Name: Trung Kien Nguyen

Student ID: 104053642

SWINBURNE UNIVERSITY OF TECHNOLOGY

COS10009

Introduction to Programming

REPORT

Custom Program – Games menu of BLOODMOON

TABLE OF CONTENTS

INTRODUCTION

Simple Games menu application using Ruby programming language (with the libraries of Gosu and Ruby2D)

TRUNG KIEN NGUYEN – 104053642

Bachelor of Computer Science – AI major

Swinburne Unirversity of Technology – Hawthorn Campus – Semester 2 – Year 2022

Nowadays, the game industry is growing like never before, besides famous games with millions of players, complexity, and a huge amount of data, very simple, killing-time games that help to relieve stress after stressful working or studying hours are also gradually becoming a trend worldwide.

On top of that, many game developers now tend to combine many simple games into a single application. This brings a lot of benefits to users, including increasing the variety of experiences as well as saving resources on their devices. Such as Microsoft Solitaire Collection with 5 different card games, Google Play games…

This report will provide detailed information about my Custom Program, a simple games menu that includes four games Gems Collection, Star Wars, 2048, and Overcoming Obstacles, writing in Ruby, a programming language that has emerged recently, along with Gosu library, as well as some related topics about it.

OVERVIEW OF THE PROGRAM

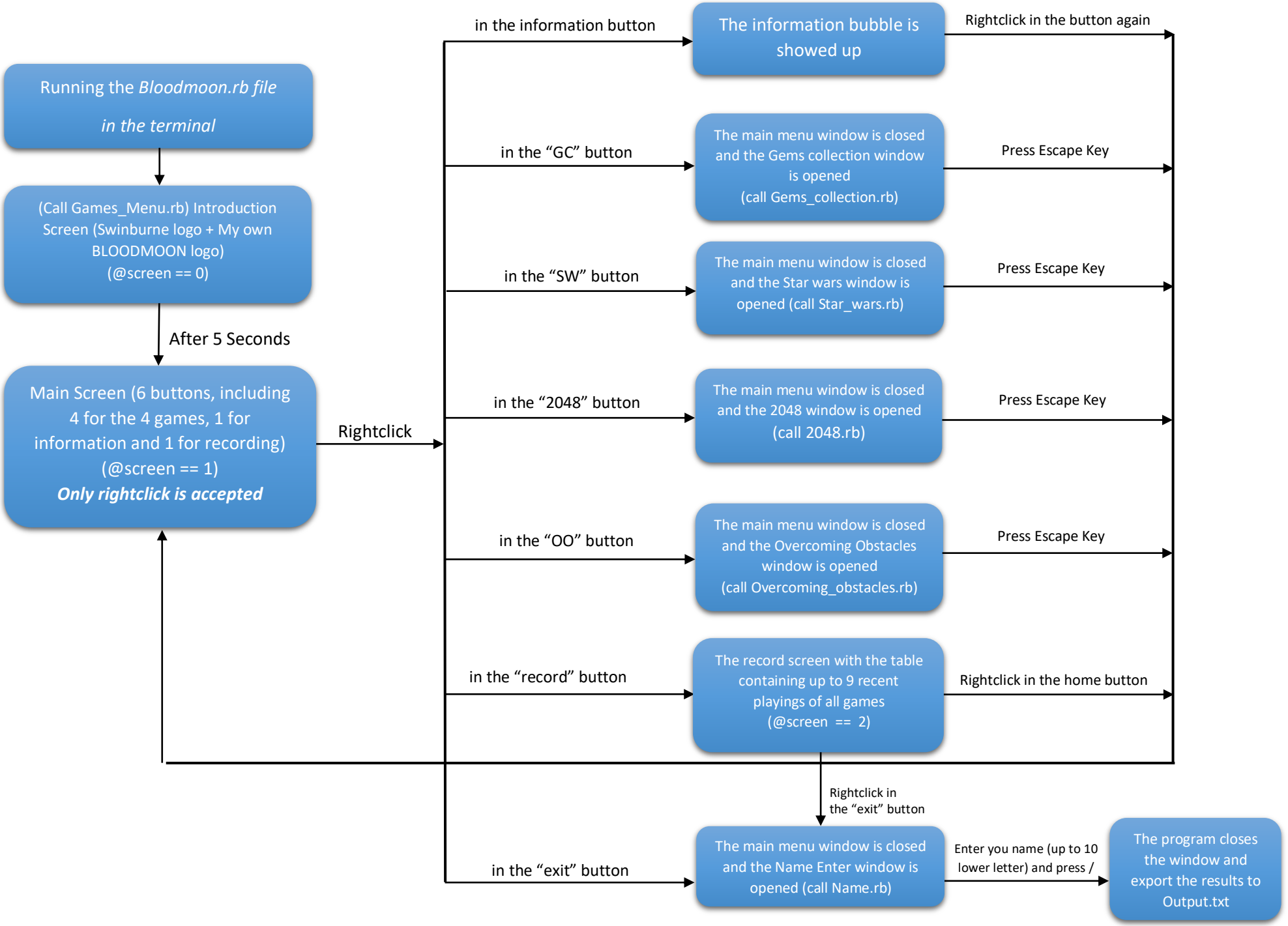
My custom program is developed in Ruby programming language, version 2.5.1p57 [x64-mingw32], with 2 libraries, gosu version 1.4.3, and ruby2d version 0.10.0. It consists of 22 ruby files, including one main file which is *Bloodmoon.rb*, 21 other files for the Main menu, the name-entering, and the four games' windows as well as a *Media* folder (download in [a]) that contains all the media files used for the program, including 50 .png and 7 .jpg (images), 28 .wav and 1 .ogg (songs and sounds).

I have produced the Custom Code Video (download in [b]) that demonstrates the uses of the programs, as well as how to play each game in the portfolio workspace.

Here is the Sequence diagrams of my program, as well as some of the explanations for it:

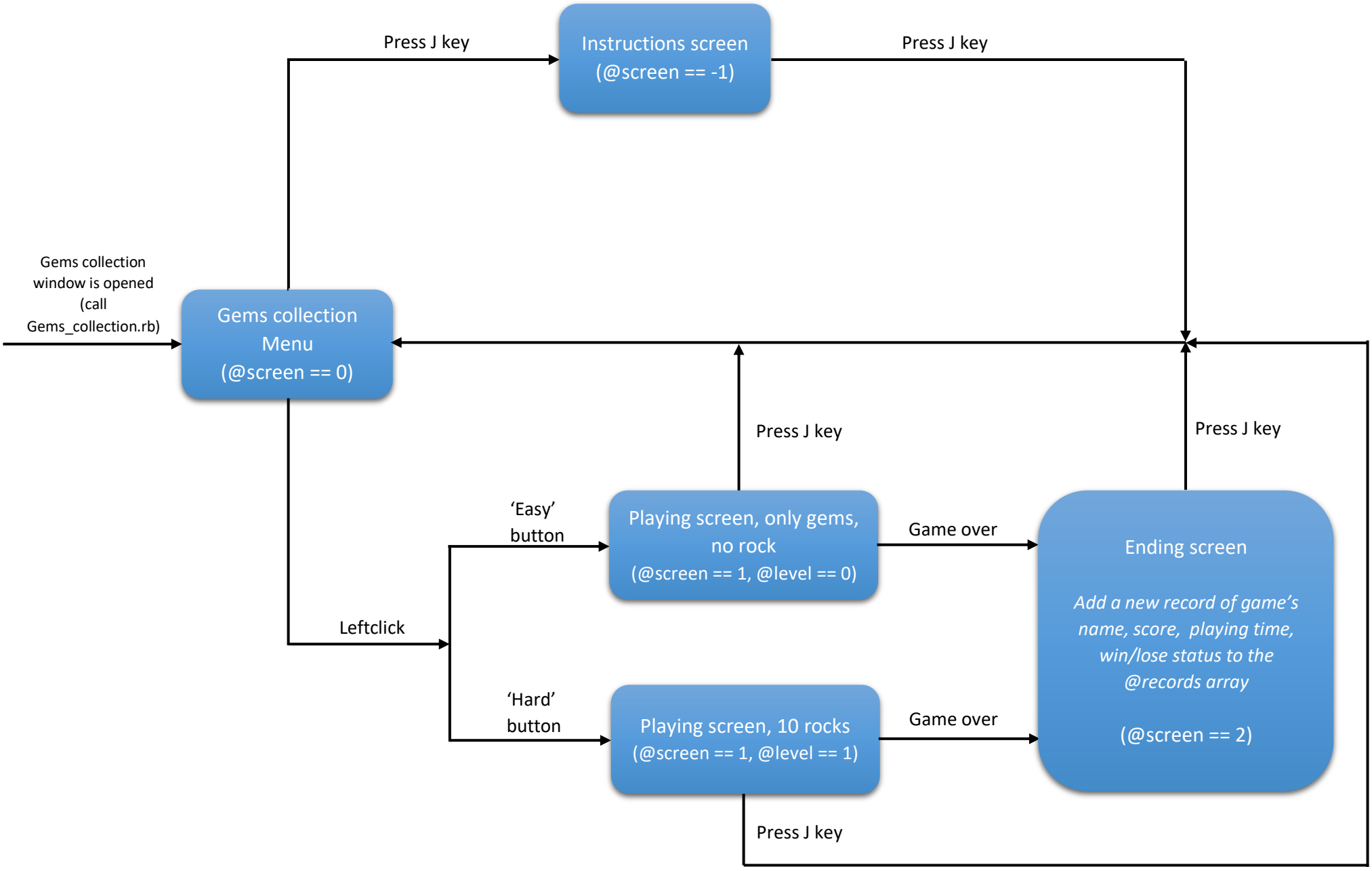
PROGRAM STRUCTURE

The Main menu window:



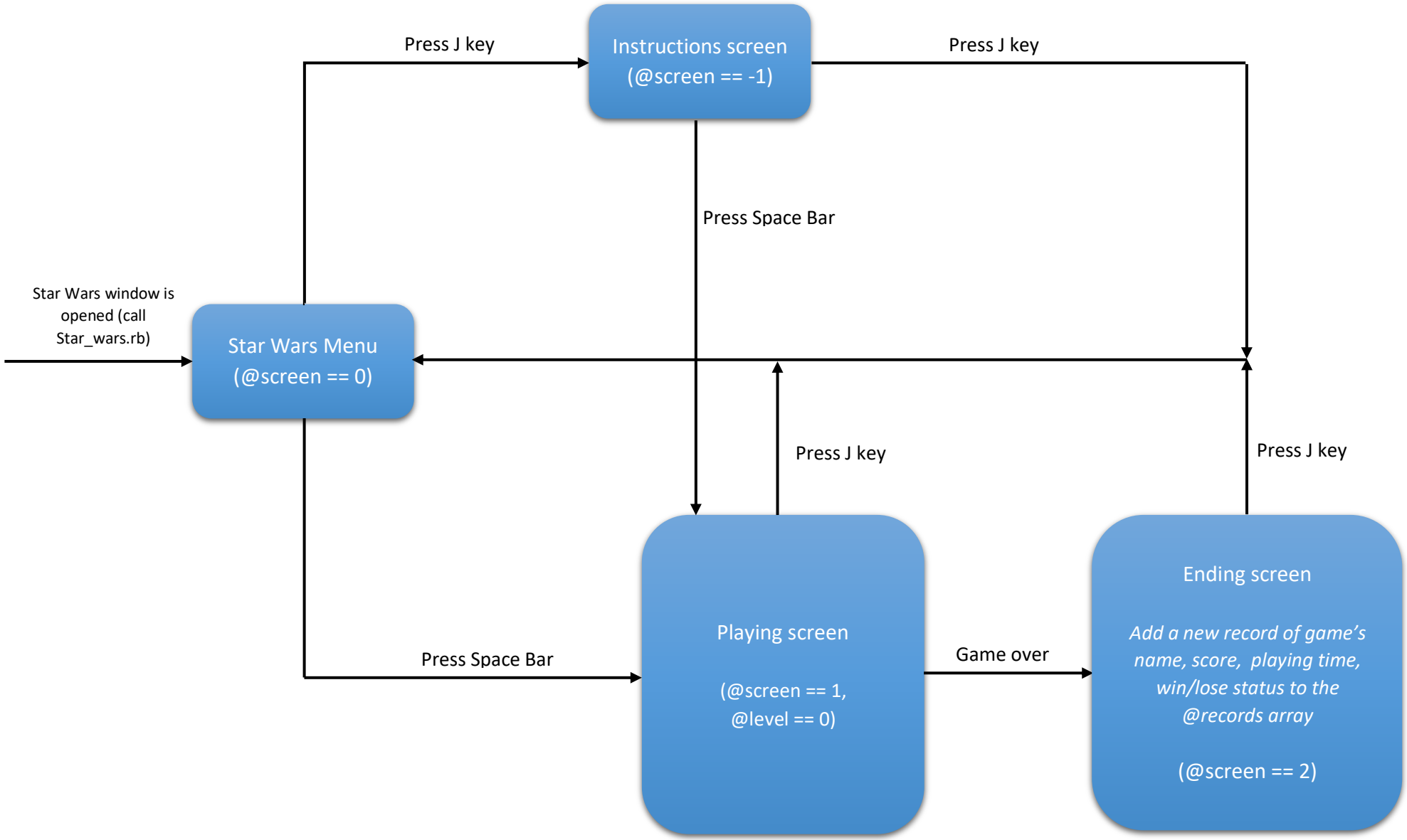
The First game's window – Gems Collection:

(In any steps, player can press Escape key to call Games_Menu.rb with @screen == 1 (no intro screen))



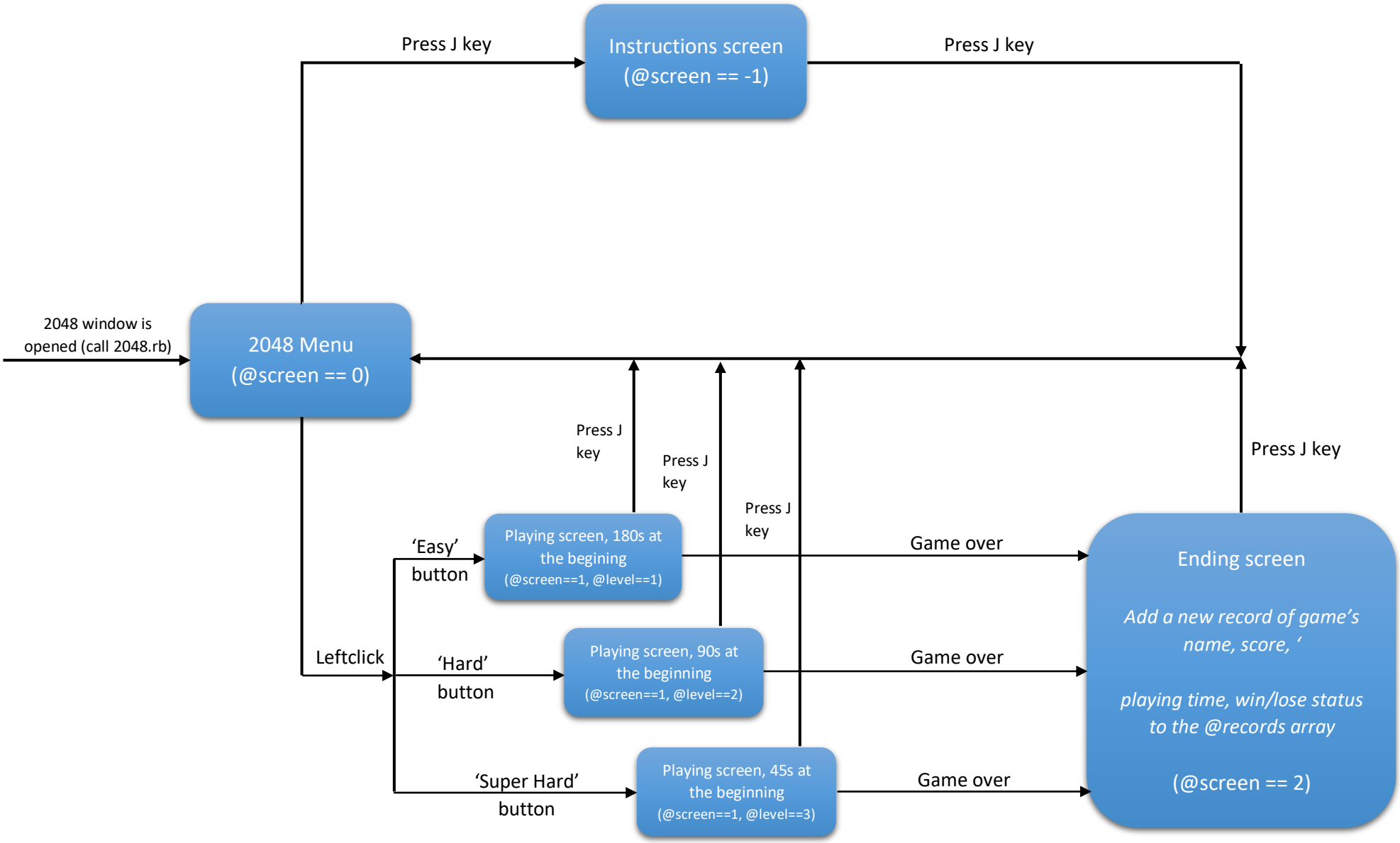
The Second game's window – Star Wars:

(In any steps, player can press Escape key to call Games_Menu.rb with @screen == 1 (no intro screen))



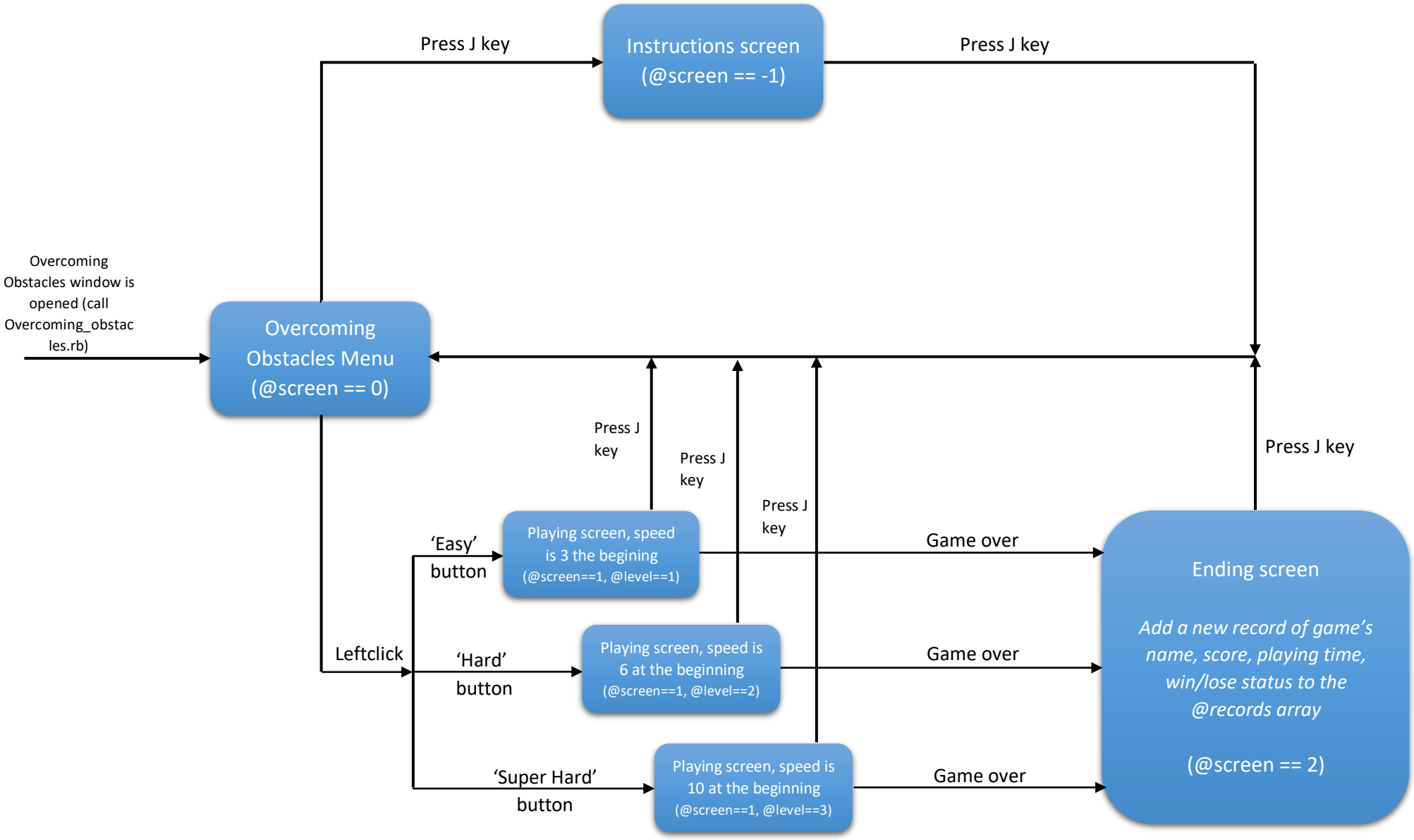
The Third game's window – 2048:

(In any steps, player can press Escape key to call Games_Menu.rb with @screen == 1 (no intro screen))



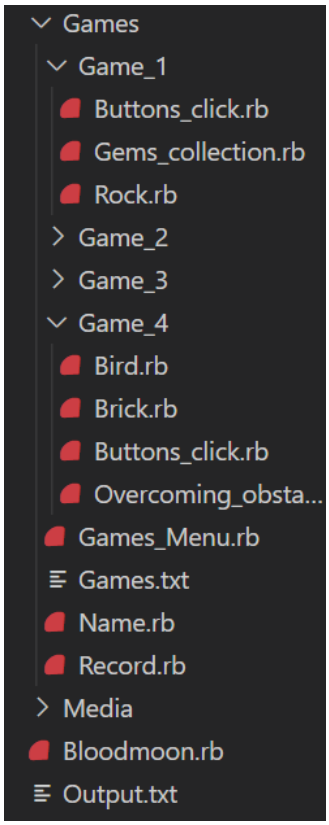
The Last game's window – Overcoming Obstacles:

(In any steps, player can press Escape key to call Games_Menu.rb with @screen == 1 (no intro screen))



SIGNIFICANT FEATURES' EXPLANATION

1) Making the code become high-quality readable: I firmly believe that to do this, I should split the code into multiple files, in different folders. They are divided by their specific purposes in the program, including different functions in a Gosu::Window large class, classes that describe one or more objects in a game, and different windows of the program. Subfiles will be linked to larger files via the require statement, and all of them are linked to *Bloodmoon.rb*.



2) Changing between different screens in each game's window: The basic idea is to set up an instance variable in the main class of each game : @screen, assigning an integer value. Before doing the main functions of a Gosu::Window class, including button_down, update and draw, there will be a branching statement (if..elsif..else..end) to decide which screen the program is currently in. And there are some ways to change this @screen variable: by using mouse or keyboard (implement in button_down) or logic conditions (such as time) in update or draw.

3) Changing between the windows: You can see the program consists of a number of windows (main menu, each game's and name-entering window), the idea is that when the player gives instructions to switch between windows (either from the mouse or the keyboard), the program simply closes the current window with the close statement, and then launches the desired window with the [(Gosu::Window)Class-name].new.show statement:

```
elsif (id == Gosu::KbEscape)      # If player wants to return to the main menu
    close
    Menu.new(1, @records).show
```

4) Combining 2 different libraries (Gosu and Ruby2d) into one program: They are both famous libraries for simple 2D applications using Ruby programming language. The thing is, just creating window classes in the two libraries (Gosu::Window and Ruby2D::Window), and changing between them when needed, but it's only working in theory. In fact, running classes from 2 different libraries at almost the same time will lead to conflicts, and errors then. I have tried many ways, and realize the best combining is to run all Gosu windows first, and the Ruby2D window should be the last one before exiting (mine is the name-entering window). This seems to be the optimal choice for my program.

5) Recording the playing results: To do this, you just need a record class with 4 attributes: :game (game's name), :score, :time (this must be total playing time, as 2048 will have bonus time actually) and :status (victory/defeat), then an array with all elements of this class. But how can it be since the global variables are not encouraged? Fortunately, the classes in Ruby allowed users to use arguments through initialize method. So simply I just need to set up a record (in *Record.rb* file), and then a records array at the beginning, then I pass on that array through different window classes when one is closed and one is opened. Each time player finishes a game, a new element will be added to the array, which will be passed on again while going back to the main menu window that can be able to show up to 9 recent playings (9 last elements).

6) Export the results: After finishing all playing, the player may want to see all his/her playings (not limited to 9 as the record screen of the main window), with the favorite nickname. The program should export all of these outputs to a text file *Output.txt*. But this is not easy, I have tried the same way I do with the Gosu::Window classes, passing on the records array from the Gosu::Window class and the Ruby2D::Window class, and, however, the conflicts happen again. So I have to write all the playing results in a text subfile *Games.txt* when closing the last Gosu windows, and after that, along with entering the name in the last Ruby2D window, the program will read the information in this subfile, and then output all the results to the *Output.txt* file as soon as the program terminates.

(For the games of Gems collection and Star wars, they just the upgraded versions, with advance features, of Whack A Ruby and Sector Five in a Mark Sobkowicz's book. I will only focus in 2048 and Overcoming Obstacles)

7) The game of 2048: This game's idea is simply that the player uses the arrow keys (or A, D, S, and W) to move all the squares to reach 2048. So the major thing that the program has to do is respond to the input from the player's keyboard, and resolve to a 16-element array containing the integer

elements (2, 4, 8, 16, ...). For example, when the player presses the right key, every square in the board, including the newly created squares from the merge, moves to the right side as much as possible. I have made the right method belonging to the Original2048 class (in *Move.rb*) to do this, as well as increase the @bonus_time when needed (>= 32 square has been created)

```ruby
def right(images, j, k)
    index = k
    while (index > j)
        index_1 = index - 1
        if (images[index] != 0)
            if (index > j) && (index <= k)
                while (images[index_1] == 0 ) && (index_1 >= j)
                    index_1 -= 1
                end
                if (images[index] == images[index_1]) && (index_1 >= j)
                    @bonus_time += ((images[index] / 32) * @level) if (images[index] >= 32)
                    @collect_sound.play(0.3)
                    images[index] *= 2
                    images[index_1] = 0
                end
            end
        end
        index -= 1
    end
    index = k
    while (index > j) do
        sum = 0
        for i in j..index
            sum += images[i]
        end
        if (sum != 0)
            while (images[index] == 0) do
                i = index
                while (i >= j) do
                    images[i] = images[i - 1]
                    i -= 1
                end
                images[j] = 0
            end
        end
        index -= 1 if (images[index] != 0) || (sum == 0)
    end
    return images
```

8) The game of Overcoming Obstacles: This game is also known as "Flappy bird". The game requires the player to help to bird pass through the gap between the walls and keep it flying by the Space bar.

Bird.rb

```ruby
class Bird
    attr_accessor :y

    def initialize
        @bird = Gosu::Image.load_tiles('Media/Image_Bird.png', 54, 50)

        @y = 240
        @draw_bird = false
        @bird_index = 0
    end

    def setup
        @y = 240
    end

    def move
        @y += 3
    end

    def jump
        @y -= 6
    end
```

In fact, the bird doesn't move forward, everything moves backward, at different speeds (stored in @speed variable), depending on the difficulty and the total time spent playing the level.

Overcoming_obstacles.rb

```ruby
elsif (id == Gosu::MsLeft)
    # If player have chosen one of the levels
    if (easy_click(mouse_x, mouse_y)) || (hard_click(mouse_x, mouse_y)) || (superhard_click(mouse_x, mouse_y))
        if easy_click(mouse_x, mouse_y)
            @speed = 3
            @level = 1
        elsif hard_click(mouse_x, mouse_y)
            @speed = 6
            @level = 2
        elsif superhard_click(mouse_x, mouse_y)
            @speed = 10
            @level = 3
        end
        @screen = 1
        @screen_1 = @x0 = @x = @speed_change = @first_brick_x = @time = @score = 0
    end
end
```

Overcoming_obstacles.rb

```ruby
    # Changing the speed during the time
    if (@time >= 12000) && (@time < 24000) && (@speed_change == 0)
        @speed += 1
        @speed_change += 1
    elsif (@time >= 24000) && (@time < 36000) && (@speed_change == 1)
        @speed += 1
        @speed_change += 1
    elsif (@time >= 36000) && (@time < 48000) && (@speed_change == 2)
        @speed += 1
        @speed_change += 1
    elsif (@time >= 48000) && (@time < 60000) && (@speed_change == 3)
        @speed += 1
        @speed_change += 1
    elsif (@time >= 60000)
        @screen = 2
    end

    # Game over if the bird drops to the ground
    @screen = 2 if (@bird.y > 440)
```

The game will end if the bird collides with one of the walls:

Overcoming_obstacles.rb

```ruby
    # Check if the bird collides with the walls
    @bricks.dup.each do |brick|
        @bricks.delete brick if (brick.x <= -76)
        if (brick.x <= 178) && (brick.x >= 48)
            @screen = 2 if ((@bird.y - 10) < (brick.space * 80) - 20) || ((@bird.y + 38) >= (brick.space * 80) + 140))
        elsif (brick.x < 47) && (brick.x >= (47 - @speed)) && (@screen == 1)
            @score += (1 * @level)
            @overpass_sound.play(0.4)
        end
    end
```

CONCLUSION

This program is my first 2D game, so I believe it is very simple and has many shortcomings. In the near future, specifically the next related units next semester, I will consider trying out some more complex areas, such as 3D games.

Thank you for reading these words.
If you have any questions or comments about this program, including errors or misunderstandings, please let me know by emailing 104053642@student.swin.edu.au.

REFERENCES

[1] Mark Sobkowicz. Learn Game Programming with Ruby: Bring Your Ideas to Life with Gosu. Retrieved from https://ebookcentral.proquest.com/lib/swin/detail.action?docID=5307452

[2] Ruby 2D - Make cross-platform 2D applications in Ruby. Retrieved from https://www.ruby2d.com/learn/

[3] Images for the program. Retrieved from https://www.google.com.au/imghp?hl=en&authuser=0&ogbl and https://openclipart.org/

[4] Songs and Sounds for the program. Retrieved from https://www.youtube.com/ (Geometry Dash and Star Wars soundtracks), https://freesound.org/ and https://incompetech.com/