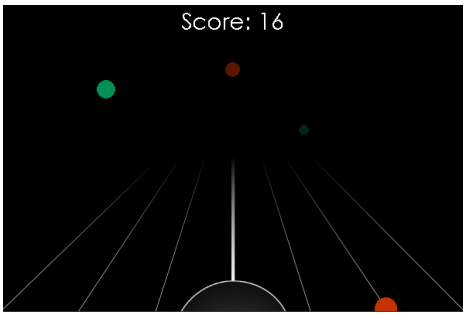
Computer Seminar

May 2023

1 Program explanation

The game involves the player avoiding the red balls thrown at them while trying to collect the green balls to get the score. The player will be able to move horizontally using the left and right key, staying in one of the three lanes. When



the player catches the green ball, their score increase by one. And the game will be over when the player catches the red ball. As the game continues, the ball will be spawn faster, thus, increasing the difficulty of the game.

2 Implementation

For the game, I created 3 kinds of object that will control and be used in the main function: `Ball()`, `Player()`, and `Gamemanager()`. First, I will go through the `main()` function to have a rough idea of how the game operates. Then, the detail explanations of each class objects will be provide.

2.1 `main()` function

```
1 def main():
2     pygame.init()
3     pygame.font.init()
4     width, height = 600, 400
5     screen = pygame.display.set_mode((width, height))
```

```
    clock = pygame.time.Clock()
    frames_per_second = 60
    player = Player()
    game = Gamemanager(frames_per_second, player)
```

In the first part of the main function, pygame is initialized, the display screen and the frame rate (`frames_per_second`) of the game is defined. Then, player is define as Player object and game is defined as Gamemanger object.

```
    while True:
        clock.tick(frames_per_second)

        should_quit = False
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                should_quit = True
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    should_quit = True
                elif event.key == pygame.K_LEFT:
                    player.shift_left()
                elif event.key == pygame.K_RIGHT:
                    player.shift_right()
                elif event.key == pygame.K_SPACE and game.gameover:
                    main()

        if should_quit:
            break
```

The second part is the while loop which will repeat itself every 1/60 second, as defined in the `frames_per_second`. In the for-loop over `pygame.event.get()`, the specified input from the keyboard will be received to operate the following task:

Key	Input
Escape	Quit and close the game window
Left	Shift the player leftward
Right	Shift the player rightward
Space	Restart the game when the game is over

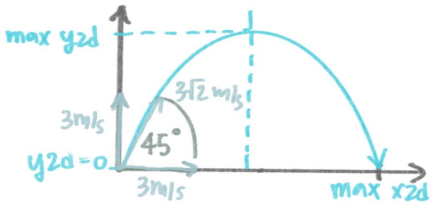
```
        screen.blit(game.background,(0,0))
        player.draw(screen)
        player.update()
        game.update(player)

        for a in list(reversed(game.ball_list)):
            a.draw(screen)
            a.update()

        game.checkstate(screen)
        game.display_score(screen)

        pygame.display.update()

    pygame.quit()
```

The last part of the main function in involves, draw the player and the background on the screen. The `game.update()` deals with spawn the balls, and change the background according to the player position. The the for-loop over the list of balls, will update each balls position and draw them on the screen. The reason for the reverse list is for the older-spawned ball to be in front of the newly-spawned ball when drawn on the screen.

Lastly, the game manager will check whether the game is over or not, and display the score or display the game over screen.

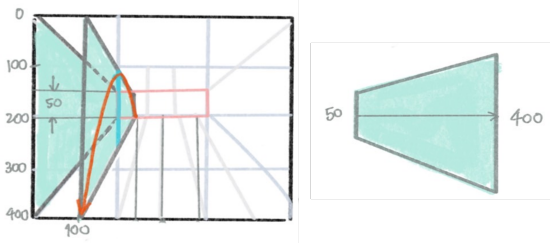
2.2 Class `Ball()`

The most challenging part of this game is how the ball shoots towards the player, similar to the 3d games. To do this we need to consider the dimensional effects on the ball's projectile. To do this, first, consider 1 simple projectile motion: shooting the ball 45 degrees with velocity of $3\sqrt{2}$ m/s, where the gravitational acceleration is set to -2 m/s.



Using the above equation of motion, we obtain the flight time of the projectile to be 3 seconds, the maximum distance traveled horizontally and vertically to be 9 m and 2.25 m. I named to the variable considered here as `x2d` and `y2d`. After acquiring these values, I located 3 possible spawn points and its trajectory on the screen. Then, I project them on to the screen.

The projection of y-coordinate.



When projected, it can be seen that as the ball move towards the player, the maximum range the ball can move (maximum `y2d` = 2.25) will increase linearly. For simplicity, I call this variable `magnify`. In this case, I set the starting point

value to be 50 and the end point value to be the same as the screen height which is 400. By considering the linear equation, the `amplifier` equation (1) is obtain.

$$\texttt{magnify} = \left(\frac{\frac{400-50}{max\ \texttt{y2d}}}{max\ \texttt{x2d}}\right)\texttt{x2d} + \left(\frac{50}{max\ \texttt{y2d}}\right) \tag{1}$$

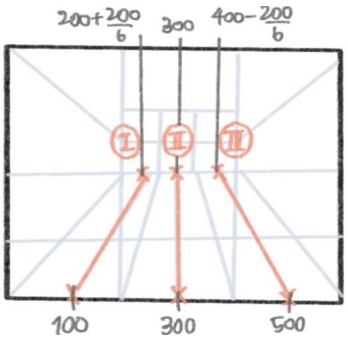
However, we also need to consider the shift in the $\texttt{y2d} = 0$ line or the ground, which when projected, it decreases linearly. For simplicity, I call this variable `offset`. When $\texttt{x2d} = 0$, the $\texttt{offset} = 200$, and when $\texttt{x2d} =$ maximum, the $\texttt{offset} = 400$ By considering the linear equation, the `offset` equation (2) is obtain.

$$\texttt{offset} = \left(\frac{400-200}{max\ \texttt{x2d}}\right) max\ \texttt{x2d} + 200 \tag{2}$$

By combining `magnify` and `offset`'s aspect, the relation between the projected `y` value and `y2d` is obtained as shown in the equation(3).

$$\texttt{y} = -\left(\texttt{magnify}\right)\texttt{y2d} + \texttt{offset} \tag{3}$$

The projection of x-coordinate.



For x projection, we need to divide the projection into 3 cases according to the 3 lanes. For the first case, the left lane, the start position on the screen is x=200+200/6 and the final position is 100. From this linear relation, we can compute the velocity of the projected x coordinate as vx = -(100+200/6)/time. For the second case, the middle lane, the x coordinate does not change. Lastly, the third case, the right lane, the start position on the screen is x=400-200/6 and the final position is 500. From this linear relation, we can compute the velocity of the projected x coordinate as vx = (100+200/6)/time.

```
class Ball():
    def __init__(self, case, type, frame_rate):
        self.case = case
        self.type = type
        self.frame = frame_rate/2.5
        self.x2d = 0
        self.y2d = 0
```

```
        self.vx = 3
        self.vy = 3
        self.ay = -2
        if self.case == 1:
            self.x = 200+(200/6)
        elif self.case == 2:
            self.x = 300
        elif self.case == 3:
            self.x = 400-(200/6)
        self.y = 200
        self.radius = 10
        self.is_alive = True
        self.color = [0, 0, 0]

    def update(self):
        #Calculateion in 2d plane before projection
        self.x2d += self.vx/self.frame
        self.vy += self.ay/self.frame
        self.y2d += self.vy/self.frame

        #Calculation after projection
        magnify = (((350/2.25)/9) * self.x2d) + (50/2.25)
        offset = (200/9)*self.x2d + 200
        if self.case == 1:
            self.x -= ((100+(200/6))/3)/self.frame
        elif self.case == 2:
            pass
        elif self.case == 3:
            self.x += ((100+(200/6))/3)/self.frame

        self.y = offset - magnify*self.y2d

        #Increasing radius
        self.radius += (60/9)/self.frame

        #Changing color
        if self.type == 1 and self.is_alive:
            self.color[1] = (200/9)*self.x2d
            self.color[2] = (120/9)*self.x2d
        elif self.type == 2 and self.is_alive:
            self.color[0] = (200/9)*self.x2d
            self.color[1] = (50/9)*self.x2d
```

The calculation previously explained is included in the `update()` method. In the section, the x and y coordinate before the projection (`x2d` and `y2d`)is calculated. It is noted that since the `update()` method is called every 1/60 second, every added value for acceleration and velocity must be further divide my the frame rate. However, after simulate the ball's projectile on the screen 3 seconds of flight time seemed to be too long, so the frame rated used in the calculation is divided by 2.5 in the `__init__()` section. Then, the 2d coordinates is projected into the screen coordinate. Moreover, to make the projection effect more obvious to the player, the ball size will increase and the color of the ball will change from black to green or red after it moves towards the player.

2.3 class `Player()`

The primary purpose of this class is to control the player movement. The player is able to move around three lanes by pressing left and right button. To do this, the attribute `state` is added, corresponding to the lane player is staying (1 for the left lane, 2 for middle, and 3 for the right lane). Firstly, when player is initialized, the player will stays in the middle lane or state 2. Then, if the method `shift_left()` or `shift_right()` is called the state will change by minus or plus 1.

```
    def shift_left(self):
        if self.state > 1 and self.control:
            self.state -= 1

    def shift_right(self):
        if self.state < 3 and self.control:
            self.state += 1
```

After the attribute `state` is changed, the `update()` method will change the player position according to the player's `state` by adding a some velocity value to the player's x-coordinate. However, the allow a smooth transition between lanes, the velocity value is determined by how far the player is from the supposed location it needs to be in according to it's state.

```
    def update(self):
        if self.state == 1 and self.x > 25:
            self.vx = (25 - self.x)/5
        elif self.state == 1:
            self.x = 25
            self.vx = 0

        if self.state == 2 and self.x != 225:
            self.vx = (225 - self.x)/5
        elif self.state == 2:
            self.x = 225
            self.vx = 0

        if self.state == 3 and self.x < 425:
            self.vx = (425 - self.x)/5
        elif self.state == 3:
            self.x = 425
            self.vx = 0

        self.x += self.vx
```

The code above show that if the player is in state 1, but it is not yet in the supposed x-coordinate which is 25, the velocity will be (25-its current position)/5.

2.4 class `Gamemanager()`

There are 3 main purposes for this class, spawning the ball, count the score, managing the display.

Ball spawning mechanism

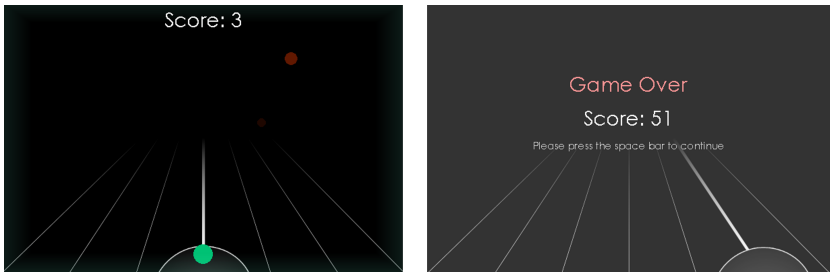
```
class Gamemanager():
    def __init__(self, frame_rate, player):
        self.blitcount = 0
        self.spawn_duration = 100
        self.spawn_cooldown = self.spawn_duration
        self.should_spawn = True

   def ball_spawn(self):
        case = random.randint(1,3)
        type = random.randint(1,2)
        a = Ball(case, type, self.frame)
        if type == 1:
            self.gball_list.append(a)
            self.ball_list.append(a)
        elif type == 2:
            self.rball_list.append(a)
            self.ball_list.append(a)

    def update(self, player):
        if self.should_spawn and not self.gameover:
            self.ball_spawn()
            self.should_spawn = False
        else:
            self.spawn_cooldown -= 1

        if self.spawn_cooldown <= 0:
            self.should_spawn = True
            self.spawn_cooldown = self.spawn_duration

        if self.spawn_duration > 20:
            self.spawn_duration -= 0.3
        else:
            self.spawn_duration -= 0.003
```

Firstly, the `ball_spawn()` method is defined. In this method, the ball's `type` (red or green) and the lane (`case`) it will spawn into are random. Then, the newly generated ball is appended according to its `type`. To actually generate the ball, the `ball_spawn` method is called in the `update()` method periodically. This is done by using `spawn_duration`, `spawn_cooldown`, `should_spawn` attributes. First, set the condition that if `should_spawn` is `True`, spawn the ball, set `should_spawn` to `False`, and set the `spawn_cooldown` to equal to `spawn_duration`. Then, every 1/60 seconds, the `update()` method is called and the `spawn_cooldown` will decrease by 1. If the `spawn_cooldown` becomes zero, the attribute `should_spawn` will be set to `True` again and the ball will be spawn. To increase the difficulty, the `spawn_duration` is decreased, so the ball will be spawn more frequently. However the rate of decrease will be different in the early game (decrease with a faster rate) and late game (decrease with slower rate) to make the game more engaging and playable.

Score counting mechanism

```
class Gamemanager():
    def __init__(self, frame_rate, player):
        self.frame = frame_rate
        self.ball_list = []
        self.gball_list = []
        self.rball_list = []
        self.background_gameover = pygame.image.load(self.
    gameover_path)
        self.background_score = pygame.image.load(self.score_path)
        self.player = player
        self.player_x = 225
        self.playerrect=pygame.Rect((self.player_x, 395),(150, 20))
        self.score = 0

    def update(self, player):
        self.player_x = player.x
        self.ball_list[:] = [a for a in self.ball_list if a.
    is_alive]
        self.gball_list[:] = [g for g in self.gball_list if g.
    is_alive]
        self.rball_list[:] = [r for r in self.rball_list if r.
    is_alive]
        self.playerrect = pygame.Rect((self.player_x, 395),(150,
    20))
        self.blitcount -= 1

    def checkstate(self, screen):
        for g in self.gball_list:
            if g.ball.colliderect(self.playerrect) and g.is_alive:
                g.is_alive = False
                if not self.gameover:
                    self.score += 1
                    self.blitcount = 8
                    print(self.score)

        for r in self.rball_list:
            if r.ball.colliderect(self.playerrect) and r.is_alive:
                r.is_alive = False
                self.gameover = True
                self.player.control = False

        if self.blitcount > 0:
            screen.blit(self.background_score, (0,0))

        if self.gameover:
            screen.blit(self.background_gameover, (0,0))
```

To detect that the player has collected the ball, `rect.colliderect(rect)` function is used to detect the collision between two `rect` object. From the previous section, the ball is already defined as `rect` object, but the player is not. Thus, the invisible `rect` is drawn over the player's position and its position will always be updated as the player moves. Now, the collision can be detected. In the function `checkstate()`, for-loop over the list of green ball is defined. if there is any collision, the score will increase by 1. Moreover, to make the



player notice that they score a point, the corner of the screen will slightly turn green by overlaying the another transparent with slightly green border picture (`self.background_score` attribute) over the screen for 8/60 seconds.

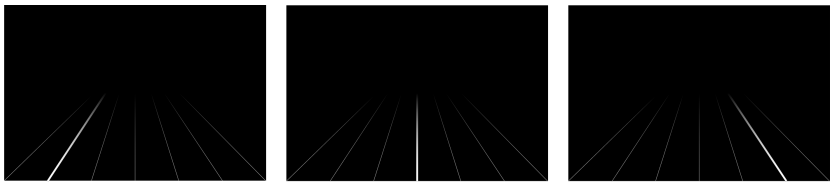
On the other hand, for-loop over the list of red ball is also defined. if there is any collision, the game will be over. In this state, the player will not be able to move, the screen will be overlay with the game over screen, which is a whitish semi-transparent screen.

Managing the display mechanism

```
class Gamemanager():
    def __init__(self, frame_rate, player):
        self.background_list = [pygame.image.load(self.
    background_path.format(k)).convert() for k in range(1,4)]
        self.background = self.background_list[1]

    def update(self, player):
        self.player_x = player.x
        self.playerrect = pygame.Rect((self.player_x, 395),(150,
    20))
        self.blitcount -= 1

    def display_score(self, screen):
        my_font = pygame.font.SysFont('Century Gothic', 30)
        my_font_sub = pygame.font.SysFont('Century Gothic', 15)
        if not self.gameover:
            score = my_font.render("Score: " + str(self.score),
    False, (255, 255, 255))
            score_width = score.get_width()
            screen.blit(score, (300-score_width/2,5))
        else:
            gameover = my_font.render("Game Over", False, (255,
    150, 150))
            space = my_font_sub.render("Please press the space bar
    to continue", False, (200, 200, 200))
            score = my_font.render("Score: " + str(self.score),
    False, (255, 255, 255))
            gameover_width = gameover.get_width()
            space_width = space.get_width()
            score_width = score.get_width()

            screen.blit(gameover, (300-gameover_width/2,100))
```

```
27                 screen.blit(space, (300-space_width/2,200))
28                 screen.blit(score, (300-score_width/2,150))
```

The primary task of this section is to display the score on the screen as defined in `display_score()` function. When the game is not over, the score will be display on middle-top of the screen. And when the game is over, the "Game over" text, the score and the instruction to press the space bar to restart the game will be displayed in the middle of the screen. Moreover, this section



also deal with changing the background according to the player position to emphasize which lane the player is currently in.