9.3HD

COS10009

Introduction to programming

Thanh Minh

Student ID: 103809048

CODE EXPLANATION DOCUMENT

Swinburne University of Technology

---

1. **Program's Window and implement some features**

```
module ZOrder
  HIDDEN, BACKGROUND, SCENE, UI, TOP = *0..4
end
```

Figure 1: set up the ZOrder to arrange the layers for later uses in the program

Hidden will be at the back of the program when I want to hide content and create the effects so that the program can have a better looking.

I have created the constant HEIGHT(800) and WIDTH(1500) for the size of the window, and put it on the top of the file so that everyone can see clearly.

```
17  def initialize()
18   super(WIDTH,HEIGHT,false)
19   self.caption = "Spaceship Hunter"
20   @player = Player.new(self)
```

Figure 2: set up for the window

If I change false into true, the program's window will be played in full screen. I also create an array for Player for displaying ships in the Selecting page.

```
window = MyGame.new()
window.show()
```

Figure 3: End code of program

Without these 2 lines of code, the program cannot run so I must include it in every single Gosu program.

```ruby
def update
  case @scene
  when :start
    update_start
  when :instruction
    update_instruction
  when :selecting
    update_selecting
  when :playing
    update_playing
  when :end
    update_end
  end
end
```

Figure 4: States and procedures of Update for program

```ruby
def draw
  case @scene
    when :start
      draw_start
    when :instruction
      draw_instruction
    when :selecting
      draw_selecting
    when :playing
      draw_playing
    when :end
      draw_end
    end
  end
```

Figure 5: States and procedures of Draw for program

First I will have to set the @scene = :start so that the program will start at the entry page and will call all the procedures related to start such as draw_start, update_start, …

I must have each screen of the program, it should have different functions so that can increase the efficient of the code, make the program run smoother, faster. The code is also can be visualized easily.

2. Entry page

First, I need to set up in order to draw up the entry page for the program.

```
#--------------- start point --------------------
@scene = :start
@start_music = Gosu::Song.new('musics/start.wav')
@start_background = Gosu::Image.new('images/start.jpg')
@title = Gosu::Font.new(50)
@start_options = Gosu::Font.new(35)
@info_font = Gosu::Font.new(15)
@hover_options_1 = ZOrder::HIDDEN; @hover_options_2 = ZOrder::HIDDEN; @hover_options_3 = ZOrder::HIDDEN
@error_music = Gosu::Song.new('musics/error.wav')
@start_play_music = Gosu::Song.new('musics/start_play.wav')
@point = Gosu::Image.new('images/pointing.png')
```

Figure 6: Setup for the entry page

I need to set @scene = :start by default so that the program can start and draw the entry page first. I also nneed to import some sounds and musics into my program. I also define the font size for drawing title, description of the program so that the user can know what is on the screen.
For the @hover_options_1,2,3 , I will use it for creating effects while suffering through options so that the user can feel like the program is make with a lot of effort.
For two images that I have imported, start.jpg is for the background of the screen to decorate the screen. The other one is for pointing at the current option (there are some RPG using this function for their menu)

```
def draw_start
  @start_background.draw(0, 0, z = ZOrder::BACKGROUND)
  @start_music.play(false)
  @title.draw_text("Spaceship hunter", 650, 50, z = ZOrder::TOP, 1, 1, Gosu::Color::WHITE)
  @start_options.draw_text("Play \n\n\nInstructions \n\n\nExit
    ", 500, 300, z = ZOrder::TOP, 1, 1, Gosu::Color::WHITE)

  @point.draw(355,305,1, @hover_options_1)
  @point.draw(355,410,1, @hover_options_2)
  @point.draw(355,510,1, @hover_options_3)

  draw_quad(490,295, Gosu::Color::BLACK, 750, 295, Gosu::Color::BLACK, 750, 340, Gosu::Color::BLACK, 490, 340, Gosu::Color::BLACK, @hover_options_1)
  draw_quad(490,400, Gosu::Color::BLACK, 750, 400, Gosu::Color::BLACK, 750, 445, Gosu::Color::BLACK, 490, 445, Gosu::Color::BLACK, @hover_options_2)
  draw_quad(490,505, Gosu::Color::BLACK, 750, 505, Gosu::Color::BLACK, 750, 550, Gosu::Color::BLACK, 490, 550, Gosu::Color::BLACK, @hover_options_3)

end
```

Figure 7: draw_start procedure

After importing background image, I need to draw it so that it can display, but I will make the ZOrder of it is Background so that it won't hide other contents.

I have made the ZOrder of @point and @hover_options_1,2,3 are hidden so that @point and black rectangle won't display on the screen until their ZOrder are changed.

```
def update_start
  if (mouse_x > 489 && mouse_x < 756 && mouse_y > 294 && mouse_y < 341) or (button_down?(Gosu::KbUp) && @hover_options_2 == ZOrder::UI)
    @hover_options_1 = ZOrder::UI
    @hover_options_2 = ZOrder::HIDDEN
    @hover_options_3 = ZOrder::HIDDEN
  elsif (mouse_x > 489 && mouse_x < 756 && mouse_y > 399 && mouse_y < 446) or (button_down?(Gosu::KbUp) && @hover_options_3 == ZOrder::UI) or (button_down?
  (Gosu::KbDown) && @hover_options_1 == ZOrder::UI)
    @hover_options_1 = ZOrder::HIDDEN
    @hover_options_2 = ZOrder::UI
    @hover_options_3 = ZOrder::HIDDEN
    sleep(0.15)
  elsif (mouse_x > 489 && mouse_x < 756 && mouse_y > 504 && mouse_y < 556) or (button_down?(Gosu::KbDown) && @hover_options_2 == ZOrder::UI)
    @hover_options_1 = ZOrder::HIDDEN
    @hover_options_2 = ZOrder::HIDDEN
    @hover_options_3 = ZOrder::UI
  elsif (button_down?(Gosu::KbDown) && @hover_options_1 == ZOrder::HIDDEN && @hover_options_2 == ZOrder::HIDDEN && @hover_options_3 == ZOrder::HIDDEN)
    @hover_options_1 = ZOrder::UI
    @hover_options_2 = ZOrder::HIDDEN
    @hover_options_3 = ZOrder::HIDDEN
  elsif (button_down?(Gosu::KbUp) && @hover_options_1 == ZOrder::HIDDEN && @hover_options_2 == ZOrder::HIDDEN && @hover_options_3 == ZOrder::HIDDEN)
    @hover_options_1 = ZOrder::HIDDEN
    @hover_options_2 = ZOrder::HIDDEN
    @hover_options_3 = ZOrder::UI
  end

  if (button_down?(Gosu::KbUp) && @hover_options_1 == ZOrder::UI) or (button_down?(Gosu::KbDown) && @hover_options_3 == ZOrder::UI) or button_down?(Gosu::KbLeft) or
  button_down?(Gosu::KbRight)
    @error_music.play(true)
    sleep(0.05)
  end
end
```

Figure 8: update_start procedure

In figure 8, I have set up for the effects when suffering through options, the black box and the pointing rocket will be displayed at the current option like in figure 9

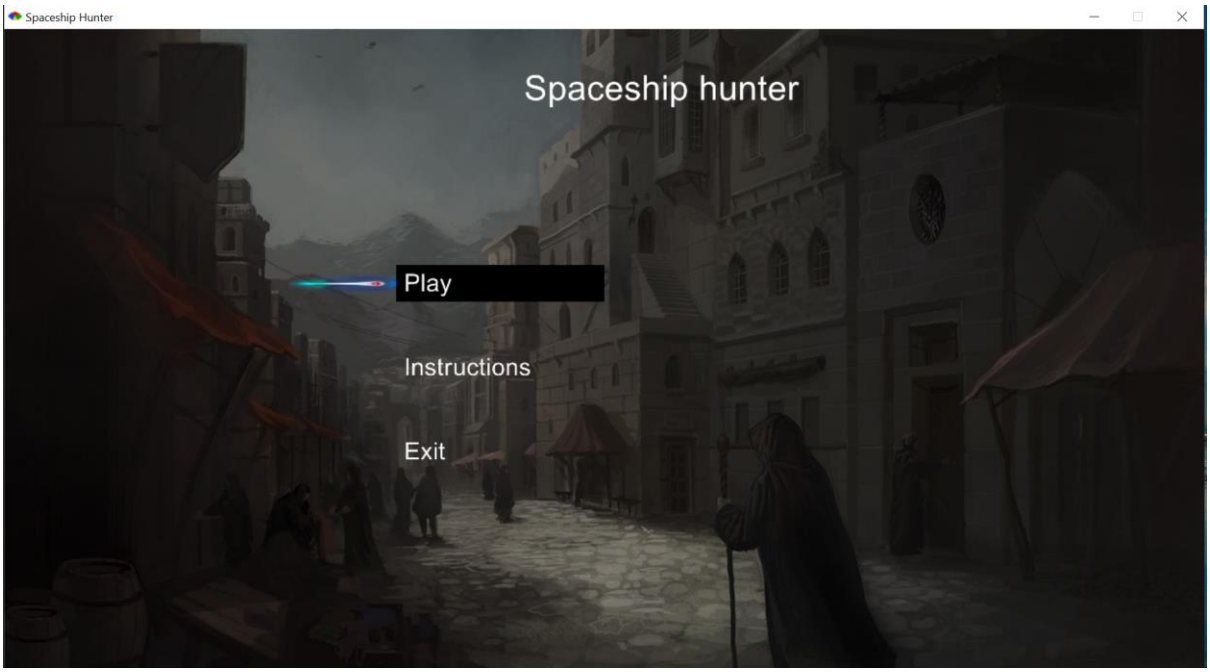

Figure 9: Entry page

```
def button_down_start(id)
  if (button_down?(Gosu::MsLeft) or button_down?(Gosu::KbReturn)) && @hover_options_1 == ZOrder::UI
    initialize_selecting
    @start_play_music.play(false)
    sleep(0.2)
  elsif (button_down?(Gosu::MsLeft) or button_down?(Gosu::KbReturn)) && @hover_options_2 ==
  ZOrder::UI
    initialize_instruction
  elsif (button_down?(Gosu::MsLeft) or button_down?(Gosu::KbReturn)) && @hover_options_3 ==
  ZOrder::UI
    close
  end

  end

end
```

Figure 10: button_down_start procedure

For this procedure, when I press the left mouse button or use Enter at the current option, it will direct me to the next page. When I choose the Play option, the start music will be played to announce me that I have entered the next stage of the game. And when I choose to close the program just choose the Exit option then the program will close immediately

3. Instruction page

```
#-------------- instruction -------------------------
@instruction_background = Gosu::Image.new('images/instruction.jpg')
@instruction_music = Gosu::Song.new('musics/instruction.wav')
@description = Gosu::Font.new(30)
```

Figure 11: set up for the instruction page

Before moving forward, I need to import music and background image for this page.

```
def draw_instruction
  @instruction_background.draw(0,0, ZOrder::BACKGROUND)
  @title.draw_text("          INSTRUCTION\n Created by Thanh Minh \n\n How to Play",
  500, 50, z = ZOrder::TOP, 1, 1, Gosu::Color::WHITE)
  @description.draw_text("Use keyboard to control:\n- ↑ or W to move up\n- ↓ or S to
  move down\n- → or D to move right\n- ← or A to move left\n- Space to
  Attack\n\n*REMEMBER: To avoid the enemy
    \nPress Q or Space to be back to the Menu",500,300, z = ZOrder::TOP, 1, 1,
    Gosu::Color::WHITE)
end
```

Figure 12: draw_instruction for the instruction page

For this procedure, mainly I draw the background image and the description for the user to read how to play this game. So that the user can understand more about the program.

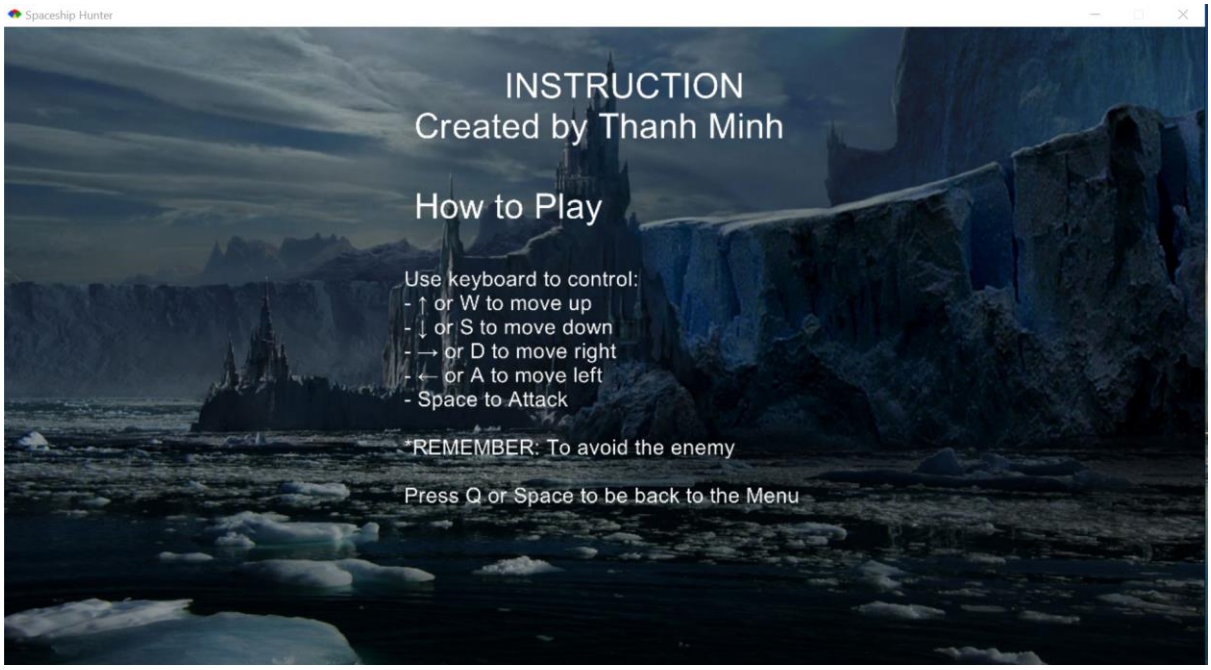

Figure 13: Instruction page

```
def update_instruction

end
```

Figure 14: Update_instruction for instruction page

This page is the easy page to be created among all the pages so in the update_instruction there is no code but if I have time to develop this project more, I will decorate more thing on this page and adding effects. Because there are some programs, games that in the instruction page, their page are so interactive and clearly show how the game work.

```
def button_down_instruction(id)
  if button_down?(Gosu::KbQ) or button_down?(Gosu::KbSpace)
    initialize_start
  end
end
```

Figure 15: button_down_instruction procedure for instruction page

When the user wants to exit and go back to the previous page, they can press Q or Space to go back.

4. **Selecting page**

At all the start of each page, I need to set up things for later use.

```
#--------------- selecting -------------------------
@selecting_music = Gosu::Song.new('musics/selecting.wav')
@select_char_1 = ZOrder::HIDDEN; @select_char_2 = ZOrder::HIDDEN
@select_color = 0xFF555555
@ship1_sel = Gosu::Image.new('images/ship1_select.png')
@ship2_sel = Gosu::Image.new('images/ship2_select.png')
```

Figure 16: Set up for Selecting page

After setting up, I also need to include the array for player for drawing it.

```
@player = Player.new(self)
```

Figure 17: Add array for player

After importing the media files, I need to draw out what I have set up above in the draw_selecting.

```
def draw_selecting
  @selecting_music.play(false)
  @title.draw_text("Selecting Your Player
    ", 550,50, z = ZOrder::TOP, 1, 1, Gosu::Color::WHITE)

    draw_quad(0,0, Gosu::Color::BLACK, WIDTH, 0, Gosu::Color::BLACK, WIDTH, HEIGHT,
    Gosu::Color::BLACK, 0, HEIGHT, Gosu::Color::BLACK, ZOrder::BACKGROUND)
    draw_quad(0,0, @select_color, WIDTH/2, 0, @select_color, WIDTH/2, 1500,
    @select_color, 0, 1500, @select_color, @select_char_1)
    draw_quad(WIDTH/2,0, @select_color, WIDTH, 0, @select_color, WIDTH, HEIGHT,
    @select_color, WIDTH/2, HEIGHT, @select_color, @select_char_2)

    @title.draw_text("Chaos Eater
      ", 250,300, z = ZOrder::TOP, 1, 1, Gosu::Color::WHITE)
    @title.draw_text("Ship Eater
      ", 1050,300, z = ZOrder::TOP, 1, 1, Gosu::Color::WHITE)
    @player.draw_select

  end
```

Figure 18: draw_selecting procedure for my program

For the first quadratic shape, I made its color Black and the ZOrder is Background and for the next two quadratic shapes I made it hidden so that when the user is choosing the player, the current option shape will be go up with the color of grey.

With the @player.draw_select, I will call the draw_select procedure for the array Player in another file so that the player can be drawn out.

```ruby
class Player
  def initialize(window)
    @x1 = 750/2
    @y1 = 650
    @x2 = 1175
    @y2 = 650
    @window = window
    @ship1 = Gosu::Image.new('images/ship1.png')
    @ship2 = Gosu::Image.new('images/ship2.png')

  end

  def draw_select
    @ship1.draw_rot(@x1 , @y1 , 1, 0)
    @ship2.draw_rot(@x2 , @y2, 1, 0)
  end
end
```

Figure 19: Array for Player

Above is the array of player I have created in another file to keep the main file shorter and clear.

```ruby
def update_selecting
  if mouse_x > 0 && (mouse_x < WIDTH/2) && mouse_y > 0 && mouse_y < 800
    @select_char_1 = ZOrder::BACKGROUND
    @select_char_2 = ZOrder::HIDDEN
  elsif (mouse_x > WIDTH/2) && mouse_x < WIDTH && mouse_y > 0 && mouse_y < 800
    @select_char_1 = ZOrder::HIDDEN
    @select_char_2 = ZOrder::BACKGROUND
  else
    @select_char_1 = ZOrder::HIDDEN
    @select_char_2 = ZOrder::HIDDEN
  end

end
```

Figure 20: update_selecting procedure

With those lines in the figure 20, the current option will be visualized for the user to see whenever they want to choose which player.

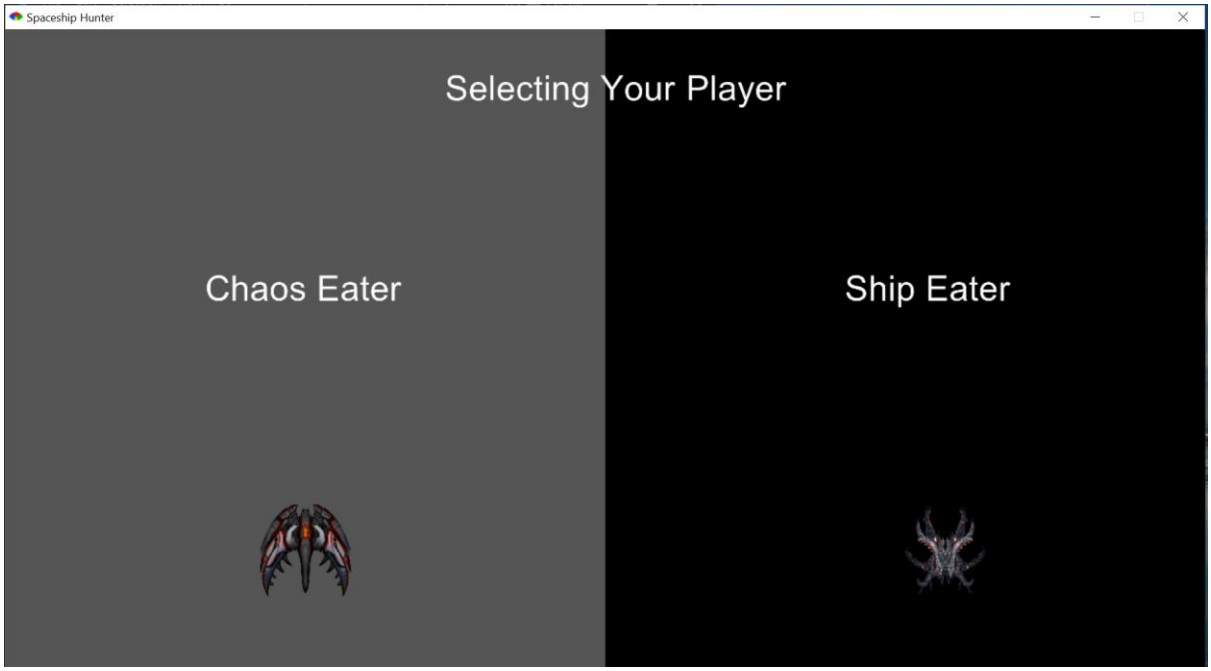

Figure 21: selecting page

The current option is chaos eater so the box on the left will be grey.

```
def button_down_selecting(id)
  if (button_down?(Gosu::KbReturn) or button_down?(Gosu::MsLeft)) && mouse_x >
  0 && mouse_x < WIDTH/2 && mouse_y > 0 && mouse_y < HEIGHT && @select_char_1
  == ZOrder::BACKGROUND
    initialize_playing
    @ship1_sel = Ship1.new(self)
    @ship = @ship1_sel
  elsif (button_down?(Gosu::KbReturn) or button_down?(Gosu::MsLeft)) &&
  mouse_x > WIDTH/2 && mouse_x < WIDTH && mouse_y > 0 && mouse_y < HEIGHT &&
  @select_char_1 == ZOrder::BACKGROUND
    initialize_playing
    @ship2_sel = Ship2.new(self)
    @ship = @ship2_sel
  elsif button_down?(Gosu::KbQ)
    initialize_start
  end
end
```

Figure 22: button_down_selecting procedure

With this procedure in figure 22, the program will know what the next step is when the user has chosen the player. The new array will be created depend on which ship the user choose and it will call exact what the user has chosen.

5. Play page

Before going further, I first need to set up and import the musics and text's font size. And I don't forget to include the files related to this file. The reason why I use the Sample for shooting music is the sound won't interrupt the music and create annoyed noise.

```
#----------------playing ---------------------------
@playing_music = Gosu::Song.new('musics/playing.wav')
@counting = Gosu::Font.new(45)
@shooting_music = Gosu::Sample.new('musics/shooting.wav')
@die_music = Gosu::Song.new('musics/die.wav')
```

Figure 23: initialize and import music

For the drawing parts, it is quite simple than others when I just need to draw out the essential things such as ship, bullets ,… more over I also want to count the scores of the players so I will use @count for counting the scores.

```
def draw_playing
  @ship.draw
  @enemies.each do |enemy|
    enemy.draw
  end
  @bullets.each do |bullet|
    bullet.draw
  end
  @explosions.each do |explosion|
    explosion.draw
  end
  @counting.draw_text("Score:",30,10,1)
  @counting.draw_text(@count,150,10,1)

end
```

Figure 24: draw_playing procedures

```ruby
def update_playing
  @ship.go_up if button_down?(Gosu::KbUp) || button_down?(Gosu::KbW)
  @ship.go_down if button_down?(Gosu::KbDown) || button_down?(Gosu::KbS)
  @ship.go_right if button_down?(Gosu::KbRight) || button_down?(Gosu::KbD)
  @ship.go_left if button_down?(Gosu::KbLeft) || button_down?(Gosu::KbA)
  #----------------enemy -------------------------
  if rand < FREQUENCY
    @enemies.push Enemy.new(self)
  end
  @enemies.each do |enemy|
    enemy.move
  end
  #---------------delete enemy -------------------
  @enemies.dup.each do |enemy|
    if enemy.y > HEIGHT + enemy.edge
      @enemies.delete enemy
    end
  end
  #-------------remove bullet no longer on screen-----------
  @bullets.dup.each do |bullet|
    @bullets.delete bullet unless bullet.onscreen?
  end
  @bullets.each do |bullet|
    bullet.move
  end
```

Figure 25: update_playing procedure part 1

For the figure 25, whenever I move the user, I will call the procedure from other files so that the player can be moved smoothly. I also generate the enemy and using rand < FREQUENCY which means the frequency of creating enemy will be randomly chosen and that value will be smaller than constant FREQUENCY. I also create procedure for automatically delete the bullets and enemy when both of them leave the screen (go out the visible area)

```ruby
    #--------------- explosion -----------------
    @explosions.dup.each do |explosion|
      @explosions.delete explosion if explosion.done
    end
    #--------------- when bullet hit enemy ------------------------
    @enemies.dup.each do |enemy|
      @bullets.dup.each do |bullet|
        distance = Gosu.distance(enemy.x, enemy.y, bullet.x, bullet.y)
        if distance < bullet.radius + enemy.edge
          @enemies.delete enemy
          @bullets.delete bullet
          @count += 1
          @explosions.push Explosion.new(self, enemy.x,enemy.y)
        end
      end
    end
    #---------------- when player hit enemy ---------------------
    @enemies.each do |enemy|
      distance = Gosu.distance(enemy.x,enemy.y, @ship.x, @ship.y)
      initialize_end(:die) if distance < @ship.rotate + enemy.edge
    end

    initialize_end(:escape) if button_down?(Gosu::KbEscape)
  end
```

Figure 26: update_playing procedures part 2

I also make the explosions after appearing, it will be deleted. When the bullets hit the enemy, 4 things are going to be happened. First and second, the enemy and bullets will be deleted. Third and fourth, the score will increase by 1 and the explosion animation will be displayed.
There are 2 cases to lead to the end of this game: first case, the user press escape to exit, second stage is when the user get hit by the enemy and depend on how the game is ended, the different messages will be shown.

```ruby
def initialize_playing
  @scene = :playing
  @enemies = []
  @bullets = []
  @explosions = []
  @credits =[]
  @count = 0
  @playing_music.play(false)

end
```

Figure 27: initialize_playing procedure

The reason why at this step I start to create the empty array and @count is if I created it sooner, when the user reset the game, the scores and enemy won't be reseted and the user's player will keep die so that's why at this stage I start to create the empty array.

```ruby
def button_down_playing(id)
  if id == Gosu::KbSpace || id == Gosu::MsLeft || id == Gosu::KbReturn
      @bullets.push Bullet.new(self, @ship.x, @ship.y, @ship.edge)
      @shooting_music.play(0.3)
  end
end
```

Figure 29: button_down_playing procedure

For this procedure, everytime I press space for left mouse button, or enter button, the ship will shoot, and the music will be played.

```ruby
class Explosion
  attr_reader :done
  def initialize(window, x, y)
    @x = x
    @y = y
    @radius = 40
    @images = Gosu::Image.load_tiles('images/explosions.png', 60, 60)
    @index = 0
    @done = false

  end

  def draw
    if @index < @images.count
      @images[@index].draw(@x - @radius, @y - @radius, 2)
      @index += 1
    else
      @done = true
    end
  end

end

```

Figure 30: Explosion array in a different file

enemy.rb

```ruby
class Enemy
  SPEED = 5
  attr_reader :x, :y, :edge
  def initialize(window)
    @edge = 10
    @x = rand(window.width - 3 * @edge) + @edge
    @y = 0
    @opponent = Gosu::Image.new('images/enemy.png')
    @window = window
  end
  def draw
    @opponent.draw(@x - @edge, @y - @edge, 1)
  end

  def move
    @y += SPEED
  end
end

```

Figure 31: Enemy array in a different file

```ruby
class Bullet
  SPEED = 30
  attr_reader :x, :y, :radius
  def initialize(window, x, y, angle)
    @x = x
    @y = y
    @direction = 0
    @image = Gosu::Image.new('images/bullet.png')
    @window = window
    @radius = 3
  end

  def draw
    @image.draw(@x - @radius, @y - @radius,1)
  end

  def move
    @x += Gosu.offset_x(@direction,SPEED)
    @y += Gosu.offset_y(@direction,SPEED)
  end

  def onscreen?
    right = @window.width + @radius
    left = -@radius
    top = -@radius
    bottom = @window.height + @radius
    @x > left && @x < right && @y > top && @y < bottom
  end

end

```

Figure 32: bullet array in a different file

6. **Credit page**

First, I need to import the music for the credit stage and the font size.

```
#------------- end --------------------------
@end_music = Gosu::Song.new('musics/end.wav')
@end_message_font = Gosu::Font.new(15)

end
```

Figure 33: set up for credit page

```
def draw_end
  clip_to(60, 140, 1400, 600) do
    @credits.each do |credit|
        credit.draw
    end
  end
  @end_message_font.draw_text(@end_message1, 400, 20, 1, 4,
  ZOrder::TOP, Gosu::Color::WHITE)
  @end_message_font.draw_text(@end_message2, 550, 60, 1, 4,
  ZOrder::TOP, Gosu::Color::WHITE)
  @end_message_font.draw_text(@bottom_end, 250, HEIGHT - 55, 1, 3,
  ZOrder::TOP, Gosu::Color::WHITE)
end
```

Figure 34: draw_end procedure

For the figure 34, the displayed text will be looped to create animation for it, I need to draw a box with 600px height and 1400 px width. A long with that, the messages for the credit will be drawn.

```ruby
def initialize_end(fate)
  @scene = :end
  @end_music.play
  sleep(0.2)
  case fate
  when :die
    @end_message1 = "You was destroyed by enemy ship"
  when :escape
    @end_message1 = "You have exited the game"
  end
  @end_message2 = "Your score: #{@count}"
  @bottom_end = "Press B to back to Menu, Press Q or Escape to exit"
  y = 1300
  File.open('credits.txt').each do |line|
    @credits.push(Credit.new(self, line.chomp, 100, y))
    y += 30
  end
end
```

Figure 35: initialize_end procedure

Depend on what the circumstance the first message will be different. And the current scores will also will be displayed along with the instructions to follow. the system will opened up the credits.txt to display the text from it and printing line by line.

```ruby
def update_end
  @credits.each do |credit|
    credit.move
  end
  if @credits.last.y < 100
      @credits.each do |credit|
          credit.reset
      end
  end
end
```

Figure 36: update_end procedure

For this procedure, it help to scroll the text from the file and reset it so that it will keep displaying.

```ruby
def button_down_end(id)
  if id  == Gosu::KbP
    initialize_selecting
  elsif id == Gosu::KbB
    initialize_start
  elsif id == Gosu::KbQ || id == Gosu::KbEscape
    close
  end
end
```

Figure 37: button_down_end procedures

Base on what the user presses, the previous stage will be recall to display or can be closed immediately by pressing Q or Escape.

credit.rb

```ruby
class Credit
  SPEED = 3
  attr_reader :y

  def initialize(window, description, x, y)
      @x = x
      @y = @init_y = y
      @description = description
      @font = Gosu::Font.new(30)
  end

  def move
      @y -= SPEED
  end

  def draw
      @font.draw_text(@description, @x, @y, 1)
  end

  def reset
      @y = @init_y
  end
end
```

Figure 38: Credit array in a different file