BINUS UNIVERSITY
BINUS INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:		**Surname**	**Given Names**	**Student ID Number**
	1.	**Kevin**	**Tarada Darmawan**	**2101693605**

Course Code	**: COMP6335**	**Course Name**	**: Introduction to Programming**
Class	**: L1AC**	**Name of Lecturer(s)**	: 1. Minaldi Loeis 2. Jude Martinez
Major	**: CS**		
Title of Assignment (if any)	: Nyan Adventure!		
Type of Assignment	**: Final Project**		
Submission Pattern			
Due Date	**: 06-11-2017**	**Submission Date**	**: 05-11-2017**

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: (Name of Student)

1. Kevin Tarada Darmawan

“Nyan Adventure!”

Name : Kevin Tarada Darmawan

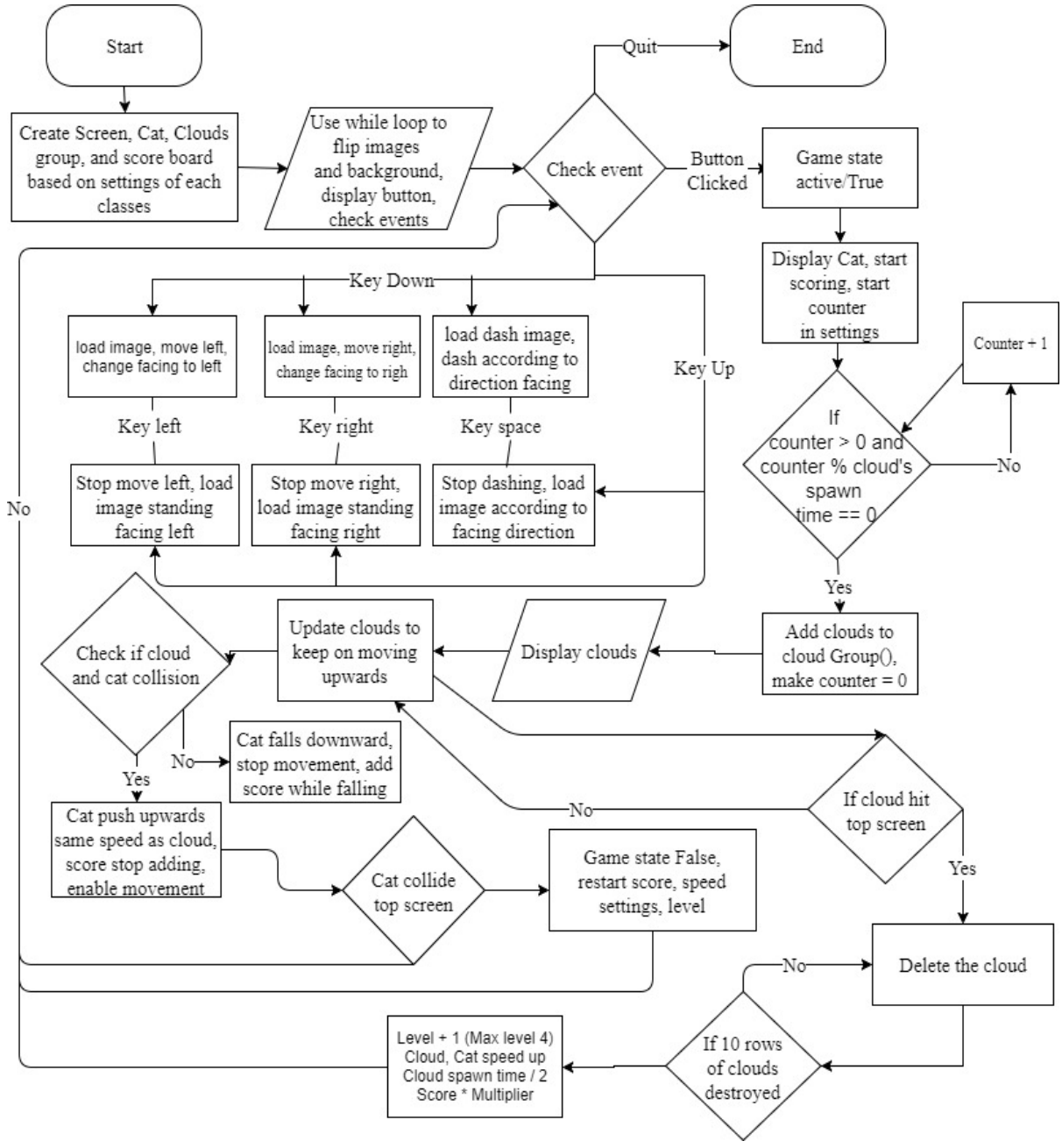
ID : 2101693605

I. Description

This program is basically a high score based game where the player controls a character, Nyan cat. The character must traverse lower as the cloud platform below it ascends. If the character hits the top of the screen, then the player may try again and try to beat the high score.

II.a. Design/Plan

Flow Chart



II.b. Explanation of Each Files

- **Run_Down.py**
  - Import codes (pygame, settings, game functions, cat and cloud settings, game stats, button, scoreboard)
  - In Play_game() function

    Call pygame.init() first. Then assign:

    1. class Settings to variable settings
    2. pygame.time.Clock() to variable clock to set FPS
    3. Button() to play_button
    4. Nyan() to cat, spawn the character which is Nyancat
    5. Set variable cloud to Group() to contain cloud sprites
    6. GameStats() to variable stats, to make highscores, level, game state (is game on or no).
    7. Scoreboard() to variable sb

    Set up caption and screen display according to screen height and width in setting. In a while loop which will keeps on going, check if there is an event queued using check_events() function from game function. If event queued is mouse click 'Run Down!' button, change game_active() to True and run game.

    Run game, start adding counter, check if counter moduled by cloud's spawn time in setting is 0, spawn a row of cloud and restart counter. Check if cloud or cat hit the top screen, if cat hit top screen restart settings and game_active become False. Check if cat collide with clouds, if it collide push cat upwards according to cloud's speed. Then update cloud so it keeps on moving upwards and cat so it can keeps on moving (left, right and dash). Use function to draw stuffs to screen. Set FPS as 80 using clock.tick().

  - Call Play_game() function

- **settings.py**

  Class Settings()

  - Def __init__ () initial basic settings of:
    - Screen height, width, background
    - Counters for cloud destroyed and spawning clouds
    - Set scale for speed up and scoring when level up

- Call initialize_dynamic_settings()

- Initialize_dynamic_settings()

  This settings is separated from the __init__() so it can be call whenever 'Run Down!' button is clicked so it restart the settings.

  - Set cat and cloud speed
  - Set points when falling of clouds and touch ground
- Increase_speed()

  This function is called whenever the game level up

  - Cat and cloud speed multiply by speed up scale
  - Hit ground and falling from cloud scores multiply by scoring scale
  - Cloud spawn time divide by speed up scale
- Add_counter()

  Counter is placed in this class' __init__() so we must use a function to add the counter.
- Reset_counter()

  Same as add_counter() we must use a function to reset the counter

- **button.py**

  Import python.font

  Class Button():

  - Def __init__(settings, screen, msg)

    Initialize the button's attributes

    - Set self.screen into screen
    - Set width and height of button, button and text color, and font from pygame.font
    - Make a rectangle according to self's height and width, then put it in center of screen
    - Set msg as message that will be written
  - Prep_msg(msg)

    Turn message into a rendered image and center the text on button
  - Draw_button()

    Draw blank button and draw the message

- **nyancat.py**

  Import pygame and pygame's Sprite

  Class Nyan(Sprite), put class Nyan to inherit class Sprite

  - Def __init__(screen, setting, x, y)

    Super the Sprite class to inherit

    - Set variable for screen and setting
    - Set up image to its rectangle
    - Set position of float of input x and y to enable it to be able to be added 0.5 when moving later
    - Set a moving state to prevent cat to move while falling (if cat move while falling it will collide with cloud and get stuck)
    - Set move left, right and dash state which can be change through game function when pressing button
    - Set facing direction to determine dash direction
    - Make rect x and y into float to enable adding by 0.5 points when move
  - Def update(screen, setting)

    Check if dash is true, if True then see the facing direction then load dash image and moves towards it twice the speed of moving left/right. Else check if moving left/right is True, if True then load image and move towards the direction equal to cat speed in setting, also if cat move out of screen, put it in the edge. This order priorities Dash rather than moving
  - Def blitme()

    Draw cat at its current location
  - Def restart_position(setting)

    Put cat to initial position

- **cloud.py**

  Import pygame and pygame's Sprite

  Class Cloud(Sprite), put class Cloud to inherit class Sprite

  - Def __init__(screen, setting)

    Super the Sprite class to inherit

    - Set variable for screen and setting
    - Set up image to its rectangle
    - Set the x and y of cloud

- Set position of float of input x and y to enable it to be able to be added 0.5 when moving later

- Def update()

  Move cloud upwards

- Def blitme()

  Draw cloud at its current location

- **game_stats.py**
  - Def __init__(setting)
    - Set variable for setting
    - Set game active True or False
    - Set high score to 0
    - Call reset_stats()
  - Def reset_stats()

    Set Cloud, Level, Cat and Cloud speed to initial amount in setting

- **scoreboard.py**

  Import font in pygame

  Class Scoreboard()
  - Def __init__(setting, screen, stats)
    - Set variable for screen and setting
    - Set text color and font
    - Use alpha to make background of score invincible
    - Call prep_score()
    - Call prep_highscore()
    - Call prep_level()
  - Def prep_score()
    - Turn score into a rendered image
    - Put score location in top right of screen
  - Def prep_highscore()
    - Turn highscore into a rendered image
    - Put highscore location in top center of screen

- Def prep_level()
    - Turn level into a rendered image
    - Put level location in top left of screen

- **game_functions.py**

  Import sys, pygame, random and Cloud() and Nyan() class

  - Def check_events(cat, setting, stats, play_button, cloud, screen, sb)

    Use for loop to get event in pygame, then
    - If event is quit then system exit which is close window
    - If event is mouse button down, get position of x and y of mouse. Then call function check_play_button(stats, play_button, mouse_x, mouse_y, cloud, cat, setting, screen, sb)
    - If event is Key down, check the move state to see if cat is allowed to move or not

      If yes check:
        - If key right, change cat facing direction to right, and make cat.move_right = True (which will activate moving when updating cat)
        - If key left, change cat facing direction to left, and make cat.move_left = True (which will activate moving when updating cat)
        - If key space, make cat.dash = True (which will activate moving when updating cat)

      If no check:
        - If key right, change cat facing direction to right.
        - If key left, change cat facing direction to left.
    - If event is Key up, check:
        - If key right, load cat facing right while standing image, and make cat.move_right = False (which will make it stop moving)
        - If key left, load cat facing left while standing image, and make cat.move_left = False (which will make it stop moving)
        - If key space, make cat.dash = False (which will make it stop dashing when updating cat), and also load image based on the direction faced when dashing

  - Def check_play_button(stats, play_button, mouse_x, mouse_y, cloud, cat, setting, screen, sb)

    When button collide with mouse, game active into True, use stats.reset_stats to reset level, score and cat and cloud speed. Reset counter using setting.reset_counter(). Call function sb.prep_score(),

sb.prep_highscore(), sb.prep_level() to display score, highsore, and level. Restart cat position and empty cloud. Then start playing music.

- Def draw_screens(setting, screen, cat, cloud, stats, play_button, sb)

  Blit back ground, cat, clouds in cloud Group(), show score, and if game is not active blit play button too, then flip everything for movements/update.

- Def create_fleet(setting, screen, cloud)

  Make a list len of 8, consists of one 0 and seven 1. Randomise position of 0, then use for loop to check wether its 1 or 0. If 1 then add cloud to Group() (it's rect.x = cloud_width multiply by order in loop). If 0 continue the loop.

- Def check_cat_under(setting, screen, cat, cloud, sb, stat)

  If cat and cloud collide, enable cat movement and push cat upwards same speed as cloud.

  If not collide:

  - check if cat bottom touch bottom screen, if touch ground then the score will be add up with touch_ground's score.
  - check if cat bottom below bottom screen, if yes then cat's rect bottom will be move to screen's bottom
  - check if cat bottom smaller than bottom screen, if yes, move cat downward with same speed of cloud, add score equals to setting.cloud_points. Make cat unable to move, make dash, move left and right into False, so it won't get stuck at cloud's side when falling.
  - Then update score and highscore

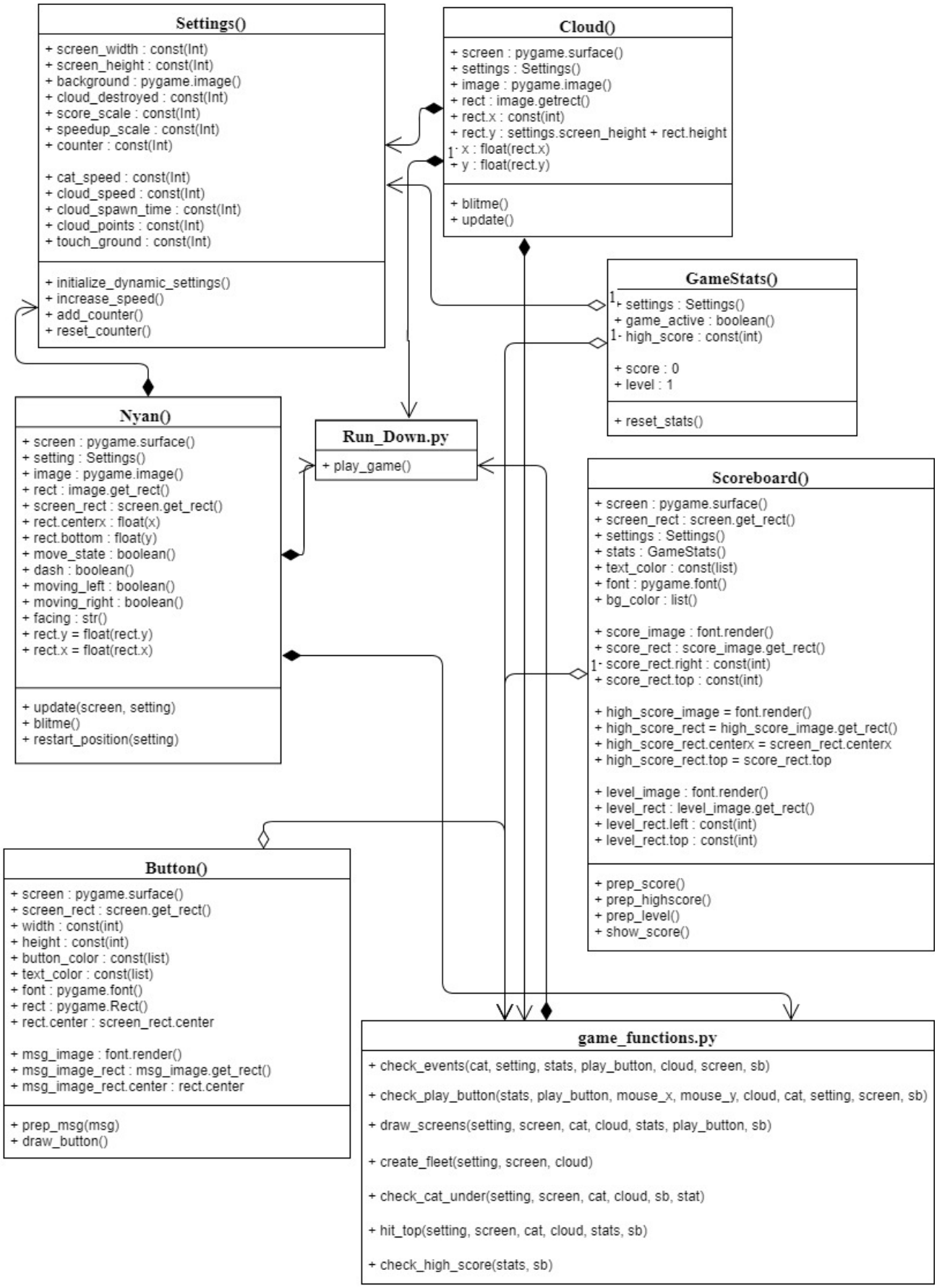
- Def hit_top(setting, screen, cat, cloud, stats, sb)

  This function cleans up the cloud Group() when it hit top screen, and for every 10 rows of clouds deleted add level to 1 (if level reach 4 change it to Max so it can never speed up again). If cat hit top music fade then return True which will stop the game in Run_Down.py.

- Def check_high_score(stats, sb)

  If score is bigger than highscore then replace highscore with score, then show highscore

UML Diagram:

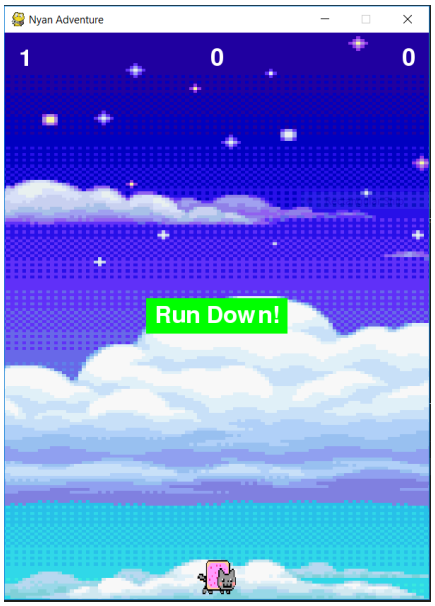

Settings()
+ screen_width : const(Int)
+ screen_height : const(Int)
+ background : pygame.image()
+ cloud_destroyed : const(Int)
+ score_scale : const(Int)
+ speedup_scale : const(Int)
+ counter : const(Int)
+ cat_speed : const(Int)
+ cloud_speed : const(Int)
+ cloud_spawn_time : const(Int)
+ cloud_points : const(Int)
+ touch_ground : const(Int)
+ initialize_dynamic_settings()
+ increase_speed()
+ add_counter()
+ reset_counter()
Cloud()
+ screen : pygame.surface()
+ settings : Settings()
+ image : pygame.image()
+ rect : image.getrect()
+ rect.x : const(int)
+ rect.y : settings.screen_height + rect.height
- x : float(rect.x)
- y : float(rect.y)
+ blitme()
+ update()
GameStats()
- settings : Settings()
+ game_active : boolean()
- high_score : const(int)
+ score : 0
+ level : 1
+ reset_stats()
Nyan()
+ screen : pygame.surface()
+ setting : Settings()
+ image : pygame.image()
+ rect : image.get_rect()
+ screen_rect : screen.get_rect()
+ rect.centerx : float(x)
+ rect.bottom : float(y)
+ move_state : boolean()
+ dash : boolean()
+ moving_left : boolean()
+ moving_right : boolean()
+ facing : str()
+ rect.y = float(rect.y)
+ rect.x = float(rect.x)
+ update(screen, setting)
+ blitme()
+ restart_position(setting)
Run_Down.py
+ play_game()
Scoreboard()
+ screen : pygame.surface()
+ screen_rect : screen.get_rect()
+ settings : Settings()
+ stats : GameStats()
+ text_color : const(list)
+ font : pygame.font()
+ bg_color : list()
+ score_image : font.render()
+ score_rect : score_image.get_rect()
- score_rect.right : const(int)
+ score_rect.top : const(int)
+ high_score_image = font.render()
+ high_score_rect = high_score_image.get_rect()
+ high_score_rect.centerx = screen_rect.centerx
+ high_score_rect.top = score_rect.top
+ level_image : font.render()
+ level_rect : level_image.get_rect()
+ level_rect.left : const(int)
+ level_rect.top : const(int)
+ prep_score()
+ prep_highscore()
+ prep_level()
+ show_score()
Button()
+ screen : pygame.surface()
+ screen_rect : screen.get_rect()
+ width : const(int)
+ height : const(int)
+ button_color : const(list)
+ text_color : const(list)
+ font : pygame.font()
+ rect : pygame.Rect()
+ rect.center : screen_rect.center
+ msg_image : font.render()
+ msg_image_rect : msg_image.get_rect()
+ msg_image_rect.center : rect.center
+ prep_msg(msg)
+ draw_button()
game_functions.py
+ check_events(cat, setting, stats, play_button, cloud, screen, sb)
+ check_play_button(stats, play_button, mouse_x, mouse_y, cloud, cat, setting, screen, sb)
+ draw_screens(setting, screen, cat, cloud, stats, play_button, sb)
+ create_fleet(setting, screen, cloud)
+ check_cat_under(setting, screen, cat, cloud, sb, stat)
+ hit_top(setting, screen, cat, cloud, stats, sb)
+ check_high_score(stats, sb)

II. User Manual

In this game you'll be Nyan cat, there will be clouds pushing you upwards, your objective is to not let them push you to the top side of the screen simply by falling through the gap of each clouds' row. You'll get points whenever you go through the cloud's gap and even bigger points when you touch the ground. So, try to last as long as possible and aim for the high score.

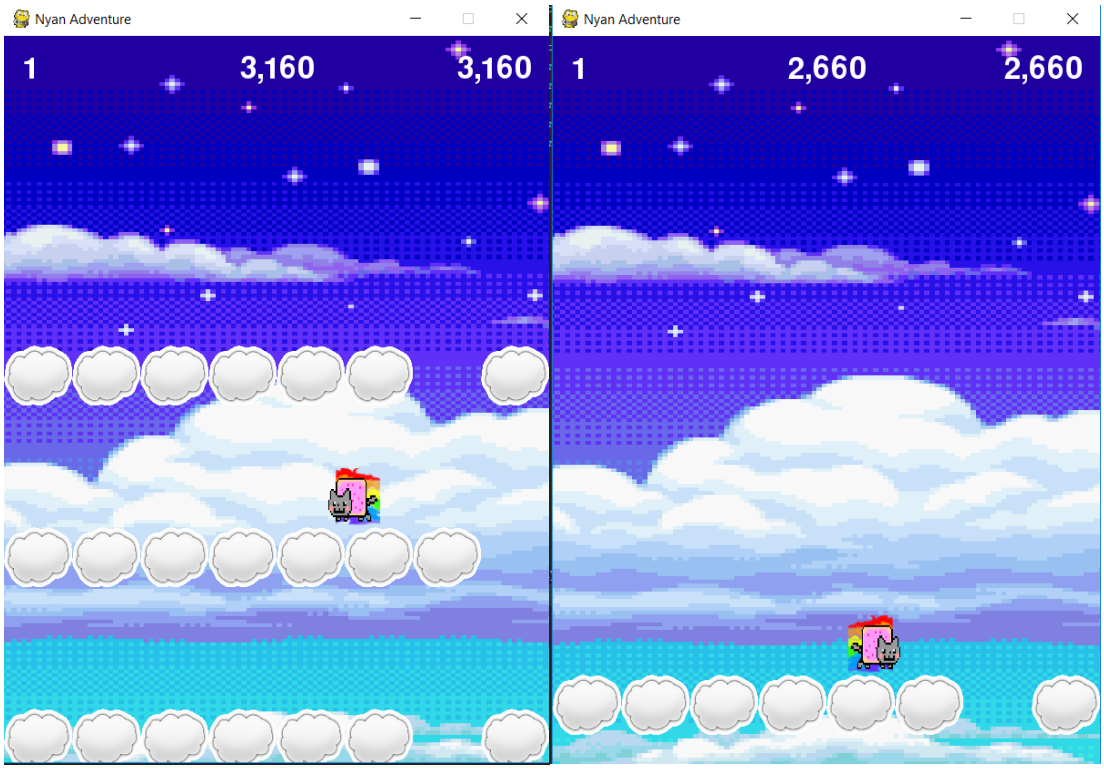
After running the game, here are the steps and controls on how to play the game:

1. Click the Run Down! Button to play

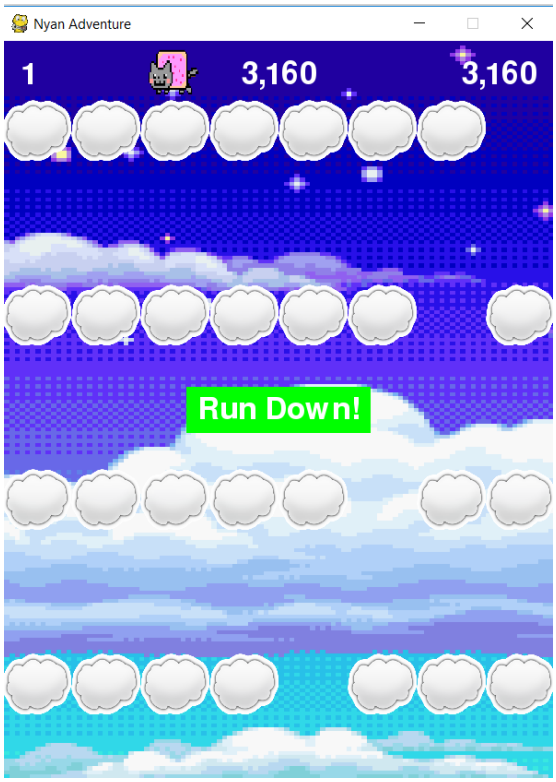


2. Press left arrow key to move left and right arrow key to move right

3. You can also dash according to the direction you're facing



4. When touching the top the game will end, to play again simply click again the Run Down! Button.



III. Source Code

1. Run_Down.py

```
import pygame
from Final_project.settings import Settings
import Final_project.game_functions as gf
from pygame.sprite import Group
from Final_project.nyancat import Nyan
from Final_project.game_stats import GameStats
from Final_project.button import Button
from Final_project.scoreboard import Scoreboard


def play_game():
    # Create screen objects
    pygame.init()
    setting = Settings()
    screen = pygame.display.set_mode((setting.screen_width, setting.screen_height))

    # Clock for FPS
    clock = pygame.time.Clock()

    # Caption
    pygame.display.set_caption("Nyan Adventure")

    # Button
    play_button = Button(setting, screen, "Run Down!")

    # Make Cat to screen
    cat = Nyan(screen, setting, setting.screen_width/2, setting.screen_height)
    cloud = Group()

    # Set up scoreboards
    stats = GameStats(setting)
    sb = Scoreboard(setting, screen, stats)


    while True:
        # Use check event
        gf.check_events(cat, setting, stats, play_button, cloud, screen, sb)

        if stats.game_active:
            # Start counter to generate clouds
            setting.add_counter()

            if setting.counter % setting.cloud_spawn_time == 0 and setting.counter > 0:
                gf.create_fleet(setting, screen, cloud)
```

```
            setting.reset_counter()


        # Check if cat/cloud hit top screen
        if gf.hit_top(setting, screen, cat, cloud, stats, sb):
            stats.game_active = False
            pygame.mouse.set_visible(True)
            setting.initialize_dynamic_settings()

        # Check if cat on top of cloud
        gf.check_cat_under(setting, screen, cat, cloud, sb, stats)

        # Update cat and cloud
        cat.update(screen, setting)
        cloud.update()

    # Use function to draw things to screen
    gf.draw_screens(setting, screen, cat, cloud, stats, play_button, sb)

    # Set FPS
    clock.tick(80)

# Call main
play_game()
```

2. settings.py

```
import pygame


class Settings:
    def __init__(self):

        # Screen settings
        self.screen_width = 600
        self.screen_height = 800
        self.background = pygame.transform.scale(pygame.image.load('final proj bg.png'),
(self.screen_width, self.screen_height))

        # Cloud destroyed counter
        self.cloud_destroyed = 0

        # Scale of how much score values increase/level
        self.score_scale = 2

        # How quickly the game speeds up
```

```
        self.speedup_scale = 2


        # Counter for cloud
        self.counter = 0

        self.initialize_dynamic_settings()

    def initialize_dynamic_settings(self):
        # Initialize settings that change throughout the game.
        # Cat speed
        self.cat_speed = 3

        # Cloud speed
        self.cloud_speed = 1

        # Cloud spawn time
        self.cloud_spawn_time = 200

        # Cloud points and touch ground points
        self.cloud_points = 5
        self.touch_ground = 10

    def increase_speed(self):
        #Increase speed settings.
        self.cat_speed += self.speedup_scale

        self.cloud_spawn_time /= self.speedup_scale
        self.cloud_speed += self.speedup_scale

        self.cloud_points *= self.score_scale
        self.touch_ground *= self.score_scale

    def add_counter(self):
        self.counter += 1

    def reset_counter(self):
        self.counter = 0
```

3. cloud.py

```
import pygame
from pygame.sprite import Sprite


class Cloud(Sprite):
```

```
    # Make cloud
    def __init__(self, screen, setting):
        super(Cloud, self).__init__()
        self.screen = screen
        self.settings = setting

        # Load the cloud image and set its rect attribute
        self.image = pygame.transform.scale(pygame.image.load('cloud.png'),
(int(setting.screen_width/8), int(setting.screen_height/12)))
        self.rect = self.image.get_rect()

        # Start each new cloud near the bottom left of the screen.
        self.rect.x = 0
        self.rect.y = setting.screen_height + self.rect.height

        # Store the cloud's exact position.
        self.x = float(self.rect.x)
        self.y = float(self.rect.y)

    def blitme(self):
        # Draw the cloud
        self.screen.blit(self.image, self.rect)

    def update(self):
        # Make move
        self.y -= self.settings.cloud_speed
        self.rect.y = self.y
```

4. nyancat.py

```
import pygame
from pygame.sprite import Sprite


class Nyan(Sprite):
    def __init__(self, screen, setting, x, y):
        super(Nyan, self).__init__()

        # Put cat's position as screen
        self.screen = screen
        self.settings = setting

        # Load the cat image and get its rect.
        self.image = pygame.transform.scale(pygame.image.load('nyanz.bmp'),
(int(setting.screen_width/10), int(setting.screen_height/12)))
        self.rect = self.image.get_rect()
```

```
        self.screen_rect = screen.get_rect()

        # Start new cat at the bottom center of the screen.
        self.rect.centerx = float(x)
        self.rect.bottom = float(y)

        # Moving state
        self.move_state = True
        self.dash = False
        self.moving_left = False
        self.moving_right = False

        # To determine dash direction
        self.facing = "right"

        # Make x and y into float
        self.rect.y = float(self.rect.y)
        self.rect.x = float(self.rect.x)

    def update(self, screen, setting):
        # Check movement state to move
        if self.dash is True:
            if self.facing == "right":
                self.image = pygame.transform.scale(pygame.image.load('nyannitro.bmp'),
(int(setting.screen_width/10), int(setting.screen_height/12)))
                self.rect.centerx += setting.cat_speed*2

                # Dash limit right
                if self.rect.right > setting.screen_width:
                    self.rect.right = setting.screen_width

            elif self.facing == "left":
                self.image =
pygame.transform.flip(pygame.transform.scale(pygame.image.load('nyannitro.bmp'),
(int(setting.screen_width/10), int(setting.screen_height/12))), True, False)
                self.rect.centerx -= setting.cat_speed*2

                # Dash limit left
                if self.rect.left < 0:
                    self.rect.left = 0

        elif self.moving_left is True and self.rect.left > 0:
            self.image =
pygame.transform.flip(pygame.transform.scale(pygame.image.load('nyanrenbo.bmp'),
(int(setting.screen_width/10), int(setting.screen_height/12))), True, False)
            self.rect.centerx -= setting.cat_speed
```

```
        elif self.moving_right is True and self.rect.right < setting.screen_width:
            self.image = pygame.transform.scale(pygame.image.load('nyanrenbo.bmp'),
(int(setting.screen_width/10), int(setting.screen_height/12)))
            self.rect.centerx += setting.cat_speed

    def blitme(self):
        # Draw the cat at its current location.
        self.screen.blit(self.image, self.rect)

    def restart_position(self, setting):
        # Make x to center
        self.rect.centerx = setting.screen_width/2
        self.rect.bottom = setting.screen_height
```

5. button.py

```
import pygame.font


class Button():
    def __init__(self, settings, screen, msg):
        # Initialize button attributes.
        self.screen = screen
        self.screen_rect = screen.get_rect()

        # Dimension and properties of the button
        self.width, self.height = 200, 50
        self.button_color = (0, 255, 0)
        self.text_color = (255, 255, 255)
        self.font = pygame.font.SysFont(None, 48)

        # Build the button's rect object and center it.
        self.rect = pygame.Rect(0, 0, self.width, self.height)
        self.rect.center = self.screen_rect.center

        # The button message needs to be prepped only once.
        self.prep_msg(msg)

    def prep_msg(self, msg):
        # Turn msg into a rendered image and center text on the button.
        self.msg_image = self.font.render(msg, True, self.text_color, self.button_color)
        self.msg_image_rect = self.msg_image.get_rect()
        self.msg_image_rect.center = self.rect.center

    def draw_button(self):
```

```
        # Draw blank button and then draw message.
        self.screen.fill(self.button_color, self.rect)
        self.screen.blit(self.msg_image, self.msg_image_rect)
```

6. game_stats.py

```
class GameStats():
    def __init__(self, settings):
        # Initialize statistics.
        self.settings = settings
        self.game_active = False
        self.reset_stats()

        # Initial highscore
        self.high_score = 0


    def reset_stats(self):
        # Initialize statistics that can change during the game.
        self.score = 0
        self.level = 1

        self.cloud_speed = self.settings.cloud_speed
        self.cat_speed = self.settings.cat_speed
```

7. scoreboard.py

```
import pygame.font


class Scoreboard():
    # A class to report scoring information.
    def __init__(self, settings, screen, stats):
        # Initialize score-keeping attributes.
        self.screen = screen

        self.screen_rect = screen.get_rect()
        self.settings = settings
        self.stats = stats

        # Font settings for scoring information.
        self.text_color = (255, 255, 255)
        self.font = pygame.font.SysFont(None, 48)

        # Make bg of score transparent
        alpha = 255
```

```
        self.bg_color = ((255, 255, 255, alpha), None, pygame.BLEND_RGBA_MULT)

        # Prepare the initial score image.
        self.prep_score()
        self.prep_highscore()
        self.prep_level()

    def prep_score(self):
        # Turn the score into a rendered image
        rounded_score = int(round(self.stats.score, -1))
        score_str = "{:,}".format(rounded_score)
        self.score_image = self.font.render(score_str, True, self.text_color, self.bg_color)

        # Display the score at the top right of the screen.
        self.score_rect = self.score_image.get_rect()
        self.score_rect.right = self.screen_rect.right - 20
        self.score_rect.top = 20

    def prep_highscore(self):
        # Turn the high score into a rendered image.
        high_score = int(round(self.stats.high_score, -1))
        high_score_str = "{:,}".format(high_score)
        self.high_score_image = self.font.render(high_score_str, True, self.text_color,
self.bg_color)

        # Center the high score at the top of the screen.
        self.high_score_rect = self.high_score_image.get_rect()
        self.high_score_rect.centerx = self.screen_rect.centerx
        self.high_score_rect.top = self.score_rect.top

    def show_score(self):
        # Draw scores & level to the screen.
        self.screen.blit(self.score_image, self.score_rect)
        self.screen.blit(self.high_score_image, self.high_score_rect)
        self.screen.blit(self.level_image, self.level_rect)

    def prep_level(self):
        # Turn the level into a rendered image.
        self.level_image = self.font.render(str(self.stats.level), True, self.text_color, self.bg_color)

        # Position the level below the score.
        self.level_rect = self.level_image.get_rect()
        self.level_rect.left = 20
        self.level_rect.top = 20
```

8. game_functions.py

```
import sys
import pygame
from Final_project.cloud import Cloud
import random
from Final_project.nyancat import Nyan


def check_events(cat, setting, stats, play_button, cloud, screen, sb):
    # Respond to key-presses and mouse events.
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            # Check if quit button is clicked and stop game
            sys.exit()

        elif event.type == pygame.MOUSEBUTTONDOWN:
            # Check if mouse click button
            mouse_x, mouse_y = pygame.mouse.get_pos()
            check_play_button(stats, play_button, mouse_x, mouse_y, cloud, cat, setting, screen,
sb)

        # Determine between key-down and up
        elif event.type == pygame.KEYDOWN:

            # If move state is True then move, else only cat facing change
            if cat.move_state == True:
                if event.key == pygame.K_RIGHT:
                    # Move right
                    cat.facing = "right"
                    cat.moving_right = True

                if event.key == pygame.K_LEFT:
                    # Move left
                    cat.facing = "left"
                    cat.moving_left = True

                if event.key == pygame.K_SPACE:
                    # Dash
                    cat.dash = True

            else:
                if event.key == pygame.K_RIGHT:
                    # Face right
                    cat.facing = "right"

                if event.key == pygame.K_LEFT:
```

```
                # Face left
                cat.facing = "left"

        elif event.type == pygame.KEYUP:
            if event.key == pygame.K_RIGHT:
                # Change pic and stop move right
                cat.image = pygame.transform.scale(pygame.image.load('nyanz.bmp'),
(int(setting.screen_width/10), int(setting.screen_height/12)))
                cat.moving_right = False

            if event.key == pygame.K_LEFT:
                # Change pic and stop move left
                cat.image =
pygame.transform.flip(pygame.transform.scale(pygame.image.load('nyanz.bmp'),
(int(setting.screen_width/10), int(setting.screen_height/12))), True, False)
                cat.moving_left = False

            if event.key == pygame.K_SPACE:
                cat.dash = False
                # Return pic
                if cat.facing == "left":
                    cat.image =
pygame.transform.flip(pygame.transform.scale(pygame.image.load('nyanz.bmp'),
(int(setting.screen_width/10), int(setting.screen_height/12))), True, False)

                elif cat.facing == "right":
                    cat.image = pygame.transform.scale(pygame.image.load('nyanz.bmp'),
(int(setting.screen_width/10), int(setting.screen_height/12)))

def check_play_button(stats, play_button, mouse_x, mouse_y, cloud, cat, setting, screen, sb):

    button_clicked = play_button.rect.collidepoint(mouse_x, mouse_y)
    if button_clicked and not stats.game_active:
        # Restart speed settings of all things
        setting.initialize_dynamic_settings()

        # Hide the mouse cursor
        pygame.mouse.set_visible(False)

        # Start a new game when the player clicks Run Down!.
        if play_button.rect.collidepoint(mouse_x, mouse_y):

            # Reset the game statistics.
            stats.reset_stats()
            stats.game_active = True
```

```
        # Reset the scoreboard images.
        sb.prep_score()
        sb.prep_highscore()
        sb.prep_level()

        # Reset counter
        setting.reset_counter()

        # Empty the list of clouds.
        cloud.empty()

        # Create a new cat and center it.
        cat.restart_position(setting)

        # Music
        pygame.mixer.music.load('Nyancat.mp3')
        pygame.mixer.music.play(-1)

def draw_screens(setting, screen, cat, cloud, stats, play_button, sb):

    # Blit/Draw things to screen
    screen.blit(setting.background, (0, 0))

    # Blit cats
    cat.blitme()

    # Blit clouds
    for clouds in cloud:
        clouds.blitme()

    # Draw the score information.
    sb.show_score()

    # Draw the play button if the game is inactive.
    if not stats.game_active:
        play_button.draw_button()

    # Make the most recently drawn screen visible.
    pygame.display.flip()

def create_fleet(setting, screen, cloud):
    # If counter reached spawn time for cloud
        # Randomise no-cloud position
        positions = [0, 1, 1, 1, 1, 1, 1, 1]

        # if counter % setting.cloud_spawn_time == 0:
```

```
        x = random.randint(0, 7)
        positions[0], positions[x] = positions[x], positions[0]

        for i in range(len(positions)):
            # Make a cloud to add
            clouds = Cloud(screen, setting)
            cloud_width = clouds.rect.width

            # If value is 1, add cloud with position of x = width * order
            if positions[i] == 1:
                clouds.x = cloud_width * i
                clouds.rect.x = clouds.x
                cloud.add(clouds)

            else:
                # Add nothing
                continue

def check_cat_under(setting, screen, cat, cloud, sb, stat):

    # Check cat on top of cloud
    if pygame.sprite.spritecollideany(cat, cloud):
        # Push cat up and let move
        cat.move_state = True
        cat.rect.y -= setting.cloud_speed

    else:
        if cat.rect.bottom == setting.screen_height:
            # Add more points if hit ground
            stat.score += setting.touch_ground

        elif cat.rect.bottom > setting.screen_height:
            # If cat fall below screen move its bottom to base of screen
            cat.rect.bottom = setting.screen_height

        elif cat.rect.bottom < setting.screen_height:
            # Push cat down while cat's bottom above screen's height
            cat.rect.y += setting.cloud_speed
            stat.score += setting.cloud_points

            # Make cat unable to move to prevent colliding with cloud's side
            cat.move_state = False
            cat.moving_right, cat.moving_left, cat.dash = False, False, False

        # Change the score after adding
        sb.prep_score()
```

```
        check_high_score(stat, sb)

    def hit_top(setting, screen, cat, cloud, stats, sb):
        # Destroy cloud in the group after hitting top screen
        cloud_destroy = 0
        for clouds in cloud.copy():
            if clouds.rect.bottom < 0:
                cloud_destroy += 1
                cloud.remove(clouds)
                if cloud_destroy == 7:
                    setting.cloud_destroyed += 1

        if setting.cloud_destroyed / 10 == 1:

            # Max level is 3
            if stats.level == "MAX":
                pass
            elif int(stats.level) < 3:
                # Increase level
                stats.level = str(int(stats.level) + 1)

                # Increase speed
                setting.increase_speed()

            elif stats.level == "3":
                stats.level = "MAX"
            setting.cloud_destroyed = 0
            sb.prep_level()

        # If cat hit top return True which will stop (not exit) game in Run_Down.py
        if cat.rect.top <= 0:
            # Music stop
            pygame.mixer.music.fadeout(1000)

            return True

    def check_high_score(stats, sb):
        # Check to see if there's a new high score.
        if stats.score > stats.high_score:
            stats.high_score = stats.score
            sb.prep_highscore()
```

IV. Code References

- Matthes, E. (2016). *Python Crash Course*. [Place of publication not identified]: [publisher not identified].