Odd Semester (2021)

BINUS UNIVERSITY

BINUS INTERNATIONAL

Final Project Cover Letter

(Individual Work)

Student Information:

Surname: Clarin **Given Name:** Maria **Student ID Number:** 2501990331

Course Code : COMP6047001 **Course Name :** Algorithm and Programming

Class : L1AC **Lecturer :** Jude Joseph Lamug Martinez, MCS

Type of Assignments: Term Final Project

Submission Pattern

Due Date **:** 17 January 2022 **Submission Date** **:** 16 January 2022

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: Maria Clarin

A. Project Specification

1. **Game Name :**
    - Mare-a-thon. The name is basically the word ‘marathon’ but I twisted the word and placed the word ‘mare’ in it which meant a female horse.
2. **Game Concept :**
    - Mare-a-thon is a horse racing game, in which the user plays as a horse jockey who is racing with another jockey that is basically a computer bot.
    - The game consists of 5 levels. In order to win the game, you need to complete every level. And to complete each level, you will need to get to the finish line first before your computer opponent does. Once you complete a level, you will level up to the next level. Once you finish the final level, level 5, you win the game. However, if you lose to the computer bot in any of the levels, you lose the game.
    - To add a sense of challenge in the game, I incorporated obstacles, boundaries, and boosts. Aside from that, the computer bot is also programmed to increase its speed every time we level up, so the bot gets faster and faster in each level. From a game I used to play when I was a child, ‘Circus Charlie’, I remembered having to avoid obstacles and it really added to the fun. So for the obstacles, I placed some poop as obstacles in the game. From the game ‘Mario Kart Racing’, I remembered rainbow lanes in which if you go over it you will accelerate faster for a short amount of time. So I incorporated that as acceleration boosts in the game. As for the boundaries, I just used basic logic. Such as not allowing the player to go over and off track of the race track, and not allowing them to simply reverse back into the finish line as finishing the game.
3. **Game Flow Summary :**
    - The player has to race against the computer bot in horse racing while trying to avoid bumping into the race track border lines and into the poop obstacles as it results in the player bouncing back and therefore losing speed and more time. However, to get a boost of speed advantage the player would want to go over the rainbow dash lanes to activate an acceleration boost. The player does this in order to get to the finish line before the opponent does.
    - This basic game flow applies to each and every level in order for the player to eventually win
4. **Game Objective :**
    - Reach the finish line before the opponent does in every level to win the game.
5. **Game Display :**
    - 2D top view, with a cartoonish style, with little bits of pixel art. (Initially I was planning to make it all pixel art style, however I don't have the resources and skill to do that for now so I just do the best I can to make it somewhat decent looking.)
6. **Game Mechanics :**
    - The player is able to move forward, backwards, stop, and turn 360 degrees while facing to the left or to the right.

7. **Game Physics :**
    - I have incorporated basic movement logic in the game. The player is given a maximum velocity. In the start of the game, the player is still, which means they had the velocity of 0. Once the player starts moving, the player's velocity will increase by the acceleration amount until it reaches its maximum velocity. When the player stops pressing the key to move forward (W key on keyboard), they will slowly reduce their speed until they have the velocity of 0 or basically until they stop moving. They can also move backwards in the same manner. As for the rotation of the player's horse, they are given a rotation velocity. Which is basically how fast you can rotate. This is so that the rotation is natural and not just flipping/rotating the character by a certain angle. When the player bumps into an obstacle/the track borders, they will receive a bounce back with the negative amount of velocity of when they initially hit the obstacle/track. When the player steps on top of the boost lanes, their acceleration will increase.
    - The computer horse however, is programmed to only move forward with its maximum velocity right from the start. I created a path for them to move and follow. They also have a rotation velocity. And I had to use trigonometry to figure out how to move the computer horse by a certain angle in the right direction of the path. In their path, the computer horse doesn't collide with any of the obstacles, tracks, or the boost lanes.
8. **Game Input :**
    - Any keyboard key - to start each levels
    - W key - move forward
    - A key - rotate left
    - S key - move backward
    - D key - rotate right
    - Mouse Left Button - quit the game
9. **Game Output :**
    - Player horse character image (can rotate and move based on the movement control)
    - Computer horse character image (will rotate and move according to the programmed path)
    - Background image (grass.png)
    - Racetrack image (racetrack.png)
    - Racetrack border image and pygame surface mask (racetrackborder.png)
    - Finish line image and pygame surface mask (finishline.png)
    - Poop obstacle images and pygame surface masks (poop.png)
    - Rainbow boost lane images and pygame surface masks (rainbow.png)
    - Initiator text ('Press any key to start! Current level is *currentlevel*')
    - Level info text
    - Time info text
    - Speed info text
    - Background music (music.mp3)
    - Track bump sound effect (bonk.mp3)

- Obstacle bump sound effect (poop.mp3)
- Boost active sound effect (wow.mp3)
- Losing sound effect (boo.mp3) - only if the player lost the game
- Winning sound effect (win.mp3) - only if the player wins the level

10. Game Libraries/Modules :

- Pygame - to make and run almost all the basics of the game.
- Math - for calculations of the movement specifics (especially for trigonometry).
- Time - to activate a timer for each levels
- Pygame Mixer - to play sounds and background music
- Sys - to properly terminate the game when called

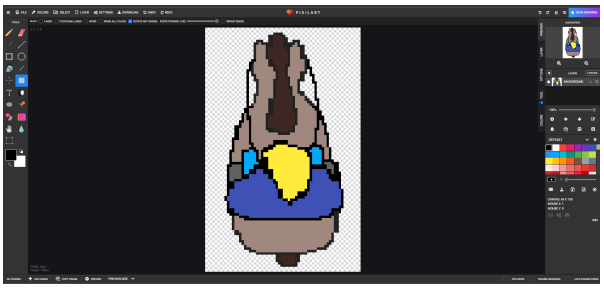
11. Game Files :

- 'documentation' folder - contains the video file and report pdf file for this project
- 'images' folder - contains all images used in the game
- 'music' folder - contains all music used in the game
- gameinfo.py - contains the class GameInfoBasics
- imagetools.py - contains functions used for some display settings of the game
- movementtools.py - contains the function for the movement control of the player character.

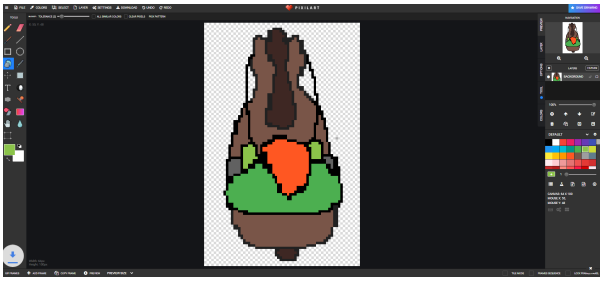
12. Game Images :

➢ Some of the images I used I made myself and the rest I altered the original image to fit the theme of the game.

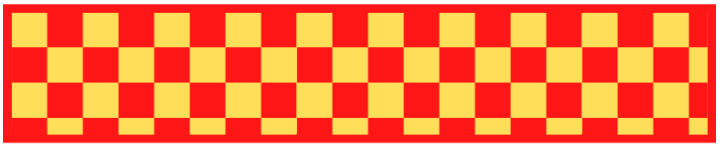
- Player horse image

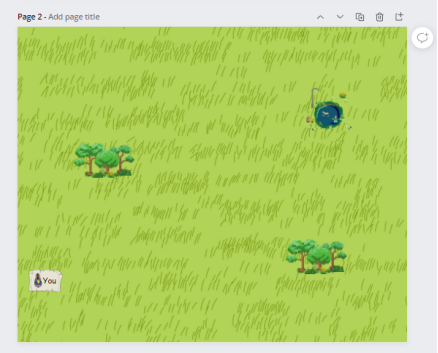
- Computer horse image

- Finish line image

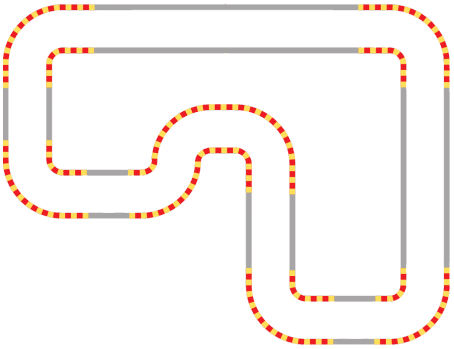


- Grass background image



- Race track image

- Race track border image



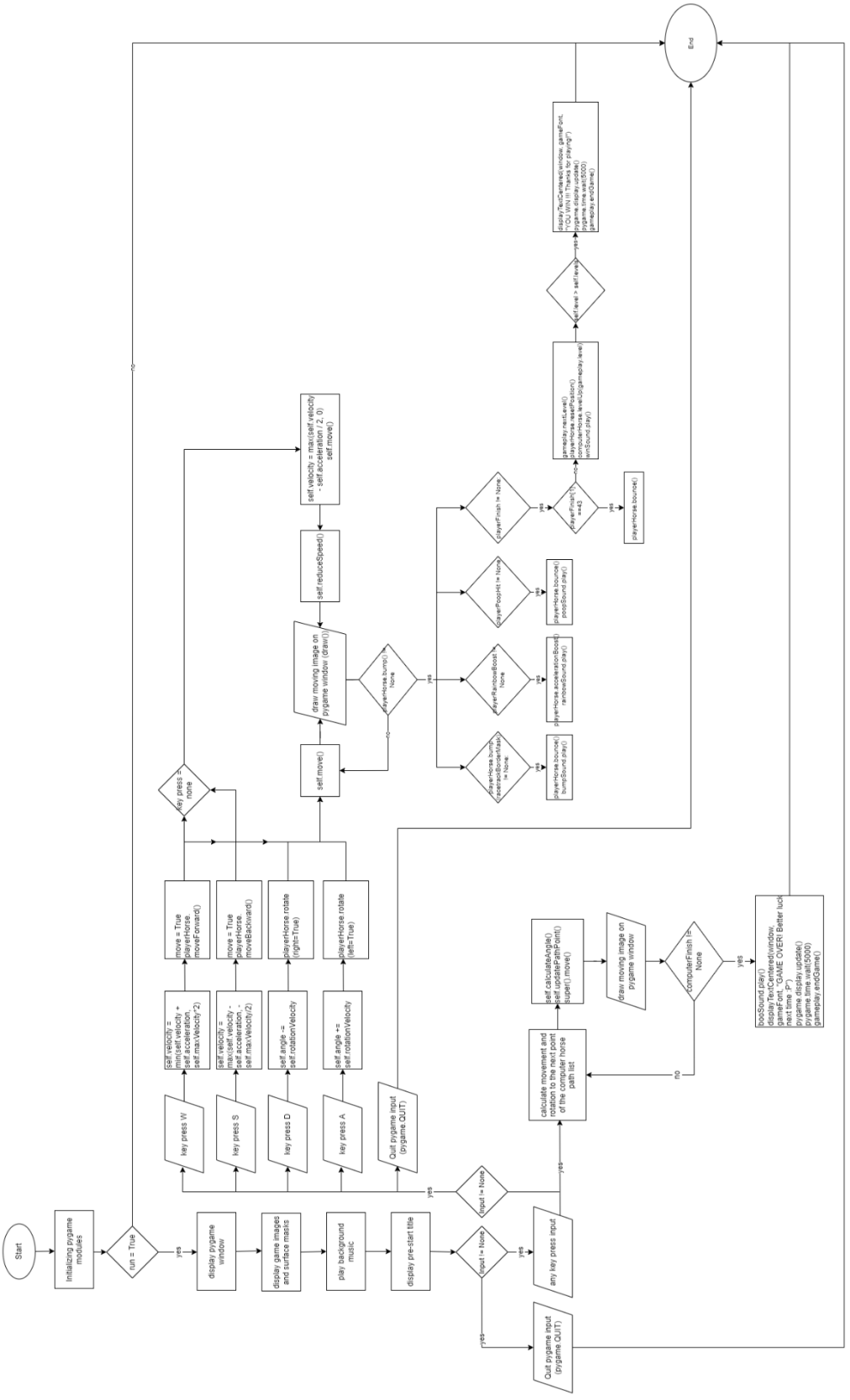
- Poop obstacle image

- Rainbow boost lane image

B. Solution Design

I. Flowchart

- For clearer view, the file for the flowchart is available in the ‘documentation’ folder.



II. Code Design and Explanation

1. Pygame Basic Setups

```
import pygame
import math
import sys
from gameinfo import GameInfoBasics
from imagetools import resizeImage, blitRotate, displayTextCentered
from movementtools import playerMovement

#initializing all pygame modules
pygame.init()
```

- All imports to the main.py file.
- Initializing the pygame module by calling pygame.init().

```
#setting up the pygame window
width, height = grass.get_width(), grass.get_height()
window = pygame.display.set_mode((width, height))
pygame.display.set_caption("Mare-a-thon")
gameFont = pygame.font.SysFont("comicsans", 30)
```

- Pygame window setup system. I used the width and height of the background image (grass) as the pygame window display size. I set the caption as the game name. I used comic sans with the size of 30 as my game font.

2. Audio and Music

```
#music settings
music = pygame.mixer.music.load("music/music.mp3")
pygame.mixer.music.play(-1) #plays background music

#load sound effects for specific uses
bumpSound = pygame.mixer.Sound("music/bonk.mp3") #sound effect whenever we bump on the track border
poopSound = pygame.mixer.Sound("music/poop.mp3") #sound effect whenever we step over the poop
rainbowSound = pygame.mixer.Sound("music/wow.mp3")  #sound effect whenever we step on the rainbow lane
booSound = pygame.mixer.Sound("music/boo.mp3") #sound effect when we lose the game
winSound = pygame.mixer.Sound("music/win.mp3") #sound effect whenever we win a level
```

- Music settings for the game. Individual sound effects that are attached to certain scenarios will be called in the if statements of the scenario.
- For the background music and sound effects, I did not make them myself. I used audios or sounds that are familiar to me that I think suits the game.
- Background music - Circus Charlie Background Music (https://www.youtube.com/watch?v=NFHvcxBCeb4)
- Bump sound effect (https://www.youtube.com/watch?v=sgCgsTTUUFs)
- Poop sound effect (https://www.youtube.com/watch?v=iWugthMcbmY)
- Rainbow sound effect (https://www.youtube.com/watch?v=BnTdfA5aTpY)
- Boo sound effect (https://www.youtube.com/watch?v=OOX1nobehhk)
- Win sound effect (https://www.youtube.com/watch?v=_Z3ra0CxCE0)

3. Image Setups

```
#loading images into the pygame window.
#resizeImage function is defined in imagetools.py
grass = resizeImage(pygame.image.load("images/grass.png"), 0.9)
racetrack = resizeImage(pygame.image.load("images/racetrack.png"), 0.9)
racetrackBorder= resizeImage(pygame.image.load("images/racetrackborder.png"), 0.9)
racetrackBorderMask = pygame.mask.from_surface(racetrackBorder)
finishline = resizeImage(pygame.image.load("images/finishline.png"), 0.139)
finishlineMask = pygame.mask.from_surface(finishline)
finishlinePosition = (16.5, 208)
poop = resizeImage(pygame.image.load("images/poop.png"), 0.3)
poopMask = pygame.mask.from_surface(poop)
poopPosition1 = (1090, 300)
poopPosition2 = (670, 600)
poopPosition3 = (30, 360)
rainbow = resizeImage(pygame.image.load("images/rainbow.png"), 0.35)
rainbowMask = pygame.mask.from_surface(rainbow)
rainbowPosition1 = (950, 890)
rainbowPosition2 = (400, 110)
rainbowPosition3 = (260, 500)
playerHorseImg = resizeImage(pygame.image.load("images/playerhorse.png"), 0.16)
computerHorseImg = resizeImage(pygame.image.load("images/computerhorse.png"), 0.16)
```

```
#creating a function to resize images by factors.
def resizeImage(img, factor):
    size = round(img.get_width()* factor), round(img.get_height()* factor) #this will give us a tuple of the new width and height.
    return pygame.transform.scale(img, size) #scaling the image in pygame.
```

- Image load setups. The resizeImage() function is used to avoid weird distortions, and to ease the whole process of resizing objects a lot faster. With the formula that returns a tuple of a rounded width and height that is multiplied by the factor we input, it returns the scaled image of the object on the pygame window screen according to the new width and height.
- All the objects that have a pygame surface mask (pygame.mask.from_surface()) for it are used for collision handling and checking.
- There are also position variables that will be used to locate the images when we call to draw the image

```
#creating a function to display images in the pygame window
def loadImage(win, images, playerHorse, computerHorse, gameplay):
    for img, position in images:
        win.blit(img, position)

    #displaying the current level info in the bottom left corner of the pygame display
    levelinfo = gameFont.render(f"Level {gameplay.level}", 1, (0,0,0))
    win.blit(levelinfo, (40, height - levelinfo.get_height() -100))

    #displaying the time info in the bottom left corner of the pygame display
    timerinfo = gameFont.render(f"Time : {gameplay.getLevelTimer()} s", 1, (0,0,0))
    win.blit(timerinfo, (40, height - timerinfo.get_height() -70))

    #displaying the speed info in the bottom left corner of the pygame display
    speedinfo = gameFont.render(f"Speed : {round(playerHorse.velocity, 1)}", 1, (0,0,0))
    win.blit(speedinfo, (40, height - speedinfo.get_height() -40))

    playerHorse.draw(window)
    computerHorse.draw(window)
    pygame.display.update()
```

- I then created a function (loadImage()) that displays all the images in the game.
- The function takes in the window we want to display, the images we want to display, the player horse image, the computer horse image, and the gameplay details.

```
#setting up the game loop to run the program
run = True
clock = pygame.time.Clock()
images = [(grass,(0,0)), (racetrack,(0,0)), (finishline, finishlinePosition), (racetrackBorder, (0,0)), (poop, poopPosition1), (poop, poopPosition2),
        (poop, poopPosition3), (rainbow, rainbowPosition1), (rainbow, rainbowPosition2), (rainbow, rainbowPosition3)]
playerHorse = Player(4, 2)
computerHorse = Computer(3.5,2, path)
gameplay = GameInfoBasics()

while run :

    clock.tick(60) #ensure the while loop cant go faster than 60 fps, so it works synchroniously if used on other computer.
    loadImage(window, images, playerHorse, computerHorse, gameplay)
```

- If you scroll to the bottom of the code file, that is where I locate all the running setups of my game. I put all the images in tuples containing the image and their position coordinates, and then I put all the tuples inside a list called images. Then I called the function loadImage() under the 'while run :' statement and just inserted the images list that I made into the function so it loads all the images in the list on the pygame window whenever the game is running.

4. Game Info Setups (gameinfo.py)

```
import time
import pygame
import sys

#setting up the game information class
class GameInfoBasics:
    levels = 5 #there are 5 levels to the game
    def __init__(self, level=1):
        self.level = level
        self.startGame = False
        self.levelStartTimer = 0

    #a function for leveling up by incrementing level by 1
    def nextLevel (self):
        self.level += 1
        self.startGame = False #game wont immediately start when we level up

    #a function that will end the game and quit the game
    def endGame(self):
        pygame.quit()
        sys.exit(0)

    #a function to finish the game when all the levels are passed.
    def finishGame(self):
        return self.level > self.levels #if the current level is more than the amount of levels there are to the game, the game is finished

    #a function to initiate the levels.
    def startLevel(self):
        self.startGame = True
        self.levelStartTimer = time.time() #to keep track when the level has started.

    #a function that returns the calculated time of the level in process
    def getLevelTimer(self):
        if not self.startGame :
            return 0 # if the game hasnt started, the timer will return 0 as the time
        return round(time.time() - self.levelStartTimer )
```

- In the file gameinfo.py, I created a class called GameInfoBasics. It contains the basic game system of the whole game. Such as the leveling system, finishing system, starting system, and timer system.
- First, I created a variable to store the amount of levels there are to the game (5).

- The __init__ method basically initializes the class with the default level of 1, with the game not starting yet (self.startGame=False) and the timer set to 0.
- Then I defined the nextLevel() function which just increments the level by 1 which will be called whenever we win a level, and when we level up, the game doesn't immediately start because the self.startGame is set to False.
- Then I created the endGame() function to terminate the game.
- The finishGame() function is just a function to check whether or not we have finished all the levels in the game by checking if our current level is more than the amount of levels there are in the game.
- The startLevel() function is the function to start the game and the game timer.
- The getLevelTimer() is a function I use to display the time info. I called it in the loadImage() function I explained in part **3. Image Setups**.

5. Basic Movement Logic (Character Class)

```
#parent class (abstract type) for the game characters (will be divided into the computer and the player)
class Character :
    def __init__ (self, maxVelocity, rotationVelocity):
        self.img = self.image
        self.maxVelocity = maxVelocity
        self.velocity = 0
        self.rotationVelocity = rotationVelocity
        self.angle = 0
        self.x, self.y = self.startingPosition
        self.acceleration = 0.05
```

- I created an abstract class that will function as the parent class for both the player class and the computer class.
- The __init__ function takes in the parameter self, the maximum velocity of the character, and the rotation velocity of the character.
- The starting velocity and the angle will both be 0 in the start. With the acceleration being 0.05 (created for the player character's movement logic).

```
#child class for the player character (concrete type) that inherits the parent class
class Player(Character):
    image = playerHorseImg
    startingPosition = (80,210)

#child class for the computer character (concrete type) that inherits the parent class
class Computer(Character):
    image = computerHorseImg
    startingPosition = (40,210)
```

- The image and startingPosition are variables that will be defined inside the player class and computer class individually. The image variable being the image of the character and the startingPosition variable being the tuple containing the x and y coordinate of the character's starting point.

```
#a function to rotate the image of the character to be able to move and turn right and left
def rotate(self, left = False, right = False) :
    if left :
        self.angle += self.rotationVelocity
    elif right :
        self.angle -= self.rotationVelocity

#a function that displays the rotated image of the character
#blitRotate is defined in the imagetools.py file
def draw(self, win):
    blitRotate(win, self.img, (self.x, self.y), self.angle )
```

- Inside the Character class, I defined multiple functions in charge of the basic logic for the character's movement.
- The rotate() function is the function that allows the player to rotate their character to a certain angle to the right or to the left according to its rotation velocity.
- The draw() function is the function that displays the rotated image of the character on the current position and the current angle. It utilizes the function called blitRotate() that I defined in the imagetools.py file

```
#creating a function that will return a rotated image of an image based off of an angle
def blitRotate (win, image, top_Left, angle):
    rotatedImage = pygame.transform.rotate(image,angle)
    nRectangle = rotatedImage.get_rect(center=image.get_rect(topleft=top_Left).center)
    win.blit(rotatedImage,nRectangle.topleft)
```

- The blitRotate() function works by returning the rotated image of the image called, based on the angle that is inputted. The way images in pygame work is that every image is basically a rectangle, even when it has a transparent background. If we just rotate an image based on an angle, there is a possibility that the image might experience weird distortions because of it being a rectangle in pygame that we may not see fully. So in order to avoid those distortions, the function rotates the image without changing the x and y coords of the image on the screen.

```
#a function that allows the character to move forward
def moveForward(self):
    self.velocity = min(self.velocity + self.acceleration, self.maxVelocity*2)
    self.move()

#a function that allows the character to move backward
def moveBackward(self):
    self.velocity = max(self.velocity - self.acceleration, -self.maxVelocity/2)
    self.move()

#a function that allows the character to move according to the x and y grid on the display of the pygame window
def move(self):
    radians = math.radians(self.angle)
    verticalVelocity = math.cos(radians) * self.velocity
    horizontalVelocity = math.sin(radians) * self.velocity

    self.y -= verticalVelocity
    self.x -= horizontalVelocity

#a function that reduces the speed of the character when theres no movement made by the player. Stops when the velocity = 0 / the car stops
def reduceSpeed(self):
    self.velocity = max(self.velocity - self.acceleration / 2, 0)
    self.move()
```

- The moveForward() function is the function that allows the character to move forward according to its velocity.
- The moveBackward() function is the function that allows the character to move backwards according to its velocity.

- The move() function is the basis of all the movement functions. It is the function that allows the character to move at all. As it is on a pygame window, the character moves on a x and y coordinate grid. Therefore I have to differentiate the velocity into 2, vertical and horizontal. Vertical velocity for the y coordinates and horizontal velocity for the x coordinates.
- The reduceSpeed() function is the function that decreases the velocity of the character whenever there is no movement until the velocity stops at 0.

```
#a function that detects if the player bumps to any mask (used for the obstacles, boundaries, and boosts)
def bump(self, mask, x=0, y=0):
    playerMask = pygame.mask.from_surface(self.img)
    offset = (int(self.x - x), int(self.y - y))
    bumpPoint = mask.overlap(playerMask, offset)
    return bumpPoint

#a function that allows the character to activate an acceleration boost by incrementing the self.acceleration
def accelerationBoost(self):
    self.acceleration += 0.1
    self.move()

#a function that will bounce back the character with the negative amount of velocity the character has at that time
def bounce(self):
    self.velocity = -self.velocity
    self.move()
```

- The bump() function works on detecting whether the player character mask has bumped/collided with any other masks. This function will return a bumpPoint. This function will be used for obstacles, boundaries, and boosts setup.
- The accelerationBoost() function works on activating an acceleration boost by incrementing the acceleration of the player character by 0.1 when active.
- The bounce() function works by bouncing back the character with the negative amount of velocity (which means they'll go the opposite direction with the same amount of velocity) the character had in the initial hit.

```
#a function that resets the player character to the starting condition
def resetPosition(self):
    self.x, self.y = self.startingPosition
    self.angle = 0
    self.velocity = 0
```

- The resetPosition() function will reset the character back to its starting condition. This function will be called when the player reaches the finish line, as they will now proceed to start the game again in the next level.

6. Player Character (Player Class)

```
#child class for the player character (concrete type) that inherits the parent class
class Player(Character):
    image = playerHorseImg
    startingPosition = (80,210)
```

- The Player class inherits all the functions from the Character class.

7. Computer Character (Computer Class)

```
#child class for the computer character (concrete type) that inherits the parent class
class Computer(Character):
    image = computerHorseImg
    startingPosition = (40,210)

    def __init__(self, maxVelocity, rotationVelocity, path =[]):
        super().__init__(maxVelocity, rotationVelocity)
        self.path = path
        self.currentPoint = 0
        self.velocity = maxVelocity
```

- The Computer class also inherits from the Character class.
- The __init__ method takes in self, a maximum velocity, a rotation velocity, and the movement path list for the computer character to move along to.

```
#function tools i use to find the pathway coordinates for the computer character
def drawPoints(self, win):
    for point in self.path:
        pygame.draw.circle(win, (255, 0, 0), point, 5)

def draw(self, win):
    super().draw(win)
    self.drawPoints(win) #this function draws the dots indicating the path of the computer horse
```

```
while run :

    clock.tick(60)
    loadImage(window, images, playerHorse, computerHorse, gameplay)

    #a while loop that is active when the level hasn't started
    while not gameplay.startGame:
        displayTextCentered(window, gameFont, f"Press any key to start! Current level is {gameplay.level}")
        pygame.display.update()
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit(0)
            if event.type == pygame.KEYDOWN:
                gameplay.startLevel()

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
            pygame.quit()
            sys.exit(0)

        if event.type == pygame.MOUSEBUTTONDOWN:
            pos = pygame.mouse.get_pos()
            computerHorse.path.append(pos)
```

```
print(computerHorse.path)
```

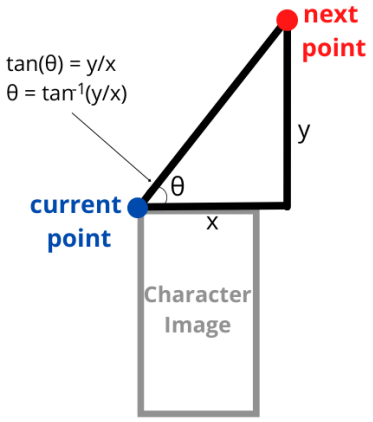
```
#a list containing tuples of coordinates for the computer car to use as a route/path
path = [(55, 139.360), (273, 75), (714, 76), (1040, 77), (1150, 234) , (1150, 419),
 (1114, 679), (1092, 882), (735, 824), (723, 500), (673, 369), (484, 383), (396, 538), (102, 535), (30,0)]
```

- These are the function tools I used to create the path for the computer character to move along to.

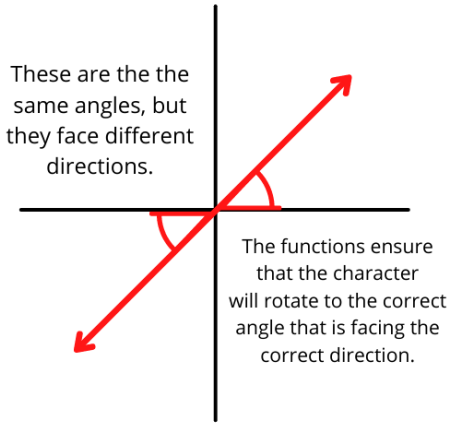
- The drawPoints() and draw() function are both located in the Computer class. They work by allowing me to red draw dots on the pygame window to indicate the path of the computer horse.
- The if statement under the 'while run:' condition that I highlighted, works by appending the red dots I drew whenever I clicked on the screen to the list of the coordinates in the path that the character will follow. I then print out the path to just copy and paste the list and declare it as the path variable.

```
#a function that calculates the angle for our computer horse to move to in the direction of.
def calculateAngle(self):
    desiredX, desiredY = self.path[self.currentPoint]
    xDifference = desiredX - self.x
    yDifference = desiredY - self.y

    if yDifference == 0:
        desiredRadianAngle = math.pi/2
    else:
        desiredRadianAngle = math.atan(xDifference/yDifference)

    if desiredY > self.y:
        desiredRadianAngle += math.pi

    angleDifference = self.angle = math.degrees(desiredRadianAngle)
    if angleDifference >= 180:
        angleDifference -= 360

    if angleDifference > 0:
        self.angle -= min(self.rotationVelocity, abs(angleDifference))
    else:
        self.angle += min(self.rotationVelocity, abs(angleDifference))
```

- This function is probably the trickiest one to understand, as it incorporates the concept of trigonometry.
- Moving the computer character isn't as easy and simple as making it move to a point while facing a certain angle, if we do that, it would just be as if the computer character teleported or flipped directions, which doesn't look natural at all.
- So in order for the computer character to move on an angle to a point while facing a certain direction, this function was created.
- The first 3 lines under the function def works to calculate the displacement of the x and y coordinates between the next point and the current computer horse position. We have to find the difference from both the x coordinate and y coordinate of the next point and the current position of the computer character.
- The first if else statement works on calculating the actual trigonometry formula of finding an acute angle. The if statement works to avoid zero division errors by making the theta of the angle as 90 degrees. The else statement works on finding the angle of the points by the formula (image attached on the right)



- The second if statement works if the next point is lower on the screen/lower on the y coordinates of the screen, which means the computer character is moving forward facing down. This if statement works on making sure that the image is properly rotated so the character is actually facing down by adding 180 degrees to the theta angle (desiredRadianAngle).
- The third statement works on finding the difference of angles of the current angle and the desired angle to move to and based on whether the number is negative or positive, the computer character will move right or left. If the angle difference calculated is larger than 180 degrees we have to decrease the angle by 360 to make sure it is going in the right direction.



- The fourth and last if else statement works to check and avoid potentially passing over the desired angle if in an instance our rotation velocity is bigger than the angle. So the function will determine the minimum number between the rotation velocity and the absolute number of the angle difference. And depending whether it goes right or left, it will increment or decrement the angle.

```
#a function that checks whether or not we move to the next point when we collide with the current point.
def updatePathPoint(self):
    target = self.path[self.currentPoint]
    rectangle = pygame.Rect(self.x, self.y, self.img.get_width(), self.img.get_height())
    if rectangle.collidepoint(*target):
        self.currentPoint += 1
```

- The updatePathPoint() is a function that will check if we have a next point to move to, and if we move to a point, it will update the current point.

```
#a function to allow the computer horse to move to a specific point
def move(self):
    if self.currentPoint >= len(self.path):
        return

    self.calculateAngle()
    self.updatePathPoint()
    super().move()
```

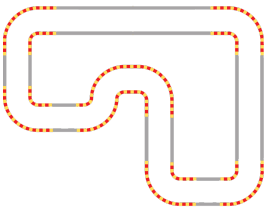
- The computer class inherits the move() function from the parent class. The move() function for the computer character basically allows the character to move to a specific point by checking and ensuring that there is still a point in the path list to move to. This function also helps avoid errors of the computer character moving to a non existent point.
- This function will call the previous functions to calculate the angle to allow the computer character to shift and rotate according to the correct angle and direction, and to update the current path point.

```
#a function used to set level settings for the computer horse for everytime we level up
def levelUp(self, level):
    self.resetPosition()
    self.velocity = self.maxVelocity + (level-1) * 0.5
    self.currentPoint = 0
```

- The levelUp() function is in charge of setting up the level settings for the computer character every time we level up. So when we level up, the computer character will reset its position and current point to 0. Every time we level up, the computer character will also be faster by 0.5 multiplied by the current level -1.

8. Obstacles, Boundaries, and Boosts Setups

```
#a function to take care of every bumps/boost activation
def bumpsAndBoosts(playerHorse, computerHorse, gameplay):
    #if the player hits/bumps into the racetrack borders
    if playerHorse.bump(racetrackBorderMask) != None:
        playerHorse.bounce()
        bumpSound.play()
```



- I defined a function to handle obstacles, boundaries and boosts. I manually removed the background and the main track road from the race track image to get the race track border with canva.
- The first if statement is for the scenario where the player bumps into the racetrack borders. The function .bounce() that I explained earlier in part **5. Basic Movement Logic** will work whenever this if statement is fulfilled where the player will be bounced back with the negative amount of velocity as the initial hit, making them move the opposite direction with the same amount of velocity.

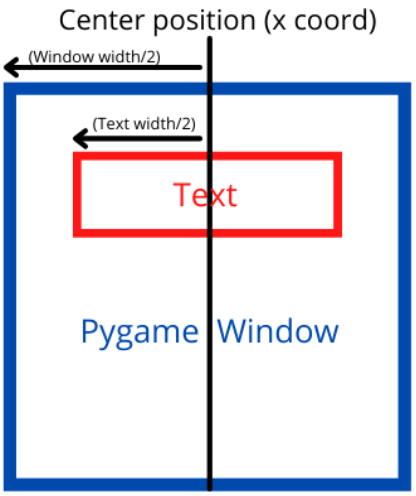
```
#Finishline setups
#1. If player finishes first
playerFinish= playerHorse.bump(finishlineMask, *finishlinePosition)
if playerFinish != None:
    # print(playerFinish)
    if playerFinish[1] == 43:
        playerHorse.bounce()
    else:
        gameplay.nextLevel()
        playerHorse.resetPosition()
        computerHorse.levelUp(gameplay.level)
        winSound.play()
# 2. If the computer finishes first
computerFinish = computerHorse.bump(finishlineMask, *finishlinePosition)
if computerFinish != None:
    booSound.play()
    displayTextCentered(window, gameFont, "GAME OVER! Better luck next time :P")
    pygame.display.update()
    pygame.time.wait(5000)
    gameplay.endGame()
```

- Inside the function, I also set up the finish line setups. For the finish line image, I actually made it myself with canva and it looks like this :D

- The first condition is if the player finishes first. The first if statement works if the player bumped into the finish line. The nested if statement under it is used to make sure the player is unable to reverse into the finish line as that doesn't count as finishing the race. The number 43 is the top y coordinate of the finish line. So if they would try to reverse, they would just bounce back. The nested else statement works if

the player actually wins the current level by reaching the finish line before the computer character does. The function will then proceed to the next level, reset the player's position, level up the computer horse according to the level setups, and play the winning sound effect.

- Second condition is if the computer finishes first, which means the player loses the game. If the computer hits the finish line before the player does, the boo sound effect will play, and a text saying 'GAME OVER! Better luck next time :P' will be displayed. The pygame window will wait for 5 seconds (5000 milliseconds) and after, would end the game and quit the window.



To find the center position for the text, we just use the formula : (window width/2) - (text width/2)

```
#a function to blit/display a text
def displayTextCentered(win, font, text):
    render = font.render(text, 1, (0, 0, 0))
    win.blit(render, (win.get_width()/2 - render.get_width()/2, 220))
```

- The text display is loaded with this function located in the imagetools.py file. It will display the text in the center of the screen with the y coordinate of 220 and the x coordinate is found with the following formula (image on the right).

```
#Obstacle Setups
playerPoopHit= playerHorse.bump(poopMask, *poopPosition1)
if playerPoopHit != None:
    playerHorse.bounce()    #setup for position 1 of the poop
    poopSound.play()
playerPoopHit= playerHorse.bump(poopMask, *poopPosition2)
if playerPoopHit != None:
    playerHorse.bounce()    #setup for position 2 of the poop
    poopSound.play()
playerPoopHit= playerHorse.bump(poopMask, *poopPosition3)
if playerPoopHit != None:
    playerHorse.bounce()    #setup for position 3 of the poop
    poopSound.play()
```

- In the function, there is also the obstacle setup with the obstacle being the poop. I actually made the image of the poop myself with pixelart.com.
- So every time the player bumps into the poop, they will bounce back and the poop sound effect will play. I repeated the if statements for every position of the poop.

```
#Boost Setups
#if the player steps on the rainbow dash square, they will receive an acceleration boost.
playerRainbowBoost= playerHorse.bump(rainbowMask, *rainbowPosition1)
if playerRainbowBoost != None:
    playerHorse.accelerationBoost() #setup for position 1 of the boost
    rainbowSound.play()
playerRainbowBoost= playerHorse.bump(rainbowMask, *rainbowPosition2)
if playerRainbowBoost != None:
    playerHorse.accelerationBoost() #setup for position 2 of the boost
    rainbowSound.play()
playerRainbowBoost= playerHorse.bump(rainbowMask, *rainbowPosition3)
if playerRainbowBoost != None:
    playerHorse.accelerationBoost() #setup for position 3 of the boost
    rainbowSound.play()
```

- The function also handles boost setups. I designed the rainbow boost lanes myself with canva.
- The if statements checks whether or not the player's character goes over the rainbow lane images. If the player character does go over the rainbow, they will receive the acceleration boost and the rainbow sound effect will play.

9. Movement Inputs (movementtools.py)

```
import pygame

def playerMovement(playerHorse):
    #setting up keyboard keys to be used in game to do actions
    keyboardKeys = pygame.key.get_pressed()
    move = False

    #a and d key on keyboard will work on rotating the character to make turns
    if keyboardKeys[pygame.K_a]:
        playerHorse.rotate(left=True)
    if keyboardKeys[pygame.K_d]:
        playerHorse.rotate(right=True)

    #w and s key on keyboard will work on moving the character forward and backward
    if keyboardKeys[pygame.K_w]:
        move = True
        playerHorse.moveForward()
    if keyboardKeys[pygame.K_s]:
        move = True
        playerHorse.moveBackward()

    #if the player character stops moving the character will slowly reduce its speed (till it stop)
    if not move :
        playerHorse.reduceSpeed()
```

- In the file movementtools.py, I defined a function for the movement control inputs of the player character. The WASD keys will work as the main movement input keys for the player to move around. The W key works on moving the character forward. The S key works on moving the character backwards. The A key works on rotating the character to the left. And the D key works on rotating the character to the right. And if the character stops moving or pressing the movement keys (W/D) the character will slowly reduce its speed until it stops.

10. The Running Condition (The Gameplay)

```
#setting up the game loop to run the program
run = True
clock = pygame.time.Clock()
images = [(grass,(0,0)), (racetrack,(0,0)), (finishline, finishlinePosition), (racetrackBorder, (0,0)), (poop, poopPosition1),
(poop, poopPosition2), (poop, poopPosition3), (rainbow, rainbowPosition1), (rainbow, rainbowPosition2), (rainbow, rainbowPosition3)]
playerHorse = Player(4, 2)
computerHorse = Computer(3.5,2, path)
gameplay = GameInfoBasics()
```

- Before writing the while loop for the game to run, I call on every class and put them as variables.

```
while run :

    clock.tick(60)
    loadImage(window, images, playerHorse, computerHorse, gameplay)
```

- I then set up the main while loop for the game to run. The clock.tick() is to ensure the while loop won't operate faster than 60 fps, so it would work synchronously if we run the program on another computer that has a faster processing system.
- I also called the loadImage() function to load every images needed to be displayed in the game window.

```
#a while loop that is active when the level hasn't started
while not gameplay.startGame:
    displayTextCentered(window, gameFont, f"Press any key to start! Current level is {gameplay.level}")
    pygame.display.update()
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit(0)
        if event.type == pygame.KEYDOWN:
            gameplay.startLevel()
```

- When the game is first launched, they enter this while loop. This while loop ensures the game doesn't start until the player presses any key down. It first displays the text "Press any key to start! Current level is *level*". If the player quits the game window before starting the game, the game would quit and end like normal. If the player presses any key down, that is when the current level will start. The game starting would mean the program now exits this while loop and enters the main while loop.

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        run = False
        pygame.quit()
        sys.exit(0)
```

- This for loop is just to ensure that if the player quits the game window when the game has started, the program will terminate and end the game.

```
playerMovement(playerHorse)
computerHorse.move()
bumpsAndBoosts(playerHorse, computerHorse, gameplay)
```

- In the main while loop, I make sure to call out all the movement logic functions I have previously explained for the player character and the computer character.

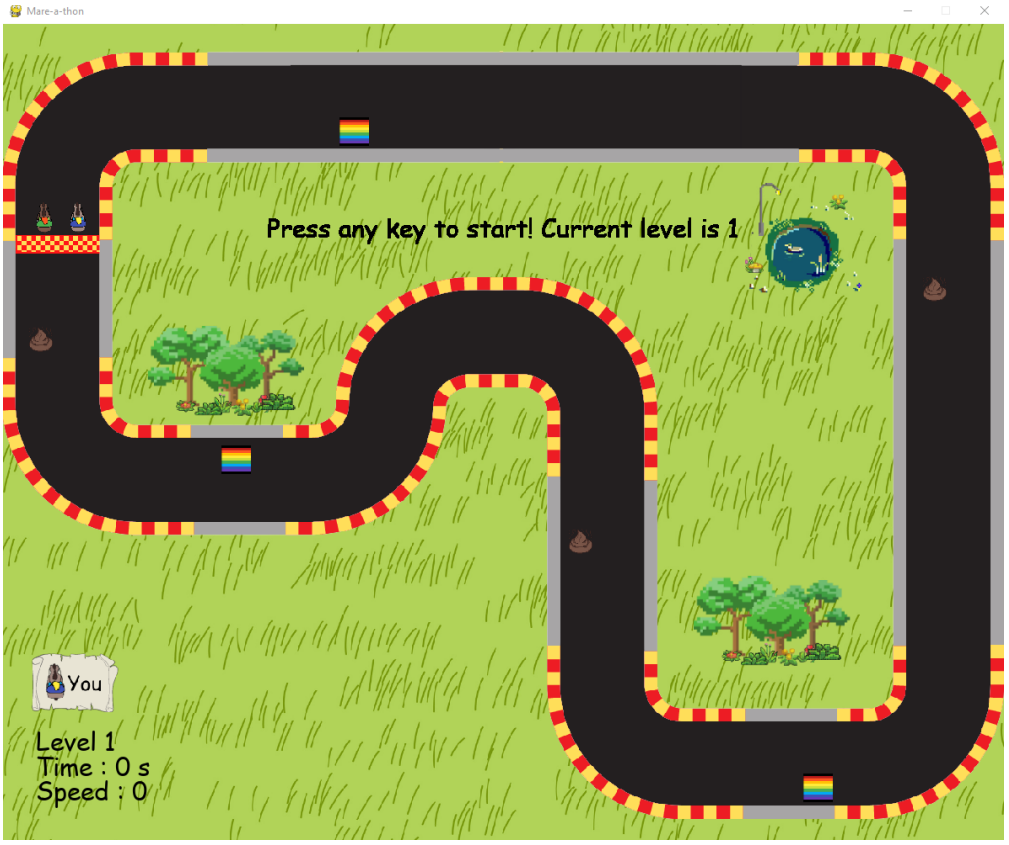
```
#if the player has completed all 5 levels, they win, so this is the if statement to check if thats the case
if gameplay.finishGame():
    displayTextCentered(window, gameFont, "YOU WIN !!! Thanks for playing!")
    pygame.display.update()
    pygame.time.wait(5000)
    gameplay.endGame()
```

- This if statement in the main while loop checks if the player has completed all 5 levels successfully. If the condition is fulfilled, the window will display a text saying “YOU WIN !!! Thanks for playing!”. The window will wait for 5 seconds (5000 milliseconds). And finally it will quit the game and terminate the window.

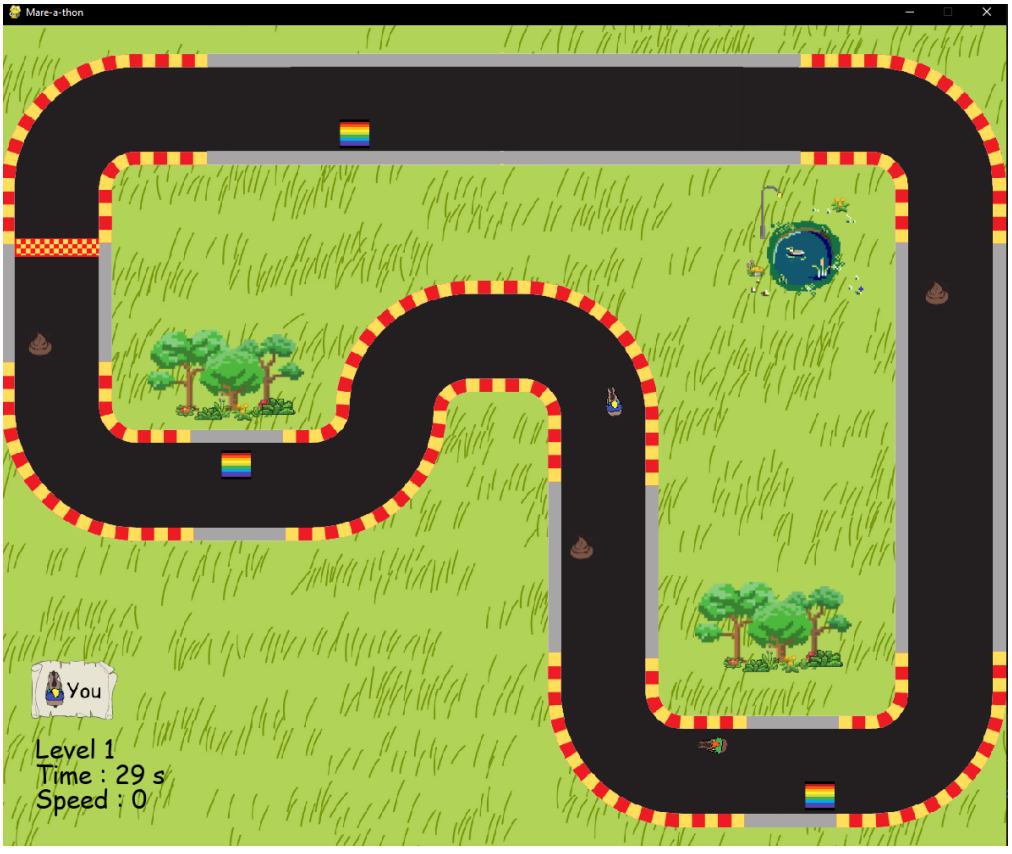
```
print("Thank you for playing Mare-a-thon")
pygame.quit()
```

- When the program is done running, it will print out the output “Thank you for playing Mare-a-thon” on the terminal and quit.

C. Game Operation Evidence

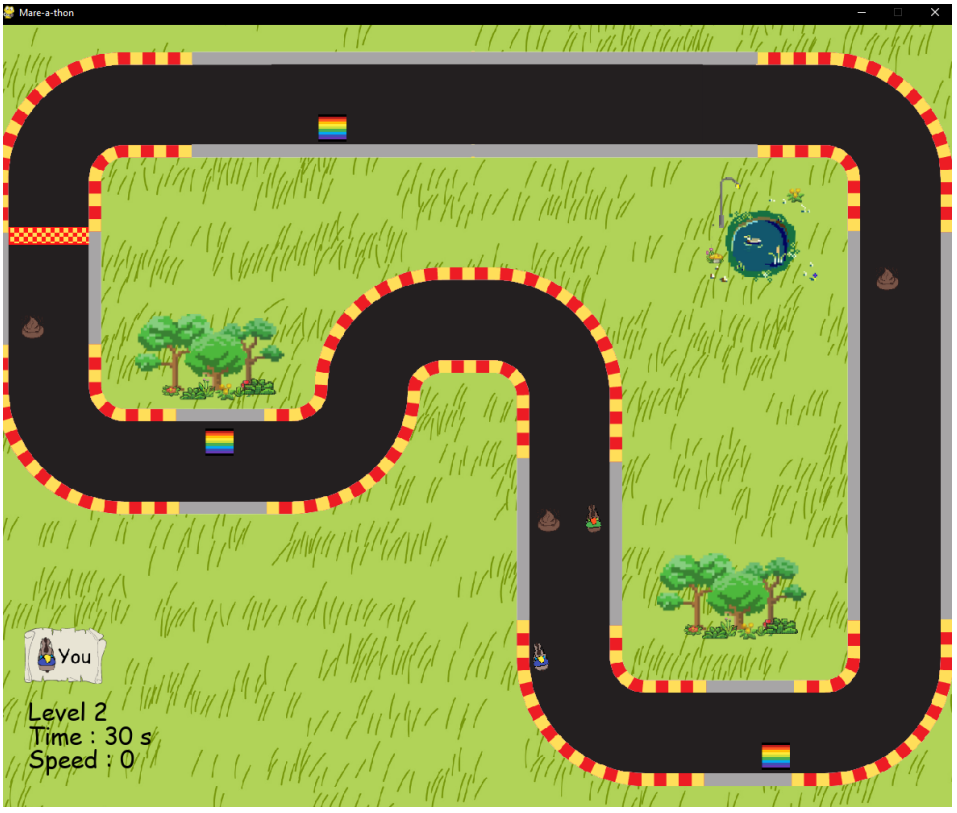


- Game initial display window

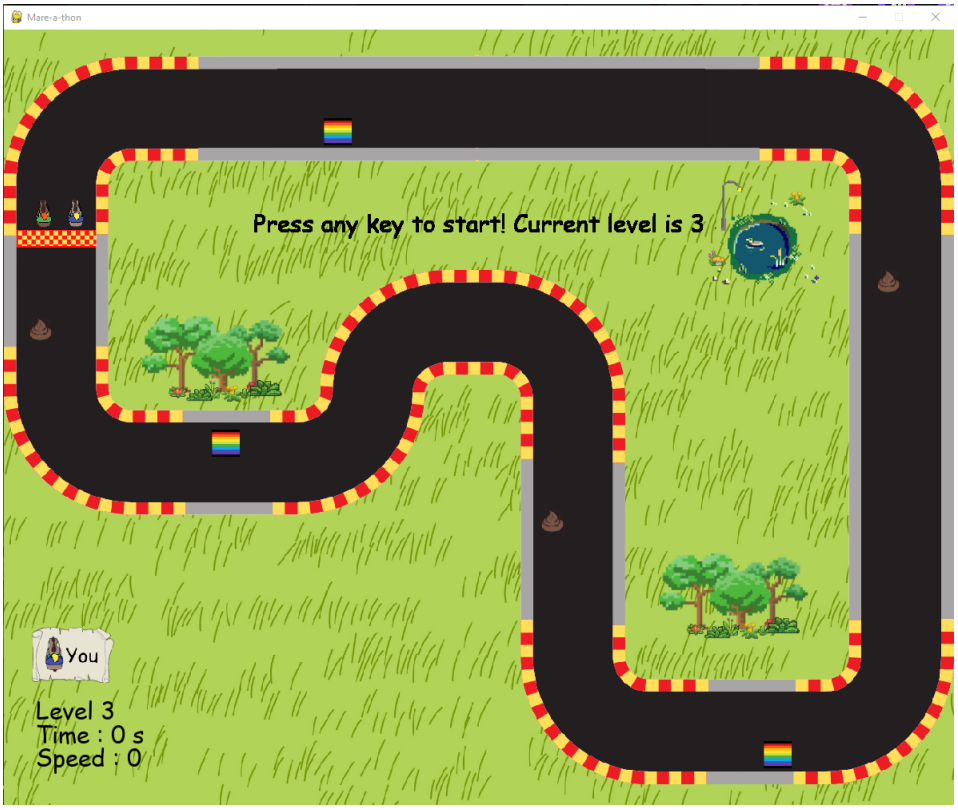


- Level 1 running game window

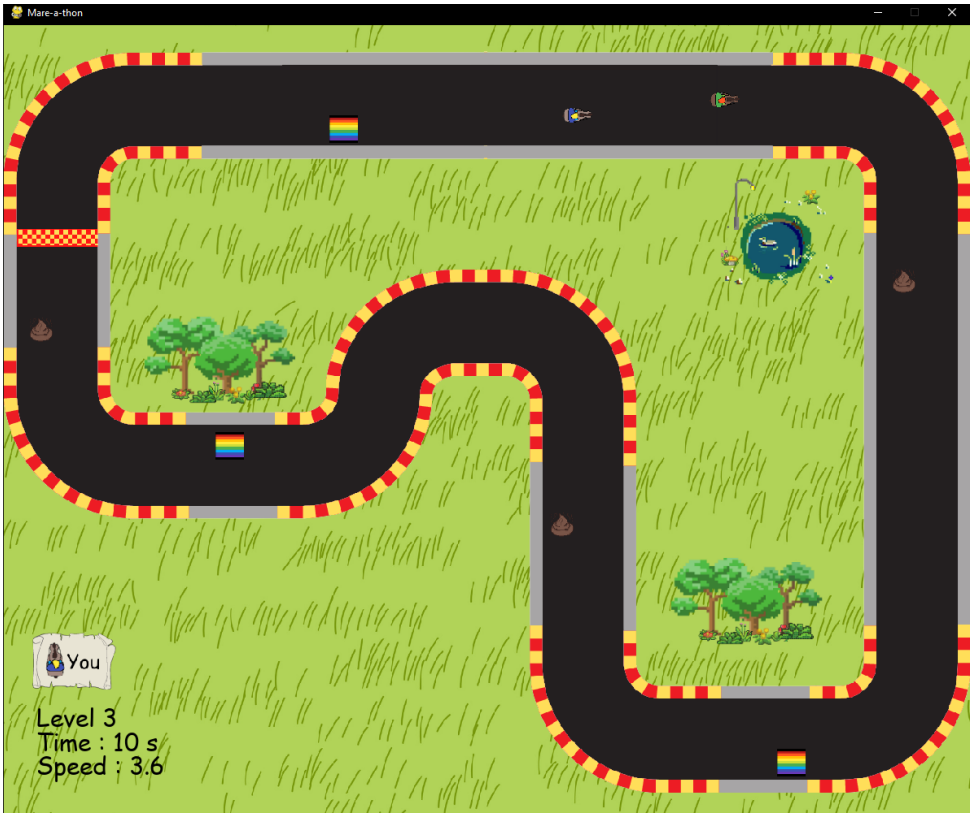
- Level 2 idle display window



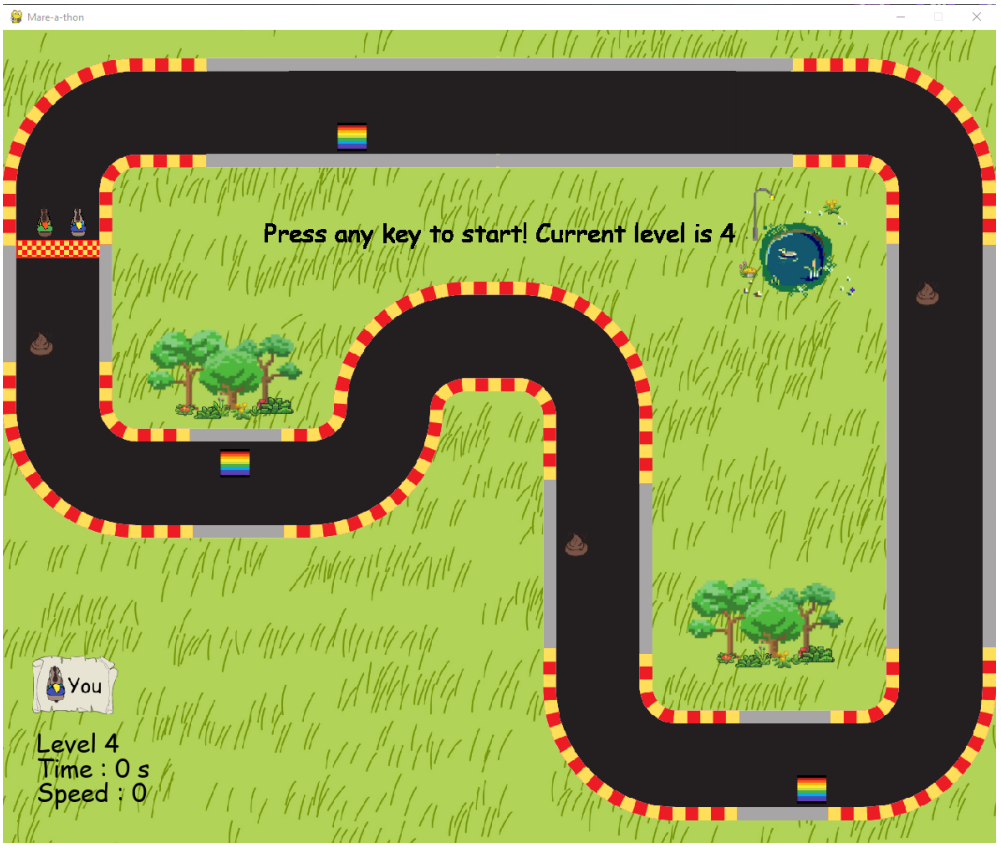
- Level 2 running game window



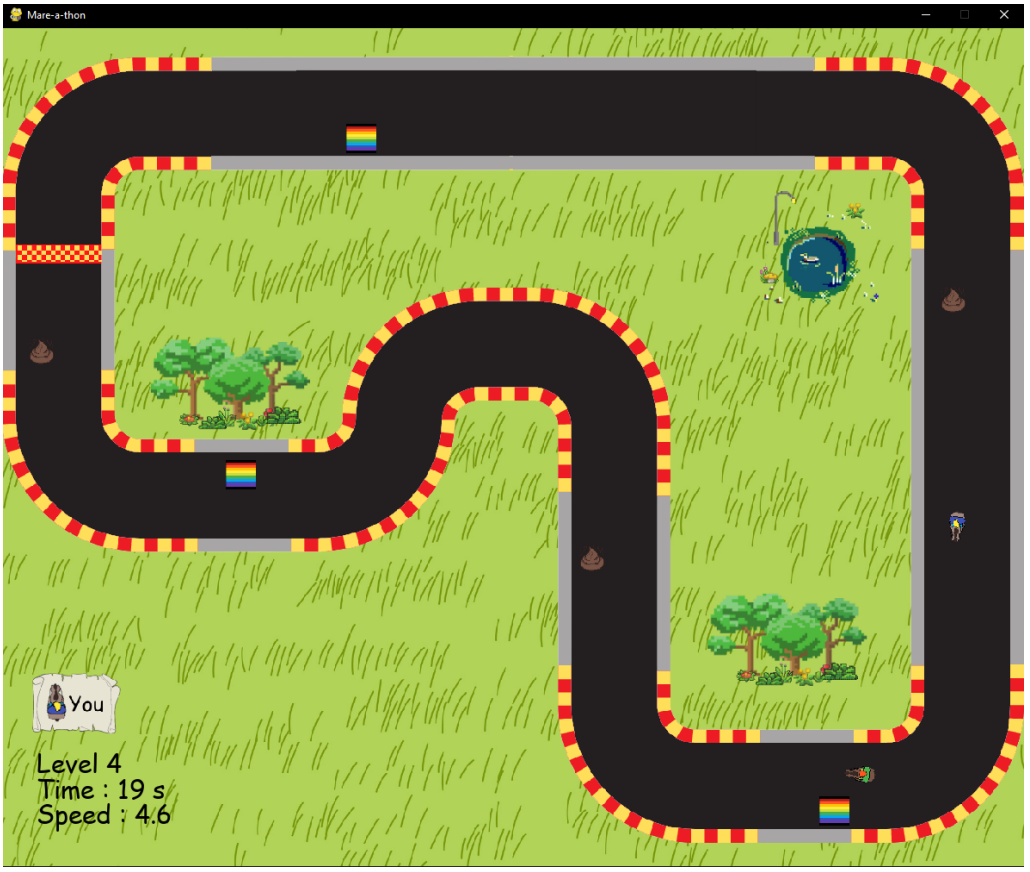
- Level 3 idle display screen



- Level 3 running game window



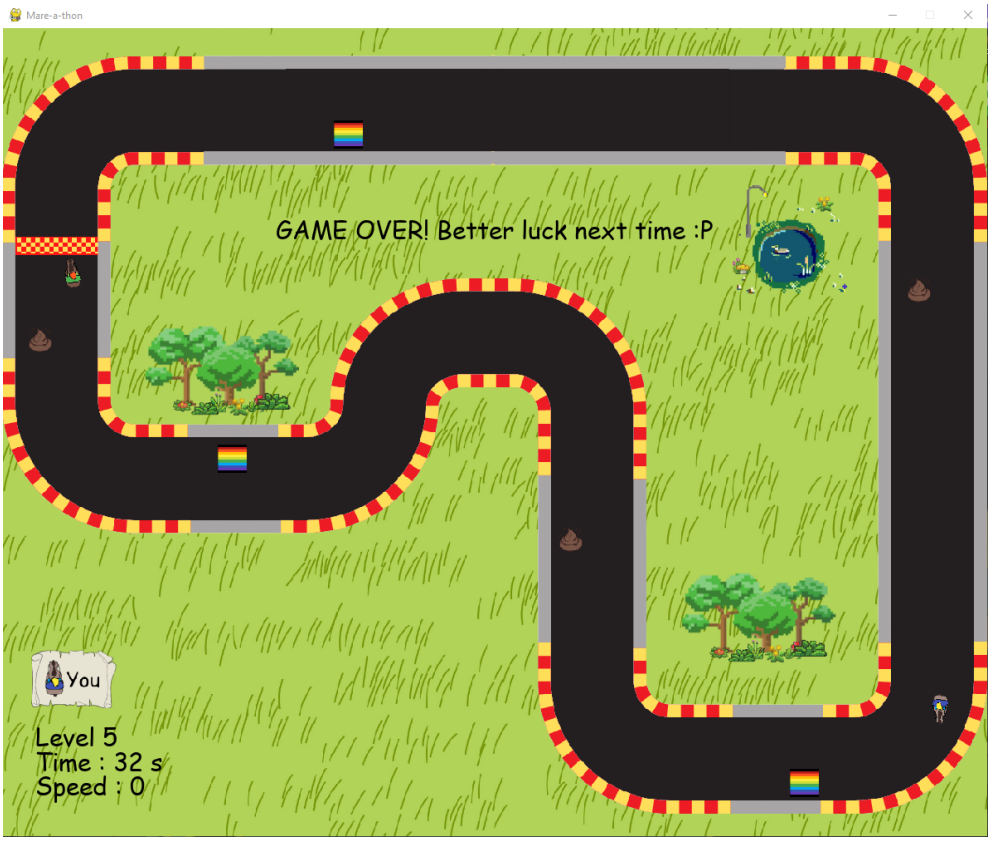
- Level 4 idle display screen



- Level 4 running game window

- Level 5 idle display screen

- Level 5 running game window



- Game over display screen

- Win display screen

D. Resources:

A. Troubleshooting

- stackoverflow.com
- geeksforgeeks.com
- titanwolf.org

B. Pygame and Racing Game

- https://www.youtube.com/channel/UC4JX40jDee_tINbkjycV4Sg
- https://www.youtube.com/channel/UC8butISFwT-Wl7EV0hUK0BQ
- https://www.youtube.com/channel/UCPrRY0S-VzekrJK7I7F4-Mg

C. Design and Images

- canva.com
- pixelart.com

D. Music and Audio

- youtube.com
- yt5s.com/en32/youtube-to-mp3
- audiotrimmer.com