczyla, zatrzymaj zegar
        if not self.running:
```

```
            return
        self.time += 1
        self.clear_clock()
        self.show_clock()

    def clear_clock(self):
        """ metoda usuwająca zegar """
        tmp = pygame.Surface((3 * 36 + 2, 30))
        tmp.fill(colors.backgroundColor)
        globals.background.blit(tmp, self.pos)

    def start_clock(self):
        """ metoda rozpoczynająca odliczanie zegara"""
        pygame.time.set_timer(globals.UPDATECLOCKEVENT,
            1000)
        self.running = True

    def stop_clock(self):
        """ metoda zatrzymująca odliczanie zegara """
        self.running = False


def main():
    pass

if __name__ == "__main__":
    main()
```

3.9. Block

.py Klasa block odzwierciedla pojedyńcze pole na planszy oraz zajmuje się zmienianiem ikonki danego pola. Zawiera także współrzędne każdego pola.

```
# -*- coding: utf-8 -*-
"""klasa odwzwierciedlająca jedno konkretne pole na
    planszy"""

import pygame
import os
import colors
from globals import *


class Block(pygame.sprite.Sprite):

    def __init__(self, posx, posy, size, i, j):
        """ inicjalizacja stanow pola, ustawianie tła itd
            ."""
        pygame.sprite.Sprite.__init__(self)
        self.size = size
        self.covered = True
        self.mined = False
        self.flagged = False
        self.question = False
        self.mines_surrounding = 0
        self.posx = posx
        self.posy = posy
        self.indexes = i, j
        self.init_image()
        self.rect = self.image.get_rect()

    def init_image(self):
        """ inicjalizacja tła"""
        self.image = pygame.image.load(os.path.join('
            ikonki', 'aktywne.png'))
        self.image = self.image.convert()

    def uncover(self, exploded=False):
        """ metoda obsługująca kliknięcie lewym
            przyciskiem myszy na pole"""
        if self.mined and not exploded:
            self.image = pygame.image.load(os.path.join('
                ikonki', 'mina.png'))
        elif self.mined and exploded:
            self.image = pygame.image.load(os.path.join('
                ikonki', 'mina1.png'))
        else:
            # jesli dookola nie ma min, nie ustawiamy
                cyfry "0"
```

```
                self.image = pygame.image.load(os.path.join('
                    ikonki', 'nieaktywne.png'))
                # ustawianie napisu na polach
                if self.mines_surrounding != 0:
                    self.image = pygame.image.load(
                        os.path.join('ikonki', '%d.png') %
                            self.mines_surrounding)
            self.covered = False

        def flag(self):
            """ metoda ustawiająca flagę jako tło """
            self.image = pygame.image.load(os.path.join('
                ikonki', 'flag.png'))
            self.flagged = True

        def question(self):
            """ metoda ustawiająca pytajnik jako tło """
            self.flagged = False
            self.question = True
            self.image = pygame.image.load(os.path.join('
                ikonki', 'mark.png'))

        def cover(self):
            """ metoda zasłaniająca pole"""
            self.flagged = False
            self.question = False
            self.image = pygame.image.load(os.path.join('
                ikonki', 'aktywne.png'))

        def mine(self):
            """ metoda ustawiająca bombę na polu """
            self.mined = True


    def main():
        pass

    if __name__ == "__main__":
        main()
```

4. Literatura wykorzystana przy tworzeniu projektu

Dokumentacja Pygame: http://www.pygame.org/docs/
Stackoverflow: http://stackoverflow.com/questions/tagged/python+pygame