BINUS UNIVERSITY
BINUS INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:		**Surname**	**Given Names**	**Student ID Number**
	1.	**Reden**	**Robert**	**2201816612**

Course Code	**: COMP6056**	**Course Name**	**: Program Design Methods**
Class	**: L1CC**	**Name of Lecturer(s)**	: Jude Joseph Lamug Martinez
Major	**: CS**		
Title of Assignment	: Frogger game		
Type of Assignment	**: Final Project**		
Submission Pattern			
Due Date	**: 23-11-2018**	**Submission Date**	**: 23-11-2017**

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 offense 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.

(Name of Student) Signature of Student:

1. Robert Reden

“Frogger Game”

Name : Robert Reden

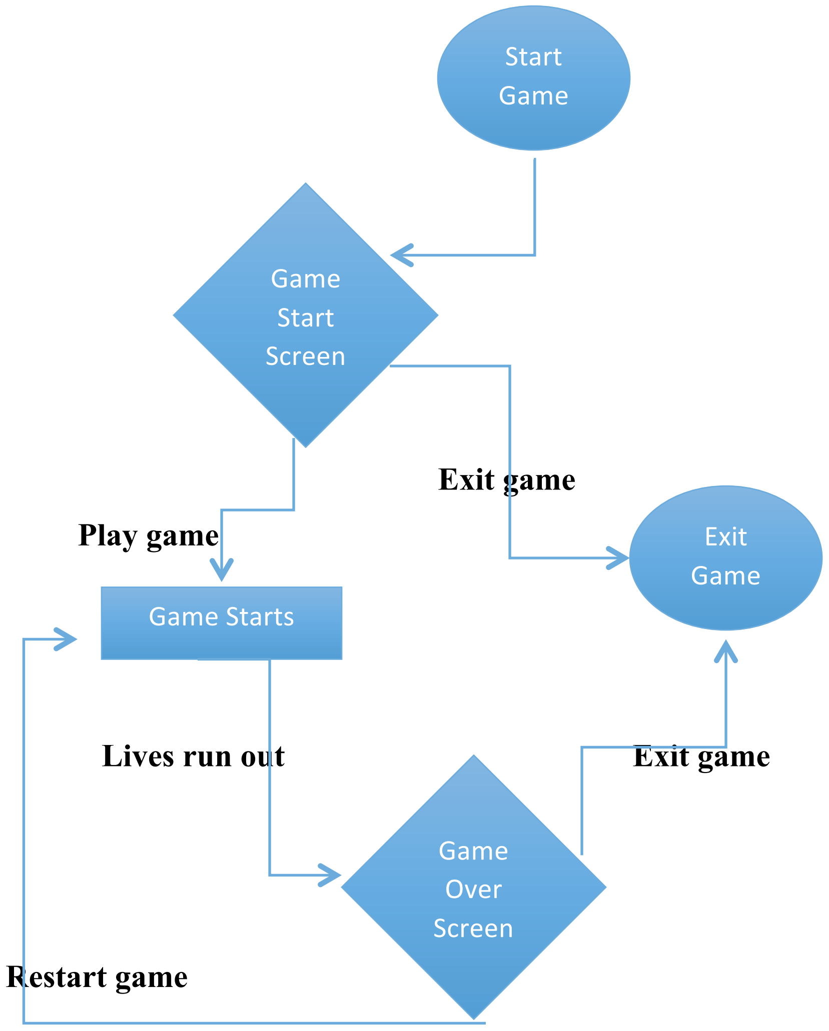
ID : 2201816612

I. Project Specification

The purpose of this program is to provide entertainment in the form of an arcade game that could be played by a maximum of two players in hopes of not only being a means to pass time but also enabling the players to bond with each other through this game. Instead of the usual frogger game, this version of the game comes with a twist as the style of the game is a competition between two players. The players will compete to see who can achieve a higher number of points before running out of lives.

This program is made using the pygame module and implements the use of primitive data, instance variables and objects, imported modules, packages, libraries, and functions, custom-built classes and methods.

II. Solution Scheme



III. Algorithm

froggergame.py

- imports pygame, Settings() class from settings.py, Player1(pygame.sprite.Sprite) class from Player1.py, Player2(pygame.sprite.Sprite) class from Player2.py, Mobs(pygame.sprite.Sprite) class from mobs.py, Floats(pygame.sprite.Sprite) class from floats.py, and Life(pygame.sprite.Sprite) class from playerlife.py
- initialize pygame using pygame.init() and mixer using pygame.mixer.init()
- create object to help track time using pygame.time.Clock() under variable "clock"
- load background music for the game using pygame.mixer.music.load() and loop it using pygame.mixer.music.play(loops=-1)
- Initializes the class "Player1()" as "bunny", "Player2()" as turtle, "Mobs()" as ("car1", "car2", "car3", "car4", "car5"), "Floats()" as ("log1", "log2", "log3", "plank1", "plank2), "Settings()" as "game_settings", "Life()" as "life_1_P1", "life_2_P1", "life_3_P1", "life_1_P2", "life_2_P2", "life_3_P2".
- create all_sprites group to contain sprites and all_mobs group to contain mobs sprite using pygame.sprite.Group().
- Sets player1 and player2 score to "0"
- Sets variable "running" and "game_over" to "True" for game loop
- Set counter to "0" for generating mobs
- Game loop
    - sprites are added to the all_sprites group at the start of the game and set players score = 0 when the game starts and when it is restarted
    - keep the loop running at the same speed with clock.tick(game_settings.fps)
    - Executes the show_score() function and add a point every time a player reaches the goal
    - Executes the generate_car() function every 100 counts by incrementing the counter by 1
    - "for" loop to get events in pygame using pygame.event.get() and "if" condition inside "for" loop to quit game loop by changing game loop variable "running" to False when event is pygame.quit().

- remove player life sprite and goes to game over screen when both players die, remove player sprite when lives left is 0.
    - check to see if a player sprite collides with a mob using pygame.sprite.spritecollide(sprite, group, dokill)
    - reset player sprite position and minus player life for collision with mobs or dropping to the river.
    - move the sprite in the same direction with the floats sprite when they collide using pygame.sprite.collide_rect(sprite, sprite) and then self.rect = self.rect.move(±game_settings.float_speed, 0)
    - updates the sprite group and draws them to screen with all_sprites.update().
    - fill screen with background image screen.blit() and flip display after everything is drawn with pygame.display.flip ().
- **draw_text(surf, text, size, x, y):**
    - function to draw text to the screen
- **generate_car():**
    - function to add (“car1”, “car2”, “car3”, “car4”, “car5”) to all_sprites group and all_mobs group using all_sprites.add()
- **show_go_screen():**
    - function to show game start and game over screen, draw command texts to screen using draw_text(surf, text, size, x, y) function, get user input to start game or quit game
- **show_score():**
    - function to draw player scores to screen using draw_text() function

Player1.py and Player2.py

- Contains class “Player1(pygame.sprite.Sprite)” for Player1.py and “Player1(pygame.sprite.Sprite)” for Player2.py that has two functions: __init__(self) and update(self).
- Imports pygame and Settings() class from settings.py
- Initialize “Settings()” class as game_settings.

__init__(self):

- calls __init__(self) from Sprite class in pygame using pygame.sprite.Sprite.__init__(self)
- loads player sprite image using pygame.image.load()
- position sprite image when game starts using self.rect.center = (x, y)
- contain sprite movement dictionary and sprite movement counts
- initialize player lives
- initialize timer to 0

update(self):

- initialize player horizontal and vertical speed
- move player sprite according to speed when movement keys are pressed using pygame.key.get_pressed() as a variable “keystate” and using “if” conditions.
- loads a different image for player sprite for every “x” number of keys pressed by multiplying the sprite movement count by “-1” therefore getting a different image to load from the sprite movement dictionary.
- disable the player sprite to move pass the edge of the screen by setting self.rect.”side” equal to coordinates of edge of screen when player sprite tries to move pass it.
- increment timer

settings.py

- import pygame
- contains class "Settings()" that has an __init__() function

__init__(self):

- initialize screen settings such as screen width, screen height, screen background image
- initialize fps of game
- initialize speed of floats movement
- initialize background music volume

playerlife.py

- import pygame
- contains class Life(pygame.sprite.Sprite) that has an __init__() function

__init__(self, number):

- calls __init__(self) from Sprite class in pygame using pygame.sprite.Sprite.__init__(self)
- load image of player life sprite using pygame.image.load()
- position player lives at bottom corner of screen using self.rect.center = (number, 620), number is the x-coordinate of the sprite that differs for each life sprite

mobs.py

- import pygame and Settings() class from settings.py
- initialize Settings() class as game_settings
- contains class Mobs(pygame.sprite.Sprite) that has two functions: __init__() and update().

__init__(self, image, position, speed, number):

- calls __init__(self) from Sprite class in pygame using pygame.sprite.Sprite.__init__(self)
- loads mob sprite image using pygame.image.load()
- initialize mob position, speed, and number
- using "if" condition spawn mobs sprite in different lanes and from different sides of the game screen according to its number, mobs with even number spawn from the right side and mobs with odd numbers spawn from the left side. self.rect = self.rect.move(x, self.pos)

update(self):

- using "if" condition, move the mobs sprite in a direction according to its number, mobs with even numbers moves to the left and mobs with odd numbers move to the right. self.rect.move(self.speed, 0) for mobs with odd numbers and self.rect.move(self.speed * -1, 0) for mobs with even numbers.

floats.py

- import pygame and Settings() class from settings.py
- initialize Settings() class as game_settings
- contains class Floats(pygame.sprite.Sprite) that has two functions: __init__() and update().

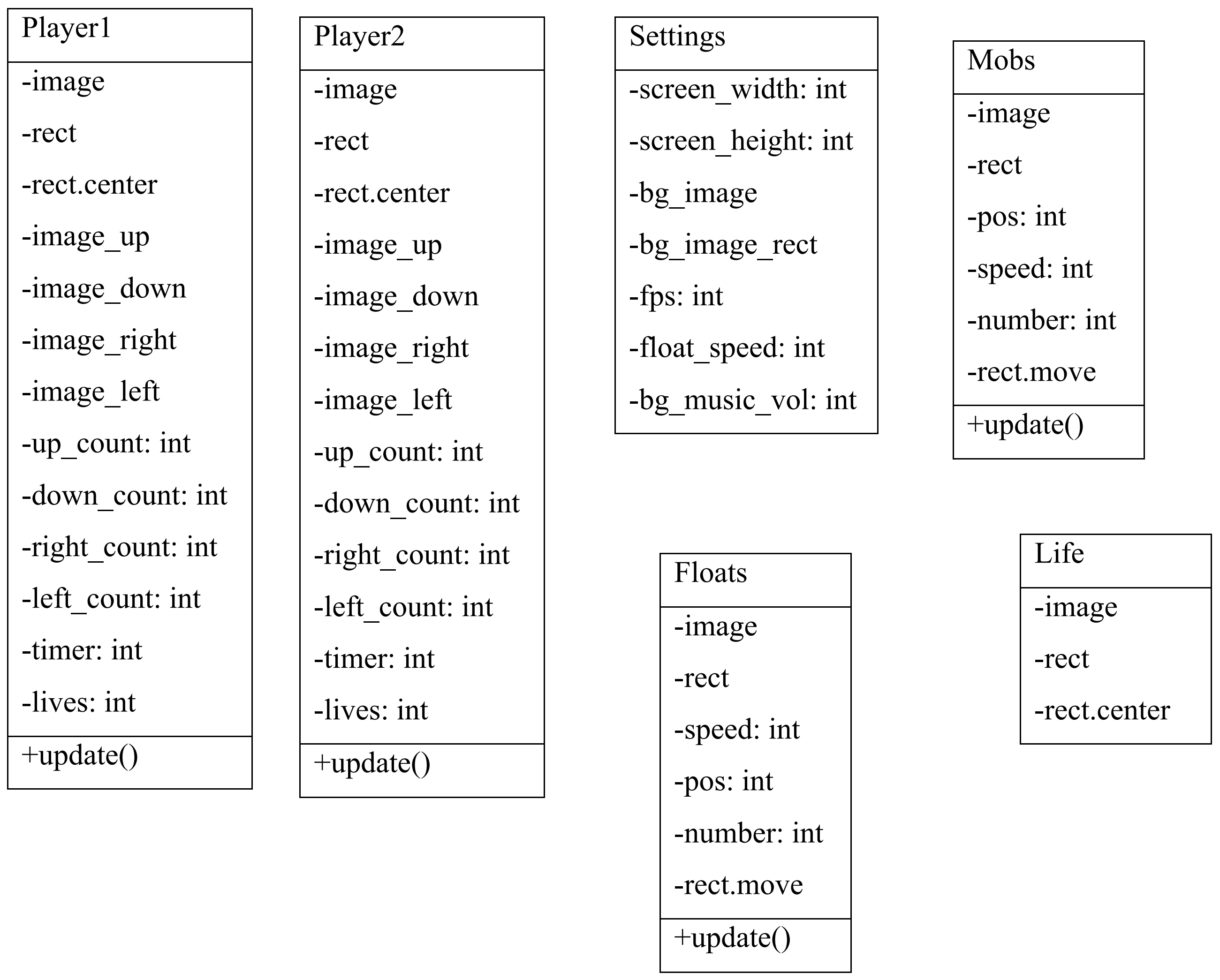
__init__(self, image, position, number):

- calls __init__(self) from Sprite class in pygame using pygame.sprite.Sprite.__init__(self)
- loads floats sprite image using pygame.image.load()
- initialize floats speed, position, and number
- using "if" condition spawn floats sprite in different lanes and from different sides of the game screen according to its number, floats with even number spawn from the right side and floats with odd numbers spawn from the left side. self.rect = self.rect.move(x, self.pos)

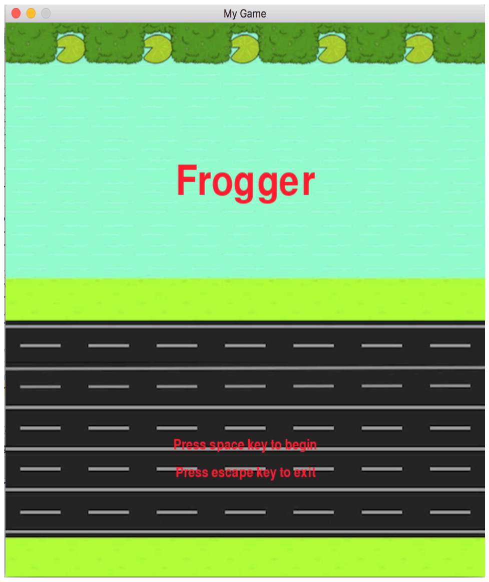
update(self):

- using "if" condition, move the floats sprite in a direction according to its number, floats with even numbers moves to the left and floats with odd numbers move to the right. self.rect.move(self.speed, 0) for floats with odd numbers and self.rect.move(self.speed * -1, 0) for floats with even numbers.

IV. Class Diagram



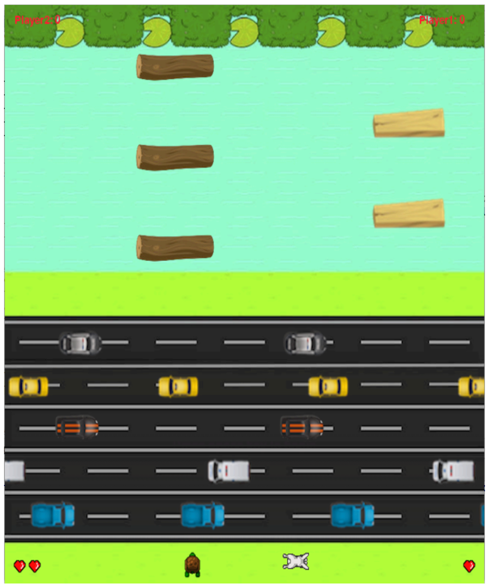
V. Program Demo

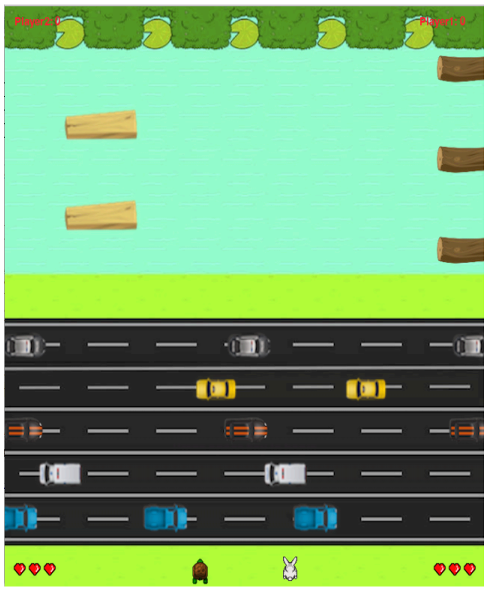


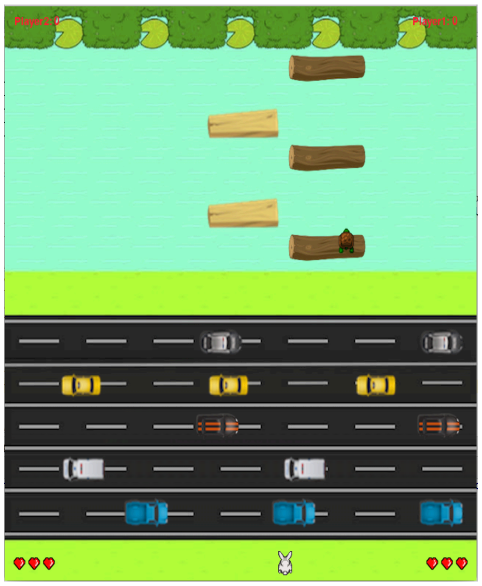
Frogger

Press space key to begin

Press escape key to exit


Player2: 0
Player1: 0


Player2: 0
Player1: 0


Player2: 0
Player1: 0