Wrocław, 02.04.2019

UCISW 2 - PROJEKT

Gra Pong

Stanisław Drelich 235902
Jan Kamuda 218202

Spis treści

1. Wprowadzenie

1.1. Założenia projektowe

Celem projektu było zaimplementowanie gry zręcznościowej Pong z wykorzystaniem języka opisu sprzętu VHDL w układzie FPGA.

Zaprojektowana gra powinna dać możliwość przeprowadzenia rozgrywki dla dwóch osób, polegającej na odbijaniu piłki z wykorzystaniem klawiatury. Projekt zakłada również szereg właściwości takich jak zliczanie punktów, zmiana szybkości poruszania się piłki, zmiana rozmiaru i szybkości poruszania się paletki w zależności od postępów gracza, zmiana rozmiaru pola gry w zależności od ogólnej liczby zdobytych punktów, podświetlanie pola gracza tracącego punkt, podświetlanie pola zwycięzcy całej rozgrywki, ustawianie pułapki, resetowanie gry, pauzowanie.

1.2. Użyty sprzęt

W celu umożliwienia rozgrywki wykorzystano następujący sprzęt:

- Zestaw dydaktyczny Spartan 3E – Starter
- Klawiatura
- Monitor LCD

1.3. Moduły wewnętrzne i zaimplementowane

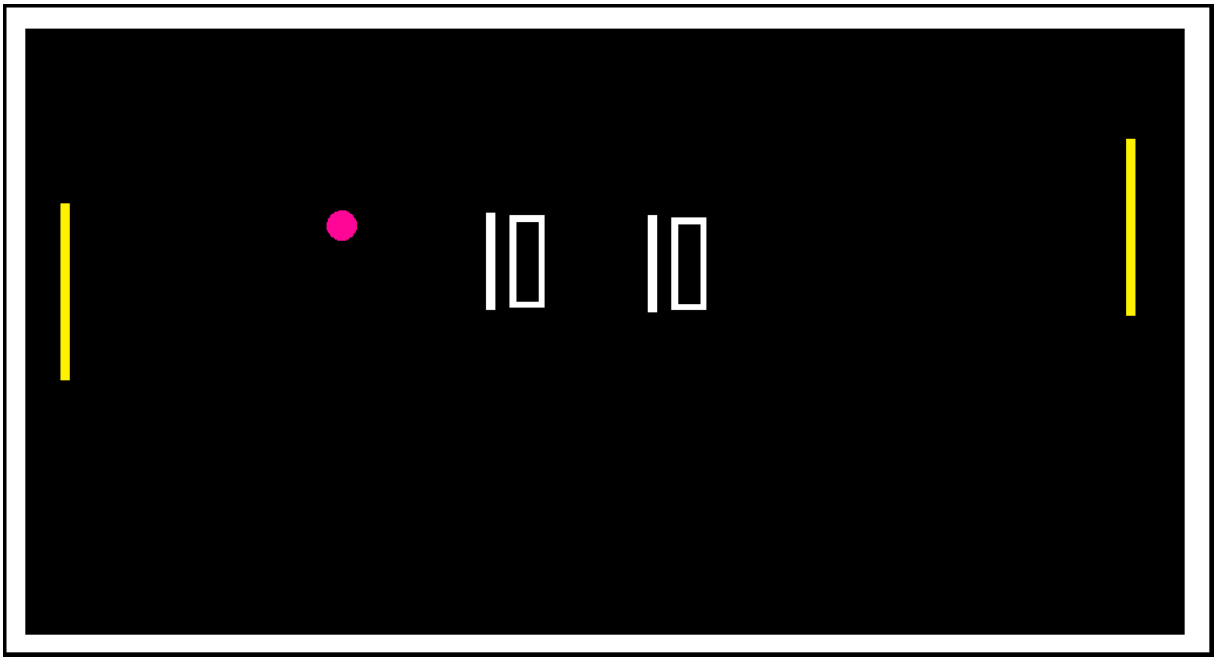
W celu umożliwienia rozgrywki wykorzystano następujące moduły:

- Moduły wewnętrzne (zaimplementowane przez Dra Sugiera)
    - PS2_Kbd
- Moduły zaimplementowane
    - Vga_driver
    - Grafika
    - Mechanika
    - Gracze
    - KbdDecode

2. Realizacja

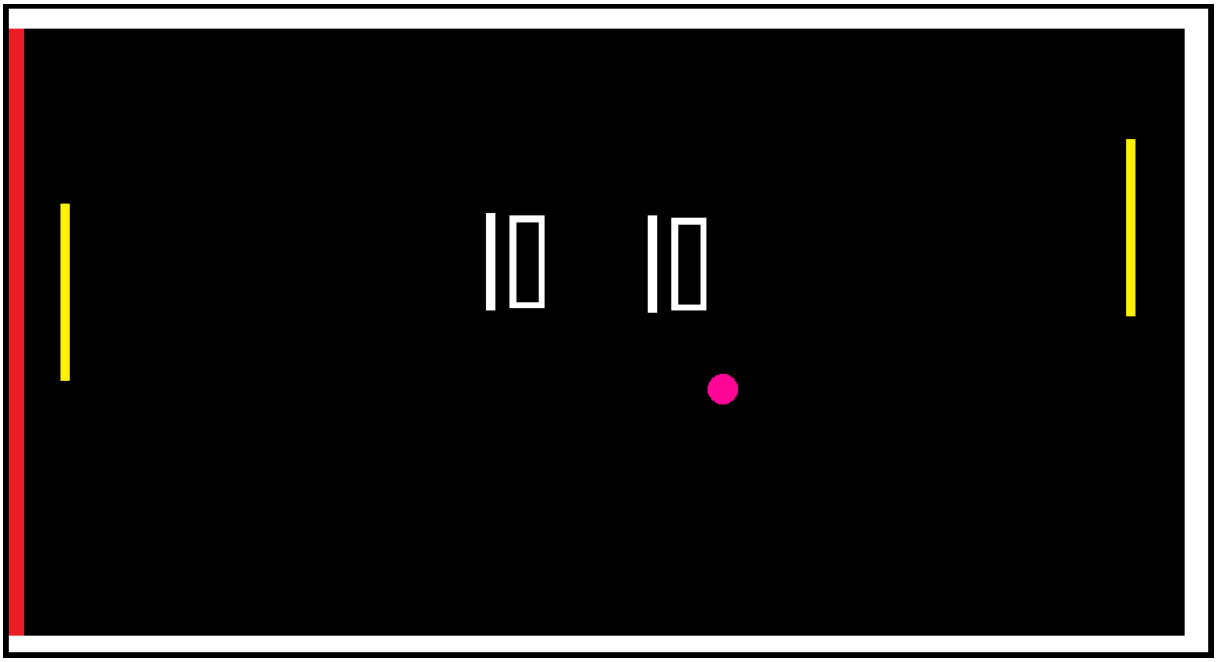
2.1 Ekrany gry

2.1.1 Widok planszy podczas regularnej rozgrywki



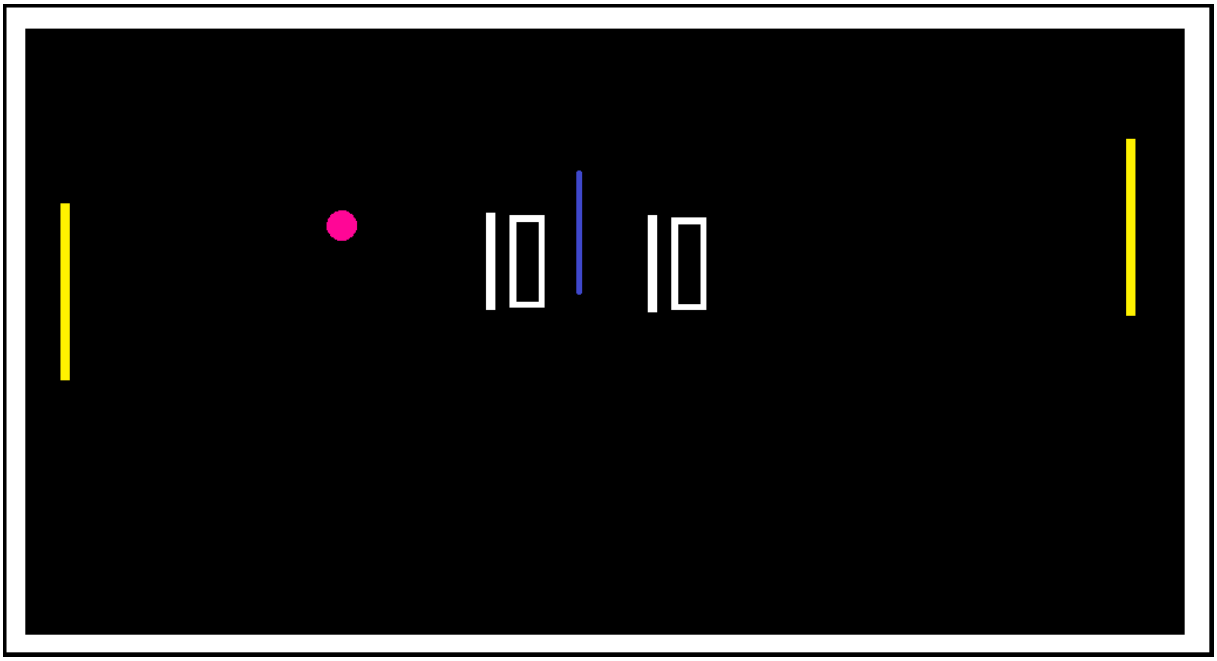
Rys. 1. Pole planszy podczas regularnej rozgrywki.

2.1.2 Widok planszy po zdobyciu punktu.



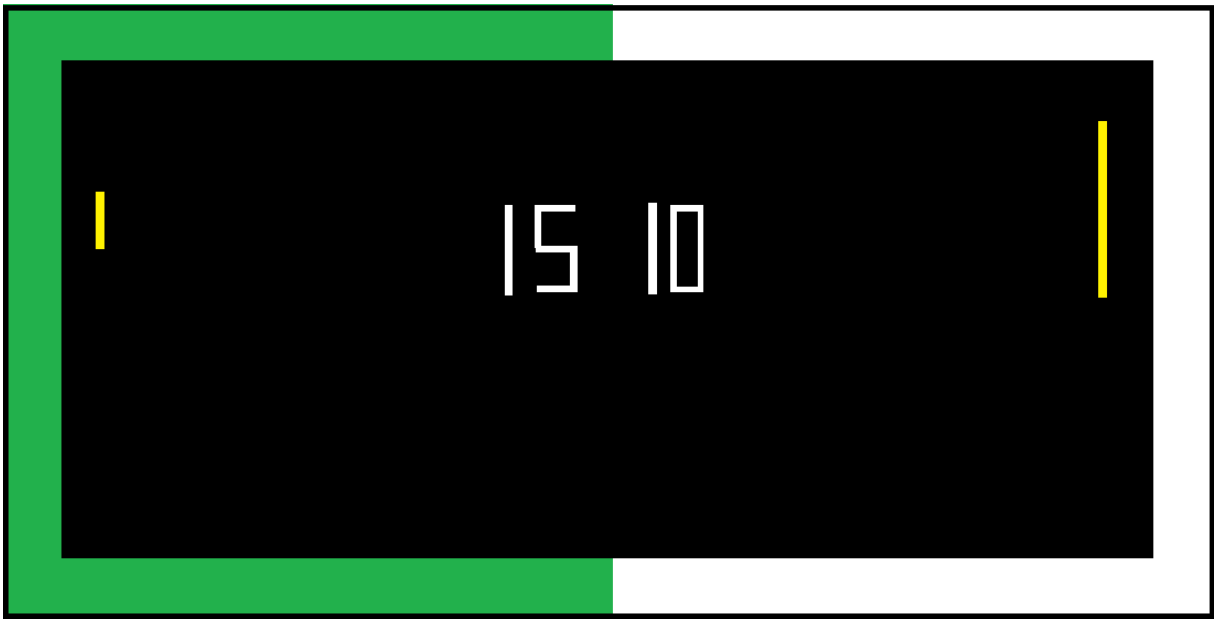
Rys. 2. Pole planszy po zdobyciu punktu (czerwone pole mruga).

2.1.3 Widok planszy podczas pojawienia się pułapki



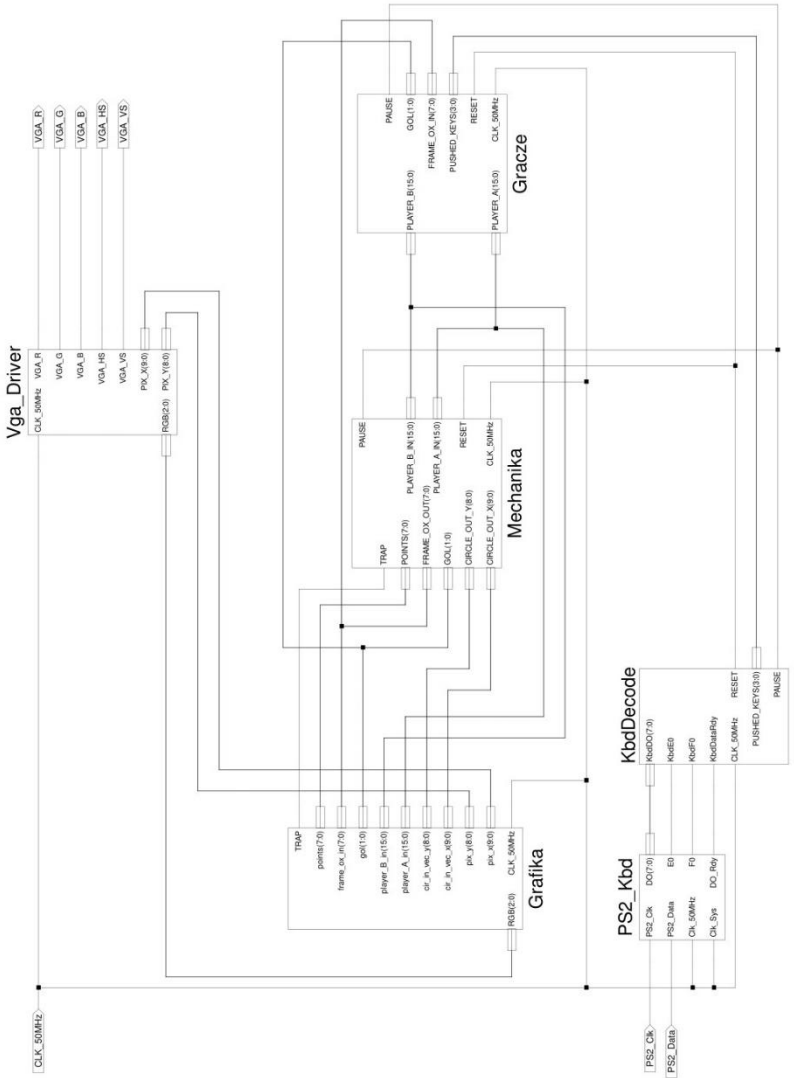
Rys. 3. Pole planszy podczas pojawienia się pułapki.

2.1.4 Widok planszy po wygranej.



Rys. 4. Pole planszy po wygranej (zielone pole mruga).

2.2 Schemat główny gry



2.3. Opis modułów

2.4.1 Grafika

Moduł odpowiedzialny za tworzenie i wyświetlanie elementów gry na ekranie tj. :

- tło, na które składa się pole gry oraz ramka podświetlająca się w zależności od zdobytych punktów.

- paletki graczy

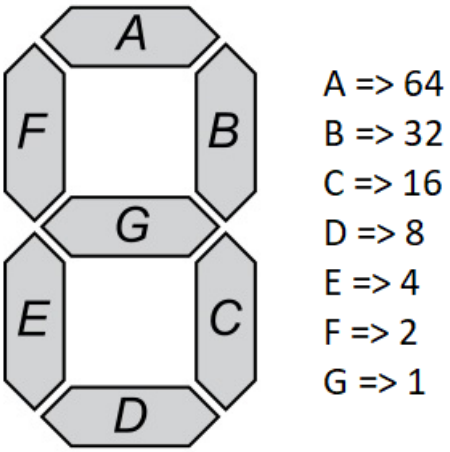
- piłka (realizacja opiera się na wykorzystaniu funkcji obliczającej pierwiastek kwadratowy)

- pułapka (w zależności od ustalonego prawdopodobieństwa

- punkty (wyświetlanie punktów zbliżone co do zasady działania wyświetlacza 7-segmetnowego)

(Poniżej schemat działania)

Przykład :



Jeżeli chcemy narysować liczbę, sumujemy wartości segmentów zgodnie z powyższym rysunkiem, następnie przekazujemy tę wartość do funkcji DrawSegment. Funkcja ta porównuje przyjętą wartość z kolejnymi potęgami liczby 2 zaczynając od 64 (2^liczba segmentów = 64), kończąc na 1, z każdym krokiem odejmując wartość w przypadku kiedy okazuje się ona być większa od porównywanego segmentu. Ten sposób pozwala na dokładne zdekodowanie i wyświetlenie właściwych segmentów. W celu rozjaśnienia działania tego sposobu, przedstawię go na przykładzie.

W celu wyświetlenia liczby „4” sumujemy wartości segmentów A, B i C, D i G, co daje nam wartość 51. Następnie przekazujemy tę wartość do funkcji DrawSegmnt. Funkcja zaczyna swoje działanie od porównania tej liczby z 64, która odpowiada za wyświetlanie segmentu A. Okazuje się ona większa i nie zostaje odjęta od 51 co skutkuje nienarysowaniem tego segmentu. Następnie funkcja przechodzi do kolejnej potęgi – 32, która okazuje się mniejsza od 51. W tym momencie a wartość zostaje odjęta od 51, wartość zostaje odjęta (51-32=19), a segment B narysowany. W kolejnych krokach schemat działania wykonuje się w ten sam sposób, co finalnie skutkuje wyświetleniem wszystkich segmentów odpowiedzialnych za narysowanie cyfry „4”.

```
--------------------------------------------------------------------------------
    function SingleDigit( pos_x, pos_y, number, pix_x, pix_y : SmallerInteger ) return STD_LOGIC is
        variable bool : STD_LOGIC;
    begin
        bool := '0';
           case number is
                   when 0 =>
                    if ( DrawSegment( pos_x, pos_y, pix_x, pix_y, 126 ) = '1' ) then
                          bool := '1';
                    end if;
              when 1 =>
                    if ( DrawSegment( pos_x, pos_y, pix_x, pix_y, 48 ) = '1' ) then
                          bool := '1';
                    end if;
              when 2 =>
                    if ( DrawSegment( pos_x, pos_y, pix_x, pix_y, 109 ) = '1' ) then
                          bool := '1';
                    end if;
              when 3 =>
                    if ( DrawSegment( pos_x, pos_y, pix_x, pix_y, 121 ) = '1' ) then
                          bool := '1';
                    end if;
              when 4 =>
                    if ( DrawSegment( pos_x, pos_y, pix_x, pix_y, 51 ) = '1' ) then
                          bool := '1';
                    end if;
              when 5 =>
                    if ( DrawSegment( pos_x, pos_y, pix_x, pix_y, 91 )= '1' ) then
                          bool := '1';
                    end if;
              when 6 =>
                    if ( DrawSegment( pos_x, pos_y, pix_x, pix_y, 95 )= '1' ) then
                          bool := '1';
                    end if;
              when 7 =>
                    if ( DrawSegment( pos_x, pos_y, pix_x, pix_y, 112 )= '1' ) then
                          bool := '1';
                    end if;
              when 8 =>
                    if ( DrawSegment( pos_x, pos_y, pix_x, pix_y, 127 )= '1' ) then
                          bool := '1';
                    end if;
              when 9 =>
                    if ( DrawSegment( pos_x, pos_y, pix_x, pix_y, 123 )= '1' ) then
                          bool := '1';
                    end if;
              when others =>
                    return '0';
           end case;
        if ( bool = '1') then
              return '1';
        else
              return '0';
        end if;
    end;
```

2.4.2 Mechanika

Najobszerniejszy moduł, odpowiedzialny za poruszanie się paletek i piłki. W celu jak najdokładniejszej realizacji obejmuje kilkanaście przypadków, w których może znaleźć się piłka.

Przykład:

```
process (clk_ball) -- poruszanie pilka
begin

    if (rising_edge( clk_ball )) then
        if ( pause = '0' ) then
                if ( ball_counter > 75000 ) then
                        ball_counter <= ball_counter - 50;
                elsif ( ball_counter > 50000 ) then
                 ball_counter <= ball_counter - 10;
            elsif ( ball_counter > 10000 ) then
                 ball_counter <= ball_counter - 1;
            end if;

                cir_x <= cir_x + direction_x;
                cir_y <= cir_y + direction_y;

             -- uderzenie gracza w sciane od srodka mapy
                if (
                cir_x - r + direction_x <= player_a_x + player_width AND
                cir_x - r + direction_x > player_a_x AND
                cir_y <= player_a_y + player_a_length AND
                cir_y >= player_a_y ) then
                        direction_x <= direction_x * (-1);
                        if ( cir_x - r < player_a_x + player_width AND
                        cir_x + r > player_a_x ) then
                                cir_x <= player_a_x + player_width + r + 1;
                        end if;
                        if ( probability < 150 ) then
                                trap <= '1';
                                trap_buffer <= '1';
                        end if;
                end if;

                if ( cir_x + r + direction_x >= player_b_x AND
                cir_x + r + direction_x < player_b_x + player_width AND
                cir_y <= player_b_y + player_b_length AND
                cir_y >= player_b_y ) then
                        direction_x <= direction_x * (-1);
                        if ( cir_x + r > player_b_x AND
                        cir_x - r < player_b_x + player_width ) then
                                cir_x <= player_b_x - r - 1;
                        end if;
                        if ( probability < 150 ) then
                                trap <= '1';
                                trap_buffer <= '1';
                        end if;
                end if;
```

2.4.3 Gracze

Moduł odpowiedzialny za parametry gry związane ze sterowaniem paletkami. Obejmuje czynniki tj. wymiar rakietki zmniejszany o 25% oraz prędkość poruszania się jej zmniejszaną o 50% po każdym golu.

Przykład :

```
-- ruszanie graczami
process (clk_pl_a)
begin

    if (rising_edge( clk_pl_a )) then
        if ( pause = '0' ) then
            if ( pushed_keys(3) = '1' AND pushed_keys(2) = '0' ) then
                if ( player_a_y - frame_ox > 1) then
                    player_a_y <= player_a_y - 1;
                else
                    player_a_y <= frame_ox;
                end if;
            end if;

            if ( pushed_keys(2) = '1' AND pushed_keys(3) = '0' ) then
                if ( vertical_max - frame_ox - (player_a_y + player_a_length) > 1) then
                    player_a_y <= player_a_y + 1;
                else
                    player_a_y <= vertical_max - frame_ox - player_a_length;
                end if;
            end if;
        end if;

        if ( player_a_y < frame_ox) then
            player_a_y <= frame_ox;
        end if;

        if ( player_a_y + player_a_length > vertical_max - frame_ox) then
            player_a_y <= vertical_max - frame_ox - player_a_length;
        end if;


        player_A <= std_logic_vector( to_unsigned( player_A_length, 7 ) )
        & std_logic_vector( to_unsigned( player_A_y, 9 ) );

    end if;


end process;
```

2.4.4 KbdDecode

Moduł odpowiedzialny za odczyt i reakcje na naciśnięcia przycisków biorących udział w grze tzn. kierunków góra i dół oraz klawiszy specjalnych tj. pauza oraz reset. Opiera się na odczycie i dekodowaniu kodów ASCII, a następnie na przekazanie ich do modułów odpowiadających za mechanikę.

Przykład :

```
process( KbdDataRdy, CLK_50MHz ) begin
    if ( rising_edge ( CLK_50MHz ) AND KbdDataRdy = '1' ) then
        case KbdF0 & KbdE0 & KbdDO is

-- wcisniecia klawiszy
        when "00" & X"1D" => -- "W"
            pushed_keys(3) <= '1';
        when "00" & X"1B" => -- "S"
            pushed_keys(2) <= '1';
        when "01" & X"75" => -- "Gora"
            pushed_keys(1) <= '1';
        when "01" & X"72" => -- "Dol"
            pushed_keys(0) <= '1';
        when "00" & X"2D" => -- "R"
            R_Pressed <= '1';
        when "00" & X"4D" => -- "P"
              pause_buffer <= not pause_buffer;

-- puszczenia klawiszy
        when "10" & X"1D" => -- "W"
            pushed_keys(3) <= '0';
        when "10" & X"1B" => -- "S"
            pushed_keys(2) <= '0';
        when "11" & X"75" => -- "Gora"
            pushed_keys(1) <= '0';
        when "11" & X"72" => -- "Dol"
            pushed_keys(0) <= '0';
        when "10" & X"2D" => -- "R"
            R_Pressed <= '0';
-- #
        when others =>
            NULL;
        end case
    end if;
end process;
```

3. Podsumowanie

Realizacja projektu przebiegła pomyślnie i objęła wszystkie zakładane elementy gry, zaczynając od generowania obrazu, poruszanie się paletek i piłki, kończąc na wygenerowaniu pułapki, zmiany trudności gry oraz zliczaniu punktów.

Jedynym istotnym problemem, który wystąpił podczas wykonania zadania były zatrzaski występujące podczas obsługi klawiatury. Wynikały one z błędnej obsługi zdarzeń generowanych

przez naciśnięcia klawiszy. Po refaktoryzacji kodu polegającej na dodaniu odczytów wciśnięcia i zwolnienia klawiszy (make code i break code) problem ustąpił.

W obecnym momencie gra realizuje wszystkie założenia, a testy nie wykazały żadnych nieprawidłowości.