Ruby on Snake

1.Wprowadzenie i instalacja

Ruby jest skryptowym językiem programowania zorientowanym obiektowo. Aplikacje napisane za pomocą tego rewolucyjnego narzędzia są przenośne na wiele platform, co czyni z Rubiego konkurenta dla takich języków jak Java. Jednak Ruby propaguje zupełnie inna koncepcję przenośności, która nie wymaga kompilacji kodu do przenośnej postaci. Skrypty same w sobie są przenośne i interpretowane w ten sam sposób na różnych maszynach.

Instalacja nie jest zbytnio trudna, dlatego nie zajmie zbyt wielu akapitów tego poradnika. Pierwszą rzeczą, którą musimy zrobić jest instalacja środowiska Rubiego. W tym celu otwieramy terminal i wydajemy polecenie.

```
sudo apt-get install ruby
```

Następnie potrzebujemy czegoś co się nazywa RubyGems, jest to zarządca Gemów czyli paczek, które dostarczą nam dodatkowych narzędzi.

```
sudo apt-get install rubygems
```

Potrzebujemy teraz czegoś co wspomoże nasz proces tworzenia gry, mowa tutaj o silniku graficznym. Dostępnych jest kilka rodzai, ja preferuję RubyGame.

```
sudo gem install rubygame
```

Po instalacji powyższych paczek, jesteśmy w stanie przystąpić do faktycznego programowania aplikacji.

Zakładam iż czytelnik zna podstawy programowania w języku obiektowym i potrafi konstruować algorytmy.

2. Zasady gry

Zanim zajmiemy się implementacja, powinniśmy przypomnieć sobie zasady jakimi rządzi się owa gra. Tytułowy wąż jest naszą główna postacią, nad którą mamy sterowanie. Bohater przemieszcza się po planszy o stałym rozmiarze w czterech kierunkach, szukając jednocześnie jedzenia, które zwiększa jego rozmiar. Warunkiem końcowym gry jest wyjechanie poza planszę lub zjedzenie własnego ogona. Warto zapamiętać ten opis, gdyż będziemy do niego wracać.

3.Szkielet

Zacznijmy od napisania szkieletu naszej aplikacji.

```
require "rubygems"  #Dołączamy obsługę gemów
require "rubygame"  #Engine, na którym będziemy pracować
```

```ruby
#Dyrektywa podobna do „using namespace” z C++ uzyskujemy dostęp
#bezpośredni do metod i obiektów z modułu Rubygame
include Rubygame

class Game #Główna klasa naszej gry
  #Definiujemy kolor tła
  BG_COLOR = 'black'

  def initialize #Metoda wykona się przy tworzeniu obiektu klasy
      #Tworzymy powierzchnię, na której będziemy rysować
      #Używamy podwójnego buffora (DOUBLEBUF) w celu uniknięcie migotania obrazu
      @screen = Screen.new [640,480], 0, [HWSURFACE, DOUBLEBUF]

      #Tworzymy kolejkę zdarzeń
      @queue = EventQueue.new
      #Tworzymy zegar
      @clock = Clock.new
      #Ustawiamy preferowaną częstotliwość odświeżania
      @clock.target_framerate = 30
  end

  def draw_background #Użyjemy tej metody do narysowania tła
      #Wypełniamy wcześniej utworzoną powierzchnie kolorem czarnym
      @screen.fill BG_COLOR
  end

  def run #Pętla główna aplikacji
    loop do
      update #przeliczenie świata
      draw #metoda rysująca
      #Funkcja tick jak sama nazwa wskazuje wykonuje jedno tyknięcie zegara
      @clock.tick
    end
  end


  #Tutaj będziemy wykonywać wszystkie logiczne operacje, oraz obsługiwać
#zdarzenia
  def update
   @queue.each do |ev| #iterujemy każdy element z kolejki
      case ev
        when QuitEvent #sprawdzamy czy jest to żądanie zakończenia programu
          Rubygame.quit #zwalniamy rubygame
          exit #zamykamy progra
      end
    end
  end
  #Funkcja rysuje wszystkie elementy naszej sceny
  def draw
    draw_background #Rysujemy tło
    #Funkcja zamienia bufory, dzięki czemu widzimy zmianę na ekranie
    @screen.update
  end
end

#Tworzymy obiekt główny
game = Game.new
#Wywołujemy pętlę głowną
game.run
```

Powyższy szablon można użyć do napisania wielu różnorodnych gier, dlatego proponuję wam zapisać owy kod w łatwo dostępnym miejscu, najlepiej pod nazwą rubygame_temp.rb

4.Snake

W tym rozdziale skupimy się na naszej głównej postaci. Zastanówmy się przez chwilę, jakie cechy posiada wąż oraz jakie struktury powinniśmy użyć.

Czy wystarczy nam długość oraz pozycja? Odpowiedź brzmi nie, ponieważ musimy wiedzieć jak są rozmieszczone poszczególne odcinki ciała.
Potrzebujemy zatem struktury przechowującej takie elementy. Z pomocą przychodzi nam wbudowana w Ruby klasa Array. Jest to połączenie listy oraz stosu, posiada bardzo rozbudowane api. przyjazne w użyciu. Kolejnym atrybutem jest kierunek oraz kolor.

```
#Stała globalna, określa ilość pikseli per obiekt na ekranie, ponieważ nasza plansza będzie się
#składać z kwadratów
BOX_SIZE=10

class Snake
    COLOR = 'green' #definiujemy kolor skóry
    HEAD_COLOR = 'yellow' #osobny kolor dla głowy
    attr_accessor :parts #przydzielamy pełny dostęp do talbicy z zewnątrz klasy

    def length #Zwracamy długość
      @parts.size #długość jest równa ilości części ciała
    end

    #Aby utworzyć obiekt, musimy podać położenie (x,y) , długość oraz kierunek
    def initialize(x,y,length,direction)
         @direction = direction #zapisujemy wartość do zmiennej instancyjnej
         @parts = [] #Tworzymy pustą tablicę
         start = Rect.new(x,y,BOX_SIZE,BOX_SIZE)
         length.times do
           start = move_by_direction(start)
           @parts.unshift start
         end
    end

    def head #Zwracamy pierwszy element z tablicy, czyli głowę węża
      head = @parts.first
    end

   #Funkcja rysująca poszczególne partie na ekranie
    def draw on_surface
         #Wycinamy pierwszy element używając operatora [] oraz zakresu
         #Pierwszy element będzie rysowany innym kolorem
         @parts[1..-1].each do |part|
            #Dla każdego elementy wykonujemy funkcję draw_box_s z klasy Surface
            #Owa funkcja przyjmuje w parametrach lewy górny oraz prawy dolny róg
            #prostokąta, oraz kolor
            on_surface.draw_box_s part.topleft, part.bottomright, COLOR
         end
         #Rysujemy głowę innym kolorem, zdefiniowanym wyżej
         on_surface.draw_box_s head.topleft, head.bottomright, HEAD_COLOR

    end
end
```