NAME: UMME-HANI (SE-21006)

AFRA RASHEED (SE-21007)

SARA AZIZ (SE-21025)

DEPARTMENT: SOFTWARE ENGINEERING

SECTION: A

TABLE OF CONTENTS:

Project idea: Motorcycle Racing Game

Project Title: The Night RiderZ

Game description:

The idea behind this game is that it would keep on going until the character in the game is hit with an obstacle. The character in the game, known as the rider because it is a motorcycle racing game, has to advance, avoiding all the obstacles that come his way. The obstacles that he has to dodge are mainly the cars that will

somehow block his way, so the rider has to jump over them to move forward. For your rider to escape the maximum cars, you must be quick with your reflexes. To set a big high score, you need to quickly decide by having complete hand control over the keyboard or the spacebar to be specific. Your quick response will ensure that the rider stays in the game for a long time until hit by the car.

Details for the source code:

- Firstly we have to import pygame as it is the step that initializes all our coding that we have to do. The method pygame.init will import all the necessary pygame methods to make the process convenient. For the music to be played in the background, the method mixer is imported.
- Afterward, we will define the measurements of the screen for the game and the name of the game by calling out pygame.display.set_mode and pygame. display.set_caption methods respectively for the given operations. Also, for the customized icon that we have designed for our game, we will use the method pygame.display.set_icon by first loading the

icon using the method pygame.image.load (icon’s image preferred as png).

- Then for the visuals involved in the game, we will import some images (objects are preferred as png) and call them out in our code using the pygame.image.load method. We will define all the required images, including our player, the background, and the hurdles. For all the images, we will also define their measurement using the pygame.transform.scale that will resize and set them according to the coordinates provided.
- Then a method mixer.init would be called to import the sound for the game. By using the method mixer.music.load, the sound would be added to the code to be played in the game.
- The following method pygame.font.Font is for calling out a text file within your code.

For the frames changed in the game in a second, a variable is defined as fps, and its value is set at 60 and then using the method pygame.time.Clock, we will record its change. For the font color, we will use the predefined color coordinates RGB for the pygame (255,255,255) for the color white used for all the text in the game.

- For the variables, instruction, active, and replay will differ as per the conditions applied. By default, instructions, active, and replay are true, false, and false, respectively. Another variable will be introduced for the high scores kept at 0 at the start and will variate as the game proceeds.
- After that, the rectangular area of the cars (hurdles) is defined using the method get_rect(). We will define the coordinate at the x-axis and the y-axis separately. Then the change in the x-axis of the cars is

mentioned. After this, the coordinates at the x-axis and the y-axis of the rectangular area of each car are defined. The coordinates at the y-axis are kept constant, whereas the coordinates of the x-axis are increasing at 750 coordinate compared to the previous car. Each increment is then added to the value assigned to the first car placed as the hurdle in the game.

- After that, the coordinate at the x-axis and the y-axis of the rider are defined separately using a suitable variable. Initially, when the game hasn't started, the default change in the y-axis of the rider is kept 0, and the gravity is kept at 1. Then we will assign the variables used for the x-axis and y-axis of the rider to rect.x and rect.y respectively.
- Then we will introduce a function for the scoring purpose. We will use the method global so that modifications can be applied

to the variables defined in the main program. We will display the font for the scores using the method font.render. Then we will apply the method screen.blit to adjust the scores on the game screen as per our requirements. After this, we will be applying a conditional statement using the "if" control structure. Using this, the game's speed and the change in x variable of cars will increase by 1 if the scores, when divided by 100, leave a remainder 0.

- Then a separate function is introduced for the highest score. We will use the method global in this function as well. We will display the font for the high scores using the method font.render. After that, the method, screen.blit will be applied to draw the high scores at the screen as per the coordinates we have assigned.

- Initially, we will set the ground speed at 10. Using the method pygame.transform.scale, the ground is resized and assigned coordinates, respectively. Initially, the change in the x-axis of the ground is kept at zero.
- Then an infinite while loop is being played is introduced in the game. We have kept the variable running as true initially. After that, we will use the method timer.tick(fps) to make sure that our landscape advances at 60 frames per second. Then we use screen.blit for background, ground, and rider to draw them on the screen by mentioning the appropriate coordinates.
- After that, for loop would be run to call the method pygame.event.get. The method pygame.event.type would define what key is pressed. If event.type is equal to

pygame.quit that appears when we close the game window.

- The control structure ‘if’ would then be applied for different conditions. This would be done if event.type is equal to pygame.KEYDOWN that means we pressed a key, then again, an “if” condition would be applied. The statement “if event.key == pygame.K_SPACE” implies that if the space bar is pressed. The instructions are being displayed on the screen, that means it is defined as true, and active and replay are false then after this instruction would disappear that means the condition for the instructions in false and the game will start that means the condition for the active in now true. Active basically refers to when the game starts.
- The second “if” conditional statement implies when the space bar is pressed while

running the game. In this situation, the condition for instruction and replay are false. The change in the y-axis of the rider is predefined as zero. But after the space bar is pressed, the change in the y-axis of the rider would be 25, as mentioned, which means that it would jump.

- If the game has ended, that means the condition for the replay is true. If the space bar is pressed, it would be the condition for replay to become false and for the instructions to become true. This would cause the instruction to be displayed on the screen again.
- After that, the statements are shown for the instructions. For the font of the instructions displayed, we have used the method font.render again and have defined its coordinates as per the requirements that

where do we want our instructions to be displayed on the screen.

- After that, another “if” statement is run if the condition for active is true. That would imply certain changes in the values of rider_rect.y and riderY_change.
- The x-coordinate of the ground would decrease by the rate of ground speed. The if condition applied for the x-coordinate of the ground implies that if it is getting lesser than or has become equal to -1280 that is the size defined for x-coordinate of the ground previously, then the value for x-coordinate of ground would automatically become zero.
- Then the decrease in the x-coordinate of each car is defined that is equal to the change in the x-coordinate of cars. Then the method screen.blit is used to draw these cars as per the coordinates assigned on the game screen.

- Then by using the control structure “if” and the method rider_rect . colliderect, the collision between the rider and each car is defined. If the rider has collided with any of the cars, then it would cause the game to stop. This implies that the condition for active has become false and the condition for replay to become true.
- After this, the “if” condition is applied when the x-rectangular component of each car has become equal to or more than -4000. If that’s the case, then it means the value for the x-rectangular component becomes 800, which means they will start repeating in a certain order.
- After this, the “if” condition is applied if the replay is true. If that’s the case, then this means the game has stopped and will be played again. For this, the font.render method would be called to display the text

"you crashed" and screen.blit to display the text at the appropriate coordinates. The text for the continuation of the game would also be called in the same way by using font.render and screen.blit.

- Another "if" condition is used so that when the score becomes equal to the high score, it would automatically turn the high score into the score.
- After this, again, the conditions for the measurements of each car are mentioned. The change in the x-axis of the cars is mentioned. The coordinates at the x-axis and the y-axis of the rectangular area of each car are defined. The coordinates at the y-axis are kept constant, whereas the coordinates of the x-axis are increasing at 750 coordinate compared to the previous car. Each increment is then added to the

value assigned to the first car placed as the hurdle in the game.

- In the last, the method pygame.display.update would be called to update the display settings.

Code:

```
import pygame
from pygame import mixer

pygame.init()

# Game Screen Display
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("The Night RiderZ")
icon = pygame.image.load("logo.png")
pygame.display.set_icon(icon)

# Images - Background, Ground, Rider, Hurdles
background = pygame.image.load("sky.jpeg")
background = pygame.transform.scale(background,
(800, 450))
ground = pygame.image.load("ground.png")
rider = pygame.image.load("rider.png")
car1 = pygame.image.load("car1.png")
car1 = pygame.transform.scale(car1, (200, 80))
car2 = pygame.image.load("car2.png")
car2 = pygame.transform.scale(car2, (200, 80))
car3 = pygame.image.load("car3.png")
car3 = pygame.transform.scale(car3, (200, 80))
car4 = pygame.image.load("car4.png")
car4 = pygame.transform.scale(car4, (200, 80))
car5 = pygame.image.load("car5.png")
car5 = pygame.transform.scale(car5, (200, 80))
car6 = pygame.image.load("car6.png")
car6 = pygame.transform.scale(car6, (200, 80))



# Background Music
```

```
mixer.init()
mixer.music.load('E:\Downloads\ontheroad.mp3')
mixer.music.play(-1)

# Game Constants
font = pygame.font.Font('freesansbold.ttf', 20)
crash_font =
pygame.font.Font('freesansbold.ttf', 50)
fps = 60
timer = pygame.time.Clock()
white = (255, 255, 255)

# Game Variables
instruction = True
active = False
replay = False
score = 0
highest_score = 0

# For Collisions
rider_rect = rider.get_rect()
car1_rect = car1.get_rect()
car2_rect = car2.get_rect()
car3_rect = car3.get_rect()
car4_rect = car4.get_rect()
car5_rect = car5.get_rect()
car6_rect = car6.get_rect()

# Movement of hurdles
car1X = 1000
carY = 340
carX_change = 10
car1_rect.x = car1X
car1_rect.y = carY
car2X = car1X + 750
```

```
car2_rect.x = car2X
car2_rect.y = carY
car3X = car1X + 1500
car3_rect.x = car3X
car3_rect.y = carY
car4X = car1X + 2250
car4_rect.x = car4X
car4_rect.y = carY
car5X = car1X + 3000
car5_rect.x = car5X
car5_rect.y = carY
car6X = car1X + 3750
car6_rect.x = car6X
car6_rect.y = carY

# Placement and Jumping of Rider
riderX = 30
riderY = 300
riderY_change = 0
gravity = 1
rider_rect.x = riderX
rider_rect.y = riderY


def scoring():
    global score, ground_speed, carX_change
    score += 0.1
    score_text = font.render("Score: " +
str(int(score)), True, white)
    screen.blit(score_text, (680, 60))
    if score % 100 == 0:
        ground_speed += 1
        carX_change += 1
```

```
def high_score():
    global highest_score
    highest_score_text = font.render("Highest
Score: " + str(int(highest_score)), True, white)
    screen.blit(highest_score_text, (600, 40))


# Movement of Ground
ground_speed = 10

ground = pygame.transform.scale(ground, (1280,
220))
groundX = 0


# Infinite While Loop
running = True
while running:
    timer.tick(fps)
    screen.blit(background, (0, 0))
    screen.blit(ground, (groundX, 395))
    screen.blit(ground, (groundX + 1280, 395))
    screen.blit(rider, rider_rect)
    high_score()

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE and
instruction is True and active is False and
replay is False:
                instruction = False
                active = True
```

```
            if event.key == pygame.K_SPACE and
riderY_change == 0 and active is True and
instruction is False and replay is False:
                riderY_change = 25

            if event.key == pygame.K_SPACE and
replay is True and instruction is False and
active is False:
                instruction = True
                replay = False

    if instruction is True and active is False
and replay is False:
        instruction_text1 =
font.render("Instructions:", True, white)
        screen.blit(instruction_text1, (350,
60))
        instruction_text2 = font.render("Press
Space to Jump", True, white)
        screen.blit(instruction_text2, (315,
90))
        instruction_text3 = font.render("Avoid
Crashing With Any Vehicle", True, white)
        screen.blit(instruction_text3, (260,
120))
        instruction_text4 = font.render("Press
Space to Continue", True, white)
        screen.blit(instruction_text4, (295,
200))

    if active is True and instruction is False
and replay is False:
        if riderY_change > 0 or rider_rect.y <
300:
```

```
            rider_rect.y -= riderY_change
            riderY_change -= gravity
        if rider_rect.y > 300:
            rider_rect.y = 300
        if rider_rect.y == 300 and riderY_change
< 0:
            riderY_change = 0

        groundX -= ground_speed
        if groundX <= -1280:
            groundX = 0

        car1_rect.x -= carX_change
        car2_rect.x -= carX_change
        car3_rect.x -= carX_change
        car4_rect.x -= carX_change
        car5_rect.x -= carX_change
        car6_rect.x -= carX_change
        screen.blit(car1, (car1_rect.x,
car1_rect.y))
        screen.blit(car2, (car2_rect.x,
car2_rect.y))
        screen.blit(car3, (car3_rect.x,
car3_rect.y))
        screen.blit(car4, (car4_rect.x,
car4_rect.y))
        screen.blit(car5, (car5_rect.x,
car5_rect.y))
        screen.blit(car6, (car6_rect.x,
car6_rect.y))

        if rider_rect.colliderect(car1_rect):
            active = False
            replay = True
```

```
if rider_rect.colliderect(car2_rect):
    active = False
    replay = True

if rider_rect.colliderect(car3_rect):
    active = False
    replay = True

if rider_rect.colliderect(car4_rect):
    active = False
    replay = True

if rider_rect.colliderect(car5_rect):
    active = False
    replay = True

if rider_rect.colliderect(car6_rect):
    active = False
    replay = True

if car1_rect.x <= -4000:
    car1_rect.x = 800
if car2_rect.x <= -4000:
    car2_rect.x = 800
if car3_rect.x <= -4000:
    car3_rect.x = 800
if car4_rect.x <= -4000:
    car4_rect.x = 800
if car5_rect.x <= -4000:
    car5_rect.x = 800
if car6_rect.x <= -4000:
    car6_rect.x = 800

scoring()
```

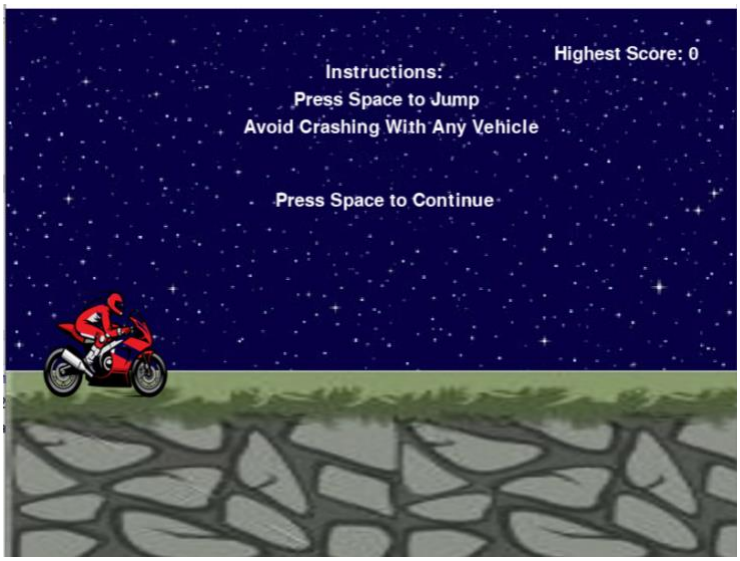
```
    if replay is True and instruction is False
and active is False:
        replay_text = crash_font.render("You
Crashed!", True, white)
        screen.blit(replay_text, (245, 200))
        continue_text = font.render("Press Space
to Continue", True, white)
        screen.blit(continue_text, (290, 280))
        if score > highest_score:
            highest_score = score
        score = 0
        car1X = 1000
        carY = 340
        carX_change = 10
        car1_rect.x = car1X
        car1_rect.y = carY
        car2X = car1X + 750
        car2_rect.x = car2X
        car2_rect.y = carY
        car3X = car1X + 1500
        car3_rect.x = car3X
        car3_rect.y = carY
        car4X = car1X + 2250
        car4_rect.x = car4X
        car4_rect.y = carY
        car5X = car1X + 3000
        car5_rect.x = car5X
        car5_rect.y = carY
        car6X = car1X + 3750
        car6_rect.x = car6X
        car6_rect.y = carY
        riderX = 30
        riderY = 300
        riderY_change = 0
        rider_rect.x = riderX
```

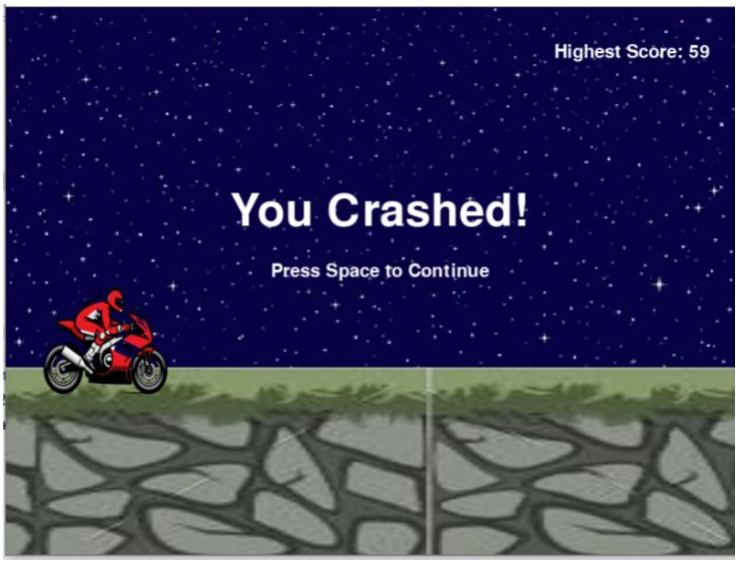
```
        rider_rect.y = riderY

    pygame.display.update()
```

Screenshots of the project:

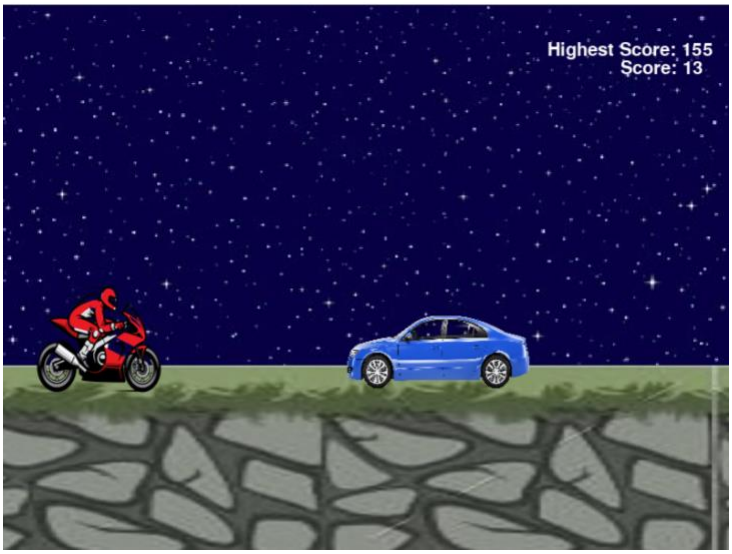
The Game Icon:

Images while the game is played:

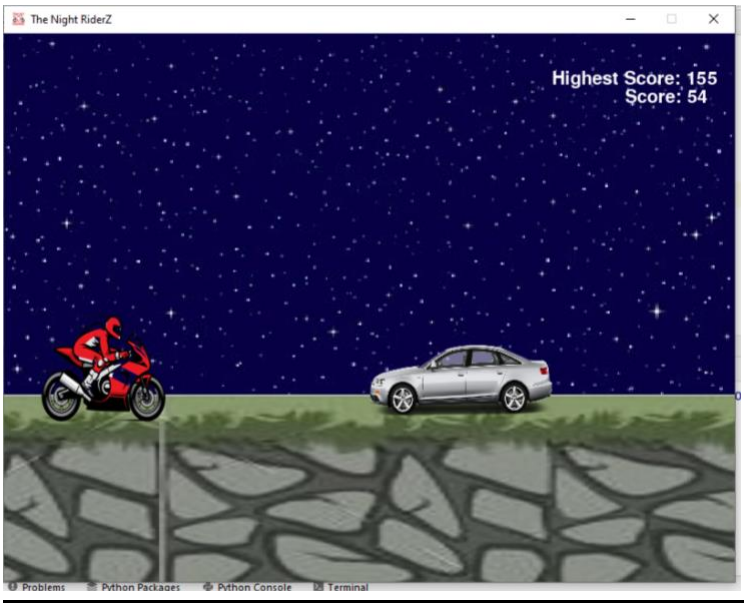


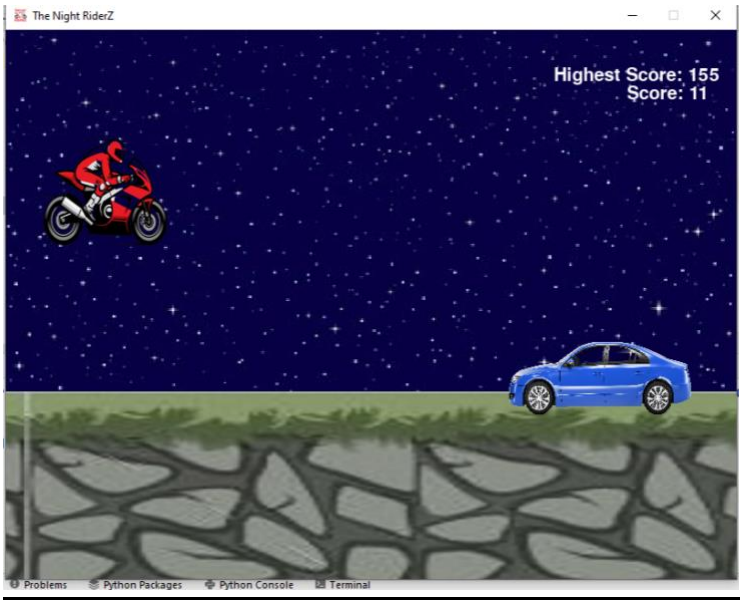

Highest Score: 59
You Crashed!
Press Space to Continue

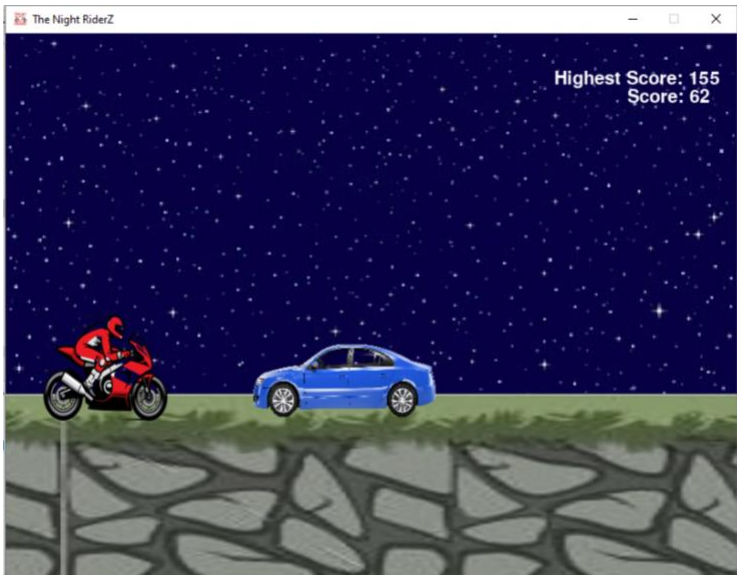

Highest Score: 155
Score: 10

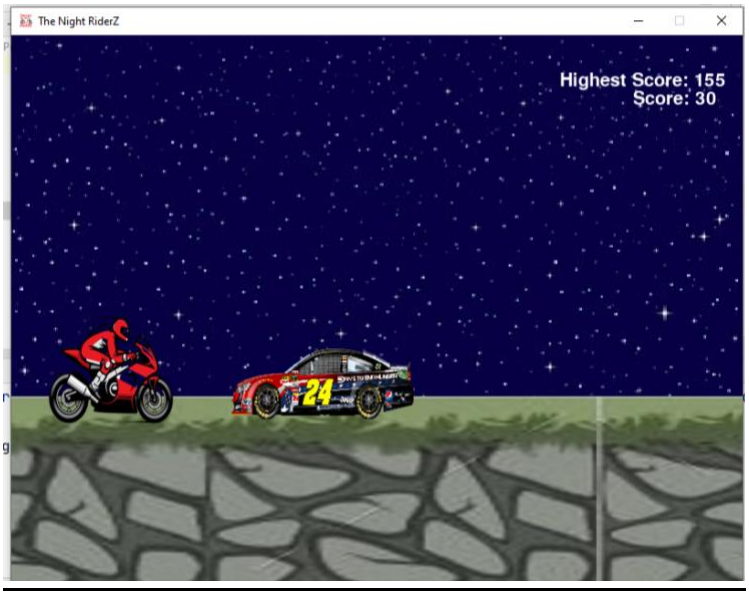

Highest Score: 155
Score: 13


Highest Score: 155
Score: 26


The Night RiderZ
Highest Score: 155
Score: 54
Problems
Python Packages
Python Console
Terminal


The Night RiderZ
Highest Score: 155
Score: 11
Problems
Python Packages
Python Console
Terminal


The Night RiderZ
Highest Score: 155
Score: 62


The Night RiderZ
Highest Score: 155
Score: 30