ISSN 2307-9851

НАУКОВО-МЕТОДИЧНИЙ ЖУРНАЛ

Виходить 8 разів на рік

Видається з лютого 1998 року

Засновники:
Інститут педагогіки НАПН України,
Інститут інформаційних технологій і засобів навчання НАПН України,
Редакція журналу

Журнал видається за сприяння Міністерства освіти і науки України
Свідоцтво про реєстрацію серія КВ №12217–1101ПР від 17.01.2007

Передплатний індекс 74248

Журнал включено до Переліку наукових фахових видань України у галузі педагогічних наук, Наказ МОН України від 29.09.2014 року №1081

Журнал індексується:

Реферативна база даних "Україніка наукова"

РИНЦ

Google Scholar

Затверджено Вченою радою Інституту педагогіки НАПН України, протокол №14 від 8 жовтня 2018 р.

Головний редактор
ЛАПІНСЬКИЙ В. В.

Заступник головного редактора
КАЛІНІНА Л. М.

E-mail: csf22101@ukr.net

Тел. 044 481 37 38

Офіційний сайт журналу:
www.csf221.wordpress.com

# КОМП'ЮТЕР у школі та сім'ї

№6 (150) ∈ 2018

## ЗМІСТ



***На першій – четвертій сторінках обкладинки***

***25 листопада 2018 року*** *за участю науковців відділів економіки та управління загальною середньою освітою, математичної та інформатичної освіти Інституту педагогіки НАПН України*

***організовано й проведено:***

***науково - практичний семінар*** *«Дисемінація впровадження в освітній процес інноваційного досвіду експериментальної діяльності — потенційний ресурс підвищення якості надання освітніх послуг»*

***круглий стіл*** *«Математична та інформатична освіта 12-річної школи: якій їй бути»*

*Докладні звіти щодо виконання Всеукраїнських педагогічних експериментів — у наступних номерах*

## МОДЕЛИРОВАНИЕ ПЛАНИМЕТРИЧЕСКИХ КОНФИГУРАЦИЙ В СРЕДЕ GEOGEBRA

***Зеленяк Олег Петрович,***
*кандидат педагогических наук,*
*учитель математики и информатики Александрийского коллегиума,*
*Заслуженный учитель Украины,*
*zlnk@ukr.net*

***Аннотация***. В статье акцентируется внимание на важности неформального изучения компьютерного моделирования в школьных курсах математики и информатики. Обоснованы актуальность и целесообразность создания алгоритмов для построения графических компьютерных моделей геометрических конфигураций в среде динамической геометрии (СДГ) GeoGebra. На основе исследований и практического опыта автора предложена классификация планиметрических конфигураций (статические, динамические, статически-динамические) и их дифференциация по четырем уровням сложности. Приведены авторские экстремальные задачи, решение которых требует интегрированного применения знаний по информатике, геометрии, алгебре и математическому анализу .

***Ключевые слова:*** компьютерное моделирование, геометрия, планиметрия, интегрирование знаний, GeoGebra

## SIMULATION OF PLANIMETRIC CONFIGURATIONS IN GEOGEBRA

*Zelenyak Oleg Petrovich,*
*Ph.D., teacher of mathematics and computer science of the Alexandria Collegium,*
*Honored Teacher of Ukraine,*
*zlnk@ukr.net*

***Annotation***. The article presents the results of a study of the experience of using modern software in the teaching of geometry. The relevance and expediency of using computer models of geometrical configurations in the GeoGebra dynamic geometry (SDG) environment in the process of teaching geometry are substantiated. Based on the research and practical experience of the author, a classification of planimetric configurations (static, dynamic, static-dynamic) and their differentiation into four levels of complexity are proposed. The author's extremal learning problems are presented, their solution requires the integrated application of knowledge in computer science, geometry, algebra and mathematical analysis.

***Keywords:*** computer modeling, geometry, planimetry, knowledge integration, GeoGebra

* * *

**УДК 37.01:004.4’2**

# RUBY 2D — БАГАТОПЛАТФОРМЕНА БІБЛІОТЕКА ДЛЯ СТВОРЕННЯ 2D-ПРОГРАМ МОВОЮ RUBY

**Рудик Олександр Борисович**

*кандидат фізико-математичних наук, доцент Інституту післядипломної педагогічної освіти Київського університету імені Бориса Грінченка,*
*rudykob@gmail.com,*
ORCID ID 0000-0003-3676-0688

***Анотація***. Стаття висвітлює основи роботи з Ruby2D — багатоплатформною бібліотекою для створення 2D-програм мовою Ruby у межах парадигм подійно та об’єктно орієнтованого програмування.

***Ключові слова***: подійно та об’єктно орієнтоване програмування, ruby2D .

У публікації [1] проведено аналіз проблем, пов’язаних зі зміною парадигми вивчення інформатики у школі: істотним зростанням кількості навчальних годин на вивчення програмування. Серед інших зазначено такі.

Намагання авторів підручників використовувати лише навчальні середовища програмування (наприклад, Lazarus), що істотно знижує мотивацію

до навчання.

Відсутність україно- і російськомовних джерел щодо важливих і важких для сприйняття аспектів вивчення актуальних для індустрії програмного забезпечення мов і середовищ програмування.

Відсутність для багатьох питань щодо програмних засобів, задіяних в індустрії програмного забезпечення, стислого викладу, прийнятного для навчального процесу, навіть у англомовних джерелах.

Майже всі джерела щодо актуальних в індустрії програмного забезпечення мов і середовищ програмування розраховані на програмістів з досвідом або хоча б на старшокурсників. Копіювання їх (лише з перекладом) у навчальних матеріалах для учнів непродуктивне. Вони вимагають тривалої і кропіткої роботи щодо пристосування їх до рівня школярів. У першу чергу щодо логіки і стислості викладу змісту, зрозумілості учнями;

Як зауважено було ще на семінарі методистів ОІППО 21.06.2018 у Києві, згідно з чинною програмою учні мають навчитися порівнювати роботу в різних середовищах програмування, відповідне зауваження (нагадування) вирішили у 2018 року не включати до методичних рекомендацій МОН, але зробити це до наступного навчального року. Причина така: у 2018/2019 навчальному році не буде 11-класників, яких навчатимуть за новою програмою, тому вимагати досягнення цієї компетенції вже цього року некоректно.

Однією з мов програмування, сфера застосування якої в індустрії програмного забезпечення постійно розширюється, є мова Ruby. І щодо неї перелічені проблеми видаються найгострішими.

У даній роботі описано основи використання багатоплатформної бібліотеки ruby2D [2]. За допомогою цієї бібліотеки можна створювати 2D-програми мовою ruby.

Незнання основ мови ruby не є перешкодою для розуміння змісту статті — настільки є простою структура поданих прикладів програм (крім останнього прикладу). Єдине застереження для початківців: відступ ліворуч у мові ruby є вказівкою для інтерпретатора: або про продовження запису виразу, або про вкладення. Наприклад, тіла циклу в записі циклу.

**Встановлення бібліотеки ruby2d:**

- для **ОС MacOS** рекомендують використовувати `Homebrew` і виконати:

```
brew tap simple2d/tap
brew install simple2d
```

- для **ОС Windows** потрібно виконати вказівки , викладені у [3];
- для **ОС Linux** виконують такі вказівки *Терміналу* (з встановленням інтерпретатора ruby, бібліотеки gtk3 і власне ruby2d*):*

```
sudo apt-get install ruby
sudo apt-get install ruby-dev
sudo apt-get install rake
sudo gem install rake

sudo apt-get install libtcltk-ruby
sudo gem install gtk3
sudo apt install -y libsdl2-dev \
  libsdl2-image-dev libsdl2-mixer-dev \
  libsdl2-ttf-dev
sudo url='https://
  raw.githubusercontent.com/simple2d/
  simple2d/master/bin/simple2d.sh';
  which curl > /dev/null &&
  cmd='curl -fsSL' || cmd='wget -qO -
';
  bash <($cmd $url) install
sudo gem install ruby2d
```

Передостанню вказівку `sudo url=…` з наступними 5 рядками потрібно набрати одним рядком у *Терміналі.*

Для наочності ознайомлення з подальшим матеріалом бажано встановити програмне забезпечення і перевіряти роботу прикладів, запускаючи програму на виконання такою вказівкою Терміналу (командного рядка):

```
ruby назва_програми.rb
```

Ruby2D має такі об'єкти: вікно, колір об'єкта, фігура, зображення, текст, звук і передбачає опрацювання подій.

Створення вікна можна здійснити, замовивши Ruby2D і викликавши метод show:

```
require 'ruby2d'
show
```

У результаті — порожнє вікно з чорним тлом розміром 640×480 пікселів та рядком заголовку з текстом: «Ruby 2D».

**Налаштування властивостей** здійснюють таким чином:

```
set властивість: надане_значення
```

Наприклад,

```
set title: "Нова назва", background:
'blue'
```

**Властивості (атрибути) вікна** (назва — зміст — значення як усталено):

*title* — заголовок вікна — "Ruby 2D";
*background* — колір тла — "black" (чорний);
*width* — ширина вікна у пікселях — 640;
*height* — висота вікна у пікселях — 480;
*viewport_width* — ширина видимої частина вікна у пікселях;
*viewport_height* — висота видимої частина вікна у пікселях;
*resizable* — визначає, чи можна змінити розмір вікна — false;
*borderless* — визначає, чи вікно має межу — false;
*fullscreen* — чи буде відображено вікно у повноекранному режимі — false;
*diagnostics* — чи буде виведено на консолі дані відлагодження — false.

**Метод get** використовують для отримання значення властивості (атрибута) вікна — див. приклад для ширини вікна.

```
get :width
```

На відміну від set, метод get дає можливість отримати лише одне значення.

**Додаткові властивості (атрибути),** що описують поточний стан:

*window* — вікно об'єкта;
*frames* — кількість кадрів, відтворених моменту створення вікна;
*fps* — поточна частота кадрів (на секунду);
*mouse_x* — *x*-координати вказівника миші відносно

вікна;

*mouse_y* — *y*-координати вказівника миші відносно вікна.

**Цикл оновлення** оживляє вікно. Зазвичай 60 разів на секунду або як можна ближче до цього, наскільки це дозволяє продуктивність комп'ютера. Проілюструємо це на прикладі зміни кольору тла вікна випадковим чином щосекунди.

```
  require 'ruby2d'
tick = 0

update do
  if tick % 60 == 0
    set background: 'random'
  end
  tick += 1
end

show
```

**Закриття вікна** легко здійснити, натиснувши кнопку *Закрити* на панелі заголовка, за допомогою клавіатури (з *Command-Q* на Mac або *ALT + F4* у Windows) або за допомогою панелі меню. Але всі ці дії використовують *інтерфейс користувача.* Для закриття вікна *з коду* використовують метод *close* після виклику show. Зауважте: показ вікна означає введення нескінченного циклу вікна, тому наступний код не буде досягнуто, доки вікно не буде закрито вікно, тобто не завершиться цикл:

```
require 'ruby2d'
   show
   puts('Вам сюди не дістатися!')
   close
```

Розглянемо приклад програми мовою Ruby, що закриває вікно через 5 секунд після того, як його буде показано і поки цикл працює, використовує метод оновлення, описаний вище.

```
require 'ruby2d'
t = Time.now
update do
  if Time.now - t > 5 then close end
end
show
```

**Color** — *це клас, що встановлює колір в Ruby2D.*

**Ініціалізація кольору за ключовим словом** має такий вигляд:

```
Color.new('назва_кольору')
```

Тут *назва_кольору* може мати одне з таких значень: aqua, black, blue, brown, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white, yellow, random. В останньому випадку буде використано випадково породжений колір.

**Ініціалізація кольору з допомогою шістнадцяткового коду кольору моделі RGB** має такий вигляд (подано приклади й відповідні значення):

```
Color.new('#001F3F') #=> navy
Color.new('#0074D9') #=> blue
Color.new('#7FDBFF') #=> aqua
```

**Ініціалізація за допомогою масиву значень інтенсивності кольорів** червоного, зеленого, синього та альфа-каналу (непрозорості), значення яких подано десятковим дробом з проміжку [0; 1].

Наприклад, таким чином:

```
Color.new([1, 0, 0, 1]) #червоний непрозорий
Color.new([1.0, 0, 0, 1.0])      # те саме
Color.new([1, 0, 0, 0])  # червоний прозорий
Color.new([0.1, 0.0, 0.0, 0.5])  # червоний
                                 # напівпрозорий
```

**Методи r, g, b** використовують, щоб повернути інтенсивності відповідно червоного, зеленого й синього каналів як значення від 0 до 1:

```
color = Color.new('fuchsia')
color.r #=> 0.9411764705882353
color.g #=> 0.0705882352941176
color.b #=> 0.7450980392156863
```

**Методи a, opacity** використовують, щоб повернути інтенсивність альфа-каналу як значення від 0 до 1 див. приклад:

```
color = Color.new([0.1, 0.2, 0.3, 0.4])
color.a       #=> 0.4
color.opacity #=> 0.4
```

**Метод opacity=** можна використати для регулювання прозорості кольору:

```
color = Color.new([0.1, 0.2, 0.3, 0.4])
color.opacity = 0.9
```

Перейдемо до роботи з фігурами примітивної графіки.

**Квадрат** (клас *Square*) створюють таким чином:

```
s=Square.new
(x:a,y:b,z:c,size:d,color:"колір")
```

Тут:

- *a*, *b* — координати (у пікселях) верхнього лівого кута зображення, значення 0 як усталено;
- *c* — *z*-індекс (координата по вертикалі, номер шару) — об'єкти з більшим значенням *z* розташовані поверх об'єктів з меншим значенням *z*, значення 0 як усталено;
- *d* — довжина сторони (у пікселях), значення 100 як усталено;
- *колір* — як усталено білий ("white"), переданий екземпляру класу *Color*, тому можна при заданні використати функції *Color*.

Можна отримувати або змінювати значення властивостей *x, y, size, color:*

```
s = Square.new
s.x #=> 0
s.y #=> 0
s.size #=> 100
s.color # => Color.new('white')


s.x = 100
s.y = 100
s.size =200
s.color = 'red'
```

s.color повертає екземпляр класу *Color* або, якщо при створені квадрату використано декілька кольорів, екземпляр класу *Color::Set*, що містить масив цих кольорів та функції від них. Наприклад, зображення квадрату з кольорами вершин зеленим, синім, червоним і жовтим та м'яким переходом кольору можна отримати такою програмою:

```
require 'ruby2d'
s = Square.new(size:475, color:
    ['green', 'blue', 'red', 'yellow'])
show
```

**Прямокутник** (клас *Rectangle*, батьківський клас щодо *Square*) описують майже так само, як квадрат (Square), але з однією істотною відмінністю: замість однієї властивості *size* (розмір) використовують *width*

і *height* (розміри по горизонталі й вертикалі) — див. приклад.

```
r = Rectangle.new(x:0, y:0, width:200,
    height:100, z:0, color:'white')
```

*Тут і далі у прикладах опису створюваних об'єктів значення усіх властивостей подано як усталено.* Тобто вказано ті значення, які буде надано властивостям, якщо їм не надано значень (не згадано) у коді програми. Будь-яку з вказаних властивостей можна отримати традиційним способом:

*назва_об'єкта.назва_властивості*

і змінити її значення

*назва_об'єкта.назва_властивості = нове_значення*

саме так, як для квадрата — приклади для класу Square подано вище.

**Чотирикутник** (клас *Quad*, батьківський клас щодо *Rectangle*) описують, використовуючи опис координат вершин у фігурних дужках — див. приклад.

```
q = Quad.new(x1:  0, y1:  0,
             x2:100, y2:  0,
             x3:100, y3:100,
             x4:  0, y4:100,
             z:0, color:'white')
```

**Трикутник** (клас *Triangle*) описують, використовуючи опис координат вершин у фігурних дужках — див. приклад:

```
t = Triangle.new(x1: 50, y1:  0,
                 x2:100, y2:100,
                 x3:  0, y3:100,
                 z:0, color:'white')
```

Зображення трикутника з кольорами вершин червоним, зеленим, синім та м'яким переходом кольору можна отримати таким кодом:

```
t = Triangle.new(color:
    ['red', 'green', 'blue'])
```

**Відрізок прямої** (клас *Line*) описують, використовуючи опис координат кінців відрізка у фігурних дужках — див. приклад.

```
l = Line.new(x1:0, y1:0, x2:100, y2:100,
             width:2,z:0,color:'white')
```

Тут використано властивість width — товщина лінії (у пікселях).

**Завантаження зображення у вікно** має такий вигляд:

```
img = Image.new(x:a, y:b, z:c,
      width:w, height:h,
      path:"шлях/до/файлу",
      color:"колір тла")
```

Тут:

- *a*, *b* — координати верхнього лівого кута зображення (у пікселях), нульові значення як усталено;
- *c* — *z*-індекс (координата по вертикалі, номер шару) — об'єкти з більшим значенням *z* розташовані поверх об'єктів з меншим значенням *z*, нульове значення як усталено;
- *w*, *h* — ширина й висота зображення (у пікселях), розміри зображення як усталено.

Порядок переліку значень властивостей зображення довільний. Лише вказання шляху є обов'язковим. Значення як усталено color:"white".

Властивість path не можна змінювати при роботі із зображенням. Якщо за вказаною адресою немає файлу, буде виведено таке повідомлення:

```
Cannot find image file "шлях/до/файлу"
```

**Приклади застосування**

```
img = Image.new(path: "img/01.jpg")
img.x = 10
img.y = 10
img.x #=> 10
img.y #=> 10
img.color  = "red"
img.color  #=> Color.new("red")
img.color  = [0.8, 1.0, 0.5, 1.0]
img.width  = 125
img.height = 125
puts img.path
```

**Визначення належності точки до об'єкту** здійснюють з допомогою методу *contains?*, що має 2 аргументи — координати точки. Наприклад, програма:

```
require 'ruby2d'
square = Square.new
puts(square.contains?(10, 10))
puts(square.contains?(110, 110))
show
```

перед створенням білого квадрата на чорному тлі виводить такі значення:

```
true
false
```

**Виведення тексту у вікні** має такий вигляд:

```
t = Text.new(x:a, y:b, z:c,
    text: "Text for output", size: 20,
    font: "шлях/до/файлу/зі/шрифтом",
    color: "колір")
```

Тут:

- *a*, *b* — координати (у пікселях) верхнього лівого кута виведеного тексту, нульові значення як усталено;
- *c* — *z*-індекс (координата по вертикалі, номер шару) — об'єкти з більшим значенням *z* розташовані поверх об'єктів з меншим значенням *z*, нульове значення як усталено;
- *колір* — як усталено білий ("white"), переданий екземпляру класу *Color*, тому можна при заданні використати функції *Color*.

Як усталено, властивість text має таке значення: "Hello World!", властивість size (розмір шрифту) — 20. Серед вказаних вище незмінюваними є лише властивості *size, path*.

Властивості *width* і *height* (ширина і висота) залежать від тексту, шрифту та розміру. Ці значення цих властивостей використовують при програмуванні меню чи кнопок.

**Приклад**

```
t = Text.new(x:0, y:0, z:0,
    text: 'Text for output', size: 20,
    font: '/usr/share/fonts/truetype/'+
    'ubuntu-font-family/Ubuntu-R.ttf',
    color: 'red')
puts(t.width)
puts(t.height)
t.color = 'fuchsia'
```

**Клас Sound** — *це клас, призначений для представлення нетривалого звуку, як, наприклад, вирізання дерева або удару мечем.* Його використовують, наприклад, таким чином:

```
s = Sound.new("шлях/до/файлу")
s.play
```

У разі відсутності файлу (у цьому чи наступному прикладі) виникає помилка виконання.

**Клас Music** — *це клас, призначений для представлення тривалого звукового супроводу*. Його використовують, наприклад, таким чином:

```
m = Music.new("шлях/до/файлу")
```

Є кілька методів, які можна застосувати до представників цього класу:

- play — відтворити;
- pause — пауза;
- resume — відновити відтворення;
- stop — припинити відтворення;
- fadeout — припинити відтворення з пониженням гучності протягом вказаної кількості мілісекунд

— див. приклади використання:

```
m = Music.new("шлях/до/файлу")
m.play  # грати
m.pause # пауза
m.resume#відновити відтворення з місця паузи
m.stop  #припинити відтворення з переведення
                # вказівника на початок файлу
m.loop = true # повторювати відтворення
                # після досягнення кінця файлу
m.play        # грати
m.fadeout(2000) # припинити відтворення з
      # пониженням гучності протягом 2 секунд
```

Ruby 2D забезпечує спосіб захоплення та реагування на події, породжені за допомогою миші, клавіатури або контролера гри.

**Натискання клавіші миш** породжує подію *mouse_down*.

**Відпускання клавіші миші** породжує подію *mouse_up* — див. приклад.

```
require 'ruby2d'
on :mouse_down do |e|
  puts "Перехоплено натискання клавіші!"
end
on :mouse_up do |e|
  puts "Перехоплено відпускання клавіші!"
end
show
```

Тут об'єкт e — об'єкт класу *MouseEvent* з такими властивостями (значення вказано після тире):

- *e.type* — *:down* або *:up* для подій *mouse_down* або *mouse_up* відповідно;
- *e.button* — *:left* , *:right* або *:middle* залежно від того, якою кнопкою створено подію;
- *e.x, e.y* — координати вказівника миші.

**Рух миші** породжує подію *mouse_move*. Просте використання цієї події має такий вигляд:

```
require 'ruby2d'
on :mouse_move do |e|
  puts "Перехоплено рух миші!"
end
show
```

Тут об'єкт e — об'єкт класу *MouseEvent* з такими властивостями (значення вказано після тире):

- *e.type* — *:move;*
- *e.x, e.y* — координати вказівника миші після переміщення;
- *e.delta_x*, *e.delta_y* — приріст координат вказівника миші при переміщенні.

**Прокручування коліщатка** породжує подію *mouse_scroll*, яку можна використати, наприклад, таким чином:

```
require 'ruby2d'
on :mouse_scroll do |e|
  puts"Перехоплено прокручування коліщатка!"
end
show
```

Об'єкт e класу MouseEvent цієї події матиме такі властивості (значення вказано після тире):

- *e.type* — *:scroll;*
- *e.direction* — *:normal;*
- *delta_x* — 0
- *delta_y* — 1 або -1 залежно від напрямку прокручування.

**Події, породжені діями з клавіатурою**:

- *:key_down* — натискання клавіші;
- *:key_held* — утримування клавіші;
- *:key_up* — відпускання клавіші

— див. приклад.

```
require 'ruby2d'

on :key_down do |e|
puts "#{e.key} було натиснуто!"
end

on :key_held do |e|
puts "#{e.key} утримано!"
end

on :key_up do |e|
puts "#{e.key} відпущено!"
end

show
```

У цьому випадку *e* — об'єкт *KeyEvent* — матиме лише такі дві властивості (значення вказано після тире):

- *e.type* — *:down, :hold* або *:up* залежно від типу події, що трапилася;
- *e.key* — назва натисненої клавіші.

**Налаштування кількох обробників подій** проілюструємо таким прикладом:

```
require 'ruby2d'
```

```
on :mouse_down do
  puts "Перше повідомлення!"
end

on :mouse_down do
  puts "Друге повідомлення!"
end

on :mouse_down do
  puts "Третє повідомлення!"
end

show
```

Щоразу, коли буде натиснута кнопка миші, повідомлення надходитимуть у тому порядку, в якому їх описано. При цьому кількість обробників подій не обмежено.

**Вилучення подій** здійснюють тоді, коли використовувати ця подію вже не потрібно. Визначення *on* обробника події повертає об'єкт класу *EventDescriptor*. Щоб видалити подію, потрібно передати цей опис методу *off* — див. приклад:

```
require 'ruby2d'

event_descriptor = on :mouse_down do
  puts "Трапилася подія mouse_down!"
end

off(event_descriptor)     # виконати
 # незакоментованим і закоментованим
show
```

**Динамічне налаштування обробників подій зсередини методів і класів** за допомогою розширення класу доменно-специфічною мовою DSL (англійською domain-specific language) полягає у використанні *extend Ruby2D::DSL* у методі *initialize* класу. У поданому далі прикладі кожне клацання лівою кнопкою миші призведе до створення квадрату зеленого кольору. При наведенні на такий квадрат вказівника миші квадрат змінює колір із зеленого на червоний і зникає при натисканні правої кнопки миші.

```
require 'ruby2d'
interactive_squares = []
class InteractiveSquare
  def initialize(x, y)
    extend Ruby2D::DSL
    @square = Square.new(x: x, y: y,
              color: "green")
    @hover_event = on :mouse_move do |e|
      if @square.contains?(e.x, e.y)
        @square.color = "red"
      else
        @square.color = "green"
      end
    end
    @remove_event = on :mouse_up do |e|
      if e.button == :right
        if @square.contains?(e.x, e.y)
          self.remove
        end
      end
    end
  end
  def remove
    @square.remove
    off(@hover_event)
    off(@remove_event)
  end
end
on :mouse_down do |e|
  if e.button == :left
    interactive_squares <<
      InteractiveSquare.new(e.x, e.y)
  end
end

show
```

**Література**

1.Рудик О. Налаштування взаємодії wxWidgets і CodeBlocks для забезпечення осучаснення навчання інформатики. — Комп'ютер у школі та сім'ї. — № 3 (147), 2018. — с. 16–20.

2.Ruby 2D. Створюй багатоплатформний 2D застосунок мовою Ruby. — Режим доступу: http://www.ruby2d.com/

3.Ruby 2D при Windows. — Режим доступу: http://www.ruby2d.com/learn/windows/

**References. Translation and transliteration**

1.Rudyk A. Setting up wxWidgets and CodeBlocks interactions for providing the studying of informatics training. — Computer at School and Family. — № 3 (147), 2018. — c. 16–20

2.Ruby 2D. Make cross-platform 2D applications in Ruby. — Access code: http://www.ruby2d.com/

3.Ruby 2D on Windows. — Access code: http://www.ruby2d.com/learn/windows/

## RUBY 2D — КРОССПЛАТФОРМНАЯ БИБЛИОТЕКА ДЛЯ СОЗДАНИЯ 2D-ПРОГРАМ НА ЯЗЫКЕ RUBY

***Рудик Александр Борисович***

*кандидат физико-математических наук, доцент Института последипломного педагогического образования Киевского университета имени Бориса Гринченко, rudykob@gmail.com*

***Аннотация***. Статья освещает принципы работы с ruby2D — кросс-платформной библиотекой для создания 2D-программ на языке ruby в пределах парадигмы событийно и объектно ориентированного программирования.

***Ключевые слова:*** событийно и объектно ориентированное программирование, ruby2D.

**RUBY 2D - CROSS-PLATFORM LIBRARY FOR CREATING 2D APPLICATIONS IN RUBY CODE**

***Rudyk Alexander Borisovich,***

*candidate of Sciences (Ph.D.), Associate Professor of the Institute of Postgraduate Pedagogical Education of the Boris Grinchenko University of Kyiv,*
*rudykob@gmail.com*

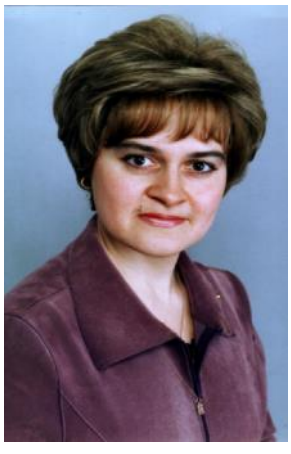
***Annotation***. The article highlights the main principles of working with a cross-platform library ruby2D for creating ruby 2D applications within the paradigm of event- and object-oriented programming. Enough simple examples of program code of two-dimensional color images are contained. Submitted in the article material is proposed as an important addition to the content of computer science learning as a practical example of modern programming.

***Keywords***: event oriented programming, object-oriented programming, ruby2D.

* * *

**УДК 37.016:911**

# ВИКОРИСТАННЯ НАВЧАЛЬНИХ ПРЕЗЕНТАЦІЙ НА УРОКАХ ГЕОГРАФІЇ ПРОФІЛЬНОГО РІВНЯ В ШКОЛІ



**Удовиченко Ірина Віталіївна,**
*кандидат педагогічних наук, доцент,*
*проректор з науково-педагогічної та методичної роботи*
*Сумського обласного інституту післядипломної педагогічної освіти ,*
*e-mail hafran@ukr.net ,*
ORCID ID 0000-0002-1980-5402

***Анотація*** У статті висвітлено питання актуальності використання елементів інформаційно-комунікаційних технологій, навчальних презентацій на уроках географії в старших класах загальноосвітньої, спеціалізованої школи. Відображено окремі аспекти створення презентацій з елементами анімацій; мультимедійних презентацій - на різних етапах уроків географії та вивчення навчального предмету на профільному рівні в школі.

***Ключові слова:*** навчальні презентації, мультимедійні презентації, інформаційно-комунікаційні технології, географія, старша школа, профільний рівень.

**Постановка проблеми.** Динамічні зміни в суспільстві викликають необхідність постійної роботи над собою кожної людини. Особливо актуальним є це питання для педагога як фахівця, фасилітатора змін, транслятора нових ідей, мобільного консультанта учнів в епоху інформаційно-цифрових технологій та миттєвого оновлення інформаційних даних.

**Аналіз останніх досліджень і публікацій**. Питання інформатизації освіти й навчального середовища, застосування комп'ютерних та інформаційно-комунікаційних технологій, формування інформаційної компетентності у своїх працях розглядали: В. Биков, Я. Ваграменко, І. Варфоломєєва, Б. Гершунський, Т. Голованова, М. Грузман, А. Гуржій, М. Жалдак, М. Згуровський, М. Камедія, Д. Корчевський, В. Лапінський, Т. Лисенко, Ю. Машбиц, О. Овчарук, О. Пометун, Й. Ривкінд, В. Тарадайник, В. Шакотько та багато інших.

Дидактичні аспекти формування інформаційної компетентності у процесі навчання географії за допомогою різних засобів навчання розглядали у своїх працях: М. Багров, Л. Вішнікіна, Н. Гончарова, В. Гудима, Л. Даценко, С. Кобернік, Р. Коваленко, В. Корнєєв, К. Костиря, О. Надтока, Т. Назаренко, В. Остроух, Л. Рибалко, В. Самойленко, О. Топузов, В. Яценко та інші.

**Формулювання мети статті.** Метою статті є висвітлення питання значимості використання елемен-