Wormy лабараториска вежба

1. По 20 секунди од почетокот на играта додадете друг црв. Оригиналниот црв треба да го избегнува вториот црв. Ако го допре со главата, неговото тело расте за еден сегмент. Ако вториот црв го допре оригиналниот, тогаш неговото тело се продолжува за еден сегмент. Движењето на вториот црв е случајно.

```
# Change for requirement 1:
ENEMY_WORM_APPEARING_TIME = 20  # the enemy worm will appear 20
seconds after the game starts
ENEMY_WORM_COLOR = (255, 0, 255)  # enemy worm will be purple
```

Додаваме константи кој ќе дефинираат кога ќе со појави непријателскиот црв и која ќе биде неговата боја.

```
# Change for requirement 1:
start_time = time.time()  # save the start time
enemyWormCoords = None  # no enemy worm at the beggining of the game
playerHitsEnemy = False  # a flag that is true if the player have hit
the enemy this turn (frame)
enemyHitsPlayer = False  # a flah that is true if the enemy has hit
the player this turn (frame)
enemyDirection = RIGHT  # represent the direction of the enemy worm
```

Во runGame() додаваме неколку променливи кои: ќе го чваат времето на почеток на играта, листа од координати која ќе претставува непријателски црв, логичка променлива дали играчот го допрел непријателскиот црв овој потег (frame), логичка променлива дали непријателскиот црв го допрел играчот овој потег (frame), насоката на движење на непријателскиот црв

```
# Change for requirement 1:
if enemyWormCoords is None and time.time() -
ENEMY_WORM_APPEARING_TIME > start_time:  # if true it is time to
create enemy worm
    startx = random.randint(5, CELLWIDTH - 6)
    starty = random.randint(5, CELLHEIGHT - 6)
    enemyWormCoords = [{'x': startx, 'y': starty},
                       {'x': startx - 1, 'y': starty},
                       {'x': startx - 2, 'y': starty}]
if enemyWormCoords:
    enemyDirection = getRandomDirection(enemyDirection,
enemyWormCoords)  # get enemy direction
```

Во main loop се проверува дали е време да се креира непријателскиот црв, ако е време тогаш се креира така што се поставува на локација по случаен избор и со должина од три сегменти. Во следниот if насоката на движење на непријателот се избира по случаен избор со користење на функцијата getRandomDirection

```
if enemyWormCoords:  # logic for the enemy worm
    for enemyWormBody in enemyWormCoords:  # check if the player worm
```

```
has hit the enemy worm
        if coalision(wormCoords[HEAD], enemyWormBody):
            # if wormCoords[HEAD]['x'] == enemyWormBody['x'] and
wormCoords[HEAD]['y'] == enemyWormBody['y']:
            playerHitsEnemy = True

    for wormBody in wormCoords:  # check if the enemy worm has hit
the player worm
        if coalision(enemyWormCoords[HEAD], wormBody):
            # if enemyWormCoords[HEAD]['x'] == wormBody['x'] and
enemyWormCoords[HEAD]['y'] == wormBody['y']:
            enemyHitsPlayer = True

    if wormEatApple(enemyWormCoords[HEAD], apple):  # check if enemy
ate red apple
        apple = getRandomLocation()  # set a new apple somewhere
    elif playerHitsEnemy:  # check if player hit enemy
        playerHitsEnemy = False  # reset the flag
    else:  # if the enemy worm didn't ate apple and didn't got hit by
the player
        del enemyWormCoords[-1]  # remove enemy worm's tail segment
```

Потоа додаваме сегмент кој ќе се справува со логика за непријателот. Први проверуваме дали играчот го допрел непријателскиот црв овој потег и дали непријателскиот црв го допрел играчот овој потег и ги поставуваме логичките променливи соодветно. Потоа проверуваме дали непријателот изел јаболко и потоа доколку непријателот не изел јаболко и не бил допрен од играчот го отсрануваме последниот сегмент (во таков случај големината на црвот останува непроменета, а во спротивен се зголемува за еден)

```
if wormEatApple(wormCoords[HEAD], apple):  # check if player ate red
apple
    # if wormCoords[HEAD]['x'] == apple['x'] and
wormCoords[HEAD]['y'] == apple['y']:
    # don't remove worm's tail segment
    apple = getRandomLocation()  # set a new apple somewhere
# Change for requirement 1:
elif enemyHitsPlayer:  # check if enemy hit player
    enemyHitsPlayer = False  # reset the flag
else:  # if the player didn't ate apple and didn't got hit by the
enemy
    del wormCoords[-1]  # remove worm's tail segment
```

Потоа додаваме код во делот каде се проверува дали големината на играчот треба да се промени, така што ако непријателот го допрел играчот големината на играчот се зголемува.

```
# Change for requirement 1:

# if direction == UP:
#     newHead = {'x': wormCoords[HEAD]['x'], 'y':
wormCoords[HEAD]['y'] - 1}
# elif direction == DOWN:
#     newHead = {'x': wormCoords[HEAD]['x'], 'y':
wormCoords[HEAD]['y'] + 1}
# elif direction == LEFT:
#     newHead = {'x': wormCoords[HEAD]['x'] - 1, 'y':
```

```
wormCoords[HEAD]['y']}
# elif direction == RIGHT:
#     newHead = {'x': wormCoords[HEAD]['x'] + 1, 'y':
wormCoords[HEAD]['y']}
newHead = getNewHead(wormCoords[HEAD], direction)  # the if else
segment is replaced with a function
wormCoords.insert(0, newHead)  # add the new head to the player worm

# Change for requirement 1:
if enemyWormCoords:
    newEnemyHead = getNewHead(enemyWormCoords[HEAD], enemyDirection)
# get new head
    enemyWormCoords.insert(0, newEnemyHead)  # add the new head to
the enemy worm
```

Логиката за креирање на нова глава ја преместуваме во функција getNewHead. Оваа функција ја искористиме да додадеме нова глава и на оригиналниот црв и на новит односно непријателскиот црв.

```
# Change for requirement 1:
if enemyWormCoords:
    drawWorm(enemyWormCoords, ENEMY_WORM_COLOR)  # draw the enemy
worm with its color
```

Го исцртуваме непријателскиот црв

```
# Change for requirement 1:
# return a random direction for the enemy worm
# there is more chance the direction the worm is currently moving to
be returned
# the enemy never hits the screen borders
# the enemy will only eat himself if it has no other options
def getRandomDirection(currentDirection, wormCord):
    head = wormCord[HEAD]
    directions = [LEFT, RIGHT, UP, DOWN]
    # replace the opposite direction with the current
    # the effect of the replacement is removed illegal move and added
more chance for the worm to not change direction
    if currentDirection == LEFT:
        directions = [LEFT, LEFT, UP, DOWN]
    if currentDirection == RIGHT:
        directions = [RIGHT, RIGHT, UP, DOWN]
    if currentDirection == UP:
        directions = [LEFT, RIGHT, UP, UP]
    if currentDirection == DOWN:
        directions = [LEFT, RIGHT, DOWN, DOWN]

    # remove the directions from the list that can cause the enemy
worm to hit the window border
    if head['x'] == 0:
        while LEFT in directions:
            directions.remove(LEFT)
    if head['x'] == CELLWIDTH - 1:
        while RIGHT in directions:
            directions.remove(RIGHT)
    if head['y'] == 0:
        while UP in directions:
            directions.remove(UP)
    if head['y'] == CELLHEIGHT - 1:
```

```
        while DOWN in directions:
            directions.remove(DOWN)
    # make a back-up of the directions list. At this point the list
contains at least 1 direction
    backup_directions = directions.copy()

    # get all possible next heads
    head_right = getNewHead(head, RIGHT)
    head_left = getNewHead(head, LEFT)
    head_up = getNewHead(head, UP)
    head_down = getNewHead(head, DOWN)

    # remove the directions from the list that can cause the enemy
worm to hit itself
    # after this logic the directions list can be empty
    for body in wormCord:
        if coalision(head_right, body):
            while RIGHT in directions:
                directions.remove(RIGHT)
        elif coalision(head_left, body):
            while LEFT in directions:
                directions.remove(LEFT)
        elif coalision(head_up, body):
            while UP in directions:
                directions.remove(UP)
        elif coalision(head_down, body):
            while DOWN in directions:
                directions.remove(DOWN)

    # return a random direction
    if len(directions) != 0:
        return random.choice(directions)
    return random.choice(backup_directions)
```

Функцијата getRandomDirection како аргументи прима моменталната насока на движење и листата која го претставува непријателскиот црв. Како резултат враќа насока по случаен избор за непријателскиот црв, шансата црвот да продолжи да се движи по досегашната насока е поголема со цел црвот да се движи пореално и да не се заплеткува во јазли, потоа се осигураме дека црвот никогаш нема да ја удро рамката на екранот и со тоа да излезе надвор од екранот и исто така црвот ќе се јаде(преклопува) сам себе ако и само ако нема друг избор.

За да се имплементира оваа логика најпрвин креираме листа која ги содржи сите насоки и потоа ја заменуваме спротивната насока од досегашната со досегашната со ова се отстранува невалидна насока на движење и се зголемуваат шансите црвот да продолжи да се движи во истата насока. Потоа ги отсрануваме насоките во листата кои може да предизвикаат црвот да излезе надвор од рамките на екранот. Во овој момент во листата имаме барем една насока па затоа листата ја копираме во нова која ќе служи како резерва. Потоа ги земаме сите возможни следни позиции и потоа проверуваме дали некоја од наредните позиции може да доведе црвот да се гризне сам зебе и ги отсрануваме тие насоки, по оваа логика возможно е листата со насоки да е празна. Па затоа проверуваме доколку листата има насоки враќаме една од тие насоки по случаен избор а во спротивен случај враќаме насока по случаен избор од резервната листа со насоки.

```
# Change for requirement 1:
# return true if the worm's head is on the same spot as the apple
def wormEatApple(head, apple):
    return coalision(head, apple)
```

Функцијата wormEatApple прима аргументи кој ја претставуваат главата на црвот и јаболко а како резултат враќа True ако главата и јаболкото се наоѓаат на исто место (имаат исти координати) за проверката се користи функцијата coalision()

```
# Change for requirement 1:
# return true if both objects are on the same spot
def coalision(obj1, obj2):
    return obj1['x'] == obj2['x'] and obj1['y'] == obj2['y']
```

Функцијата coalision како аргументи прима два објекти кои имаат x и y координати а како резултат враќа True доколку двата објекти се наоѓаат на исто место односно имаат исти координати а False во спротивен случај.

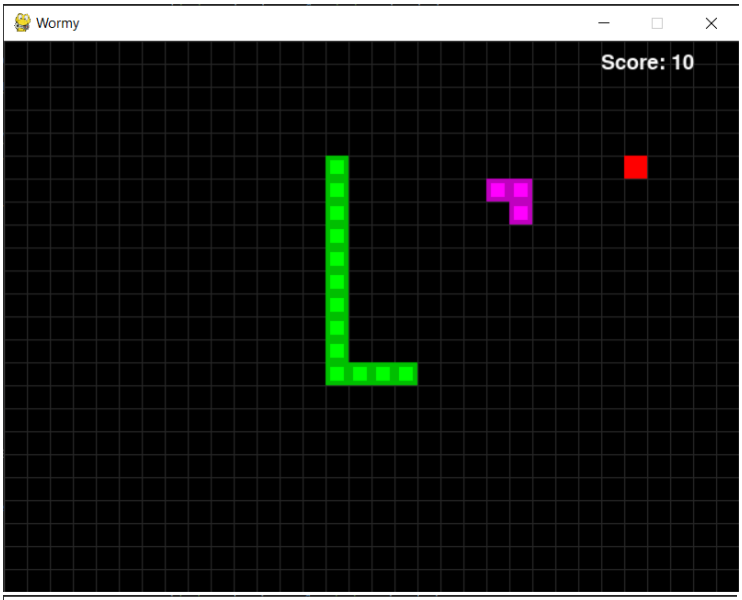
```
# Change for requirement 1:
# return new head created form the coordinates of the current head
moved by one position in the given direction
def getNewHead(head, direction):
    if direction == UP:
        newHead = {'x': head['x'], 'y': head['y'] - 1}
    elif direction == DOWN:
        newHead = {'x': head['x'], 'y': head['y'] + 1}
    elif direction == LEFT:
        newHead = {'x': head['x'] - 1, 'y': head['y']}
    else:  # direction == RIGHT
        newHead = {'x': head['x'] + 1, 'y': head['y']}
    return newHead
```

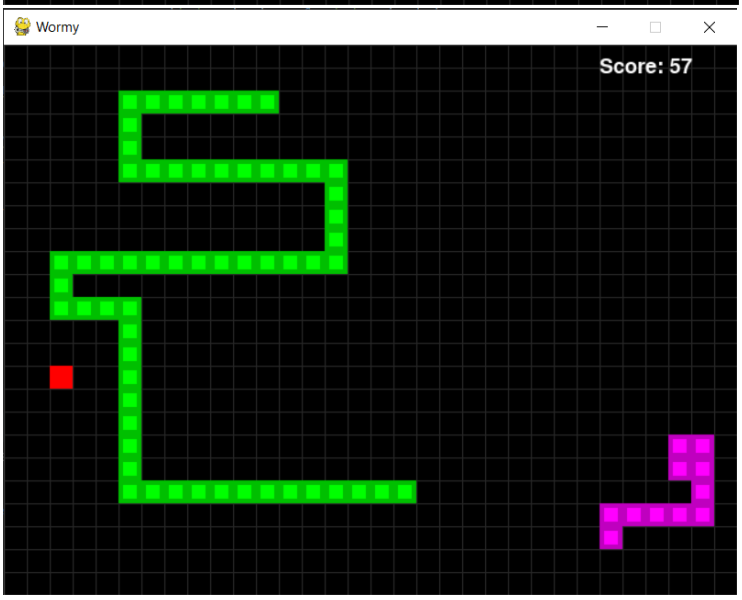
Функцијата getNewHead како аргументи прима глава на црв и насока а како резултат враќа нова галва, односно креира нова глава така што ги зема координатите на старата и ги поместува за една позиција во насоката која е пратена како аргумент

```
# Change for requirement 1:
# the function is changed so it can draw a worm in a color passed as
an argument
# def drawWorm(wormCoords):
def drawWorm(wormCoords, color=GREEN):
    # Change for requirement 1:
    r, g, b = color
    dark_color = (int(r * 0.75), int(g * 0.75), int(b * 0.75))  # get
a dark variant of the color, used for 3D effect
    for coord in wormCoords:
        x = coord['x'] * CELLSIZE
        y = coord['y'] * CELLSIZE
        wormSegmentRect = pygame.Rect(x, y, CELLSIZE, CELLSIZE)
        # Change for requirement 1:
        # pygame.draw.rect(DISPLAYSURF, DARKGREEN, wormSegmentRect)
        pygame.draw.rect(DISPLAYSURF, dark_color, wormSegmentRect)
        wormInnerSegmentRect = pygame.Rect(x + 4, y + 4, CELLSIZE -
8, CELLSIZE - 8)
        # pygame.draw.rect(DISPLAYSURF, GREEN, wormInnerSegmentRect)
        pygame.draw.rect(DISPLAYSURF, color, wormInnerSegmentRect)
```

Функцијата drawWorm ја менуваме така што ќе може да црта црв со боја пратена како аргумент. За таа цел ја менуваме дефиницијата на функцијата за да може да прима

уште еден аргумент color кој има предефинирана вредност зелена потоа во функцијата од бојата пратена како аргумент креираме темна варијанта која се користи за исцртување на надворешниот дел од еден сегмент од црвот а другата светлата боја се користи за исцртување на внатрешноста на сегментите.





2. Додадете два елемента што трепкаат (секој пат по три) на случајно избрани позиции со димензија од едно квадратче. Едниот се појавуваат во времетраење од 5 секунди на секои 5 секунди, а другиот еднократно, во времетраење од 7 секунди. Ако оригиналниот црв ги изеде, играчот добива дополнителни поени (по 3 за секој изеден елемент). Овие поени треба да се вклучат во конечниот резултат на начин што вие ќе го изберете. Треба да обезбедите објаснување за формулата што ја користите за пресметување на резултатот. Резултатот треба да се прикаже на екранот што се појавува кога играта ќе заврши.

Ќе ги додадеме истите елементи од вежбата работа на час жолто (златно) јаболко и сино јаболко, но ќе ги модифицираме да одговараат на барањата.

```
# Change for requirement 2:
GOLD_COLOR = (255, 215, 0)
GOLD_APPLE_TIMEOUT = 5  # the golden apple will last for 5s
GOLD_APPLE_COOLDOWN = 5  # the golden apple will reappear in 5s
BLUE = (0, 0, 255)
BLUE_APPLE_TIMEOUT = 7  # the blue apple will last for 7s
BLUE_APPLE_COOLDOWN = 15  # the blue apple will reappear in 15s
```

Најпрво додаваме константи кои ќе означуваат: златна боја, сина боја, (тајмоут) колку време ќе трае секој од новите елементи (јаболка) и (cooldown) односно време до повторно појавување за секое од јаболката.

```
# Change for requirement 2:
golden_apple = getRandomLocation()  # make golden apple at the start
of the game
golden_apple_appearing_time = time.time()  # save the golden apple
appearing time
blue_apple = getRandomLocation()  # make blue apple at the start of
the game
blue_apple_appearing_time = time.time()  # save the blue apple
appearing time

player_score = 0  # player score is 0 at the start of the game
enemy_score = 0  # enemy score is 0 at the start of the game
```

Потоа во runGame() додаваме променливи кои ќе ги означуваат јаболката (златно и сино) и нивните времиња на појавување. Исто така додаваме променливи кои ќе го чуваат резултатот (score) на играчот и на непријателот.

```
# Change for requirement 2:
if golden_apple is None:
    if time.time() - GOLD_APPLE_COOLDOWN >
golden_apple_appearing_time:  # check if golden apple should appear
        golden_apple = getRandomLocation()
        golden_apple_appearing_time = time.time()
elif time.time() - GOLD_APPLE_TIMEOUT > golden_apple_appearing_time:
# check if golden apple should disappear
    golden_apple = None

if blue_apple is None:
    if time.time() - BLUE_APPLE_COOLDOWN > blue_apple_appearing_time:
# check if blue apple should appear
        blue_apple = getRandomLocation()
        blue_apple_appearing_time = time.time()
elif time.time() - BLUE_APPLE_TIMEOUT > blue_apple_appearing_time:  #
check if blue apple should disappear
    blue_apple = None
```

Потоа во main loop додаваме код со кој проверуваме дали некое од новите јаболка треба да се појави односно креира (во случај јаболкото да не постои и времето на cooldown е поминато) и дали некое од јаболката треба да исчезне (во случај јаболкото да постои и времето за timeout да истече).

```
# Change for requirement 2:
if golden_apple and wormEatApple(enemyWormCoords[HEAD],
golden_apple):  # check if enemy ate golden apple
    golden_apple = None  # golden_apple is eaten
    enemy_score += 3  # enemy_score increases when enemy eats golden
apple
if blue_apple and wormEatApple(enemyWormCoords[HEAD], blue_apple):  #
check if enemy ate blue apple
    blue_apple = None  # blue apple is eaten
    enemy_score += 3  # enemy_score increases when enemy eats blue
apple

if wormEatApple(enemyWormCoords[HEAD], apple):  # check if enemy ate
red apple
    apple = getRandomLocation()  # set a new apple somewhere
    enemy_score += 1  # enemy_score increases when enemy eats red
apple
elif playerHitsEnemy:  # check if player hit enemy
    playerHitsEnemy = False  # reset the flag
    enemy_score += 2  # enemy_score increases when player hits enemy
else:  # if the enemy worm didn't ate apple and didn't got hit by the
player
    del enemyWormCoords[-1]  # remove enemy worm's tail segment
```

Потоа во делот каде одвива логиката на непријателскиот црв додаваме код со кој проверуваме дали непријателот изел некое од новите јаболка. Дополнително при јадење на некое од новите јаболка златно или сино црвот добива +3 score, доколку изеде обично црвено јаболко добива +1 score и доколку е допрен од играчот добива +2 score.

```
# Change for requirement 2:
if golden_apple and wormEatApple(wormCoords[HEAD], golden_apple):  #
check if player ate golden apple
    golden_apple = None  # golden_apple is eaten
    player_score += 3  # player_score increases when player eats blue
apple
if blue_apple and wormEatApple(wormCoords[HEAD], blue_apple):  #
check if player ate blue apple
    blue_apple = None  # blue_apple is eaten
    player_score += 3  # player_score increases when player eats blue
apple

if wormEatApple(wormCoords[HEAD], apple):  # check if player ate red
apple
    # if wormCoords[HEAD]['x'] == apple['x'] and
wormCoords[HEAD]['y'] == apple['y']:
    # don't remove worm's tail segment
    apple = getRandomLocation()  # set a new apple somewhere
    player_score += 1  # player_score increases when player eats red
apple
# Change for requirement 1:
elif enemyHitsPlayer:  # check if enemy hit player
    enemyHitsPlayer = False  # reset the flag
    # note: the player don't get additional score for getting hit my
the enemy
else:  # if the player didn't ate apple and didn't got hit by the
```

```
enemy
    del wormCoords[-1]  # remove worm's tail segment
```

Додаваме дел кој проверува дали играчот изел некое од новите јаболка златно или сино и дополнително додаваме логика за пресметување на резултат (score) така што доколку играчот изеде златно или сино јаболко добива +3 score, ако изеде обично односно црвено јаболко +1 score и доколку е допрен од непријателот не добива дополнителни поени (score).

```
# Change for requirement 2:
flashIsOn = round(time.time(), 1) * 10 % 10 >= 2
# each second the apples will be displayed for 0.7s and not displayed
for 0.3s
if golden_apple and flashIsOn:
    drawApple(golden_apple, GOLD_COLOR)  # drw the golden apple
flashIsOn = round(time.time(), 1) * 10 % 10 <= 6
if blue_apple and flashIsOn:
    drawApple(blue_apple, BLUE)  # drw the blue apple
```

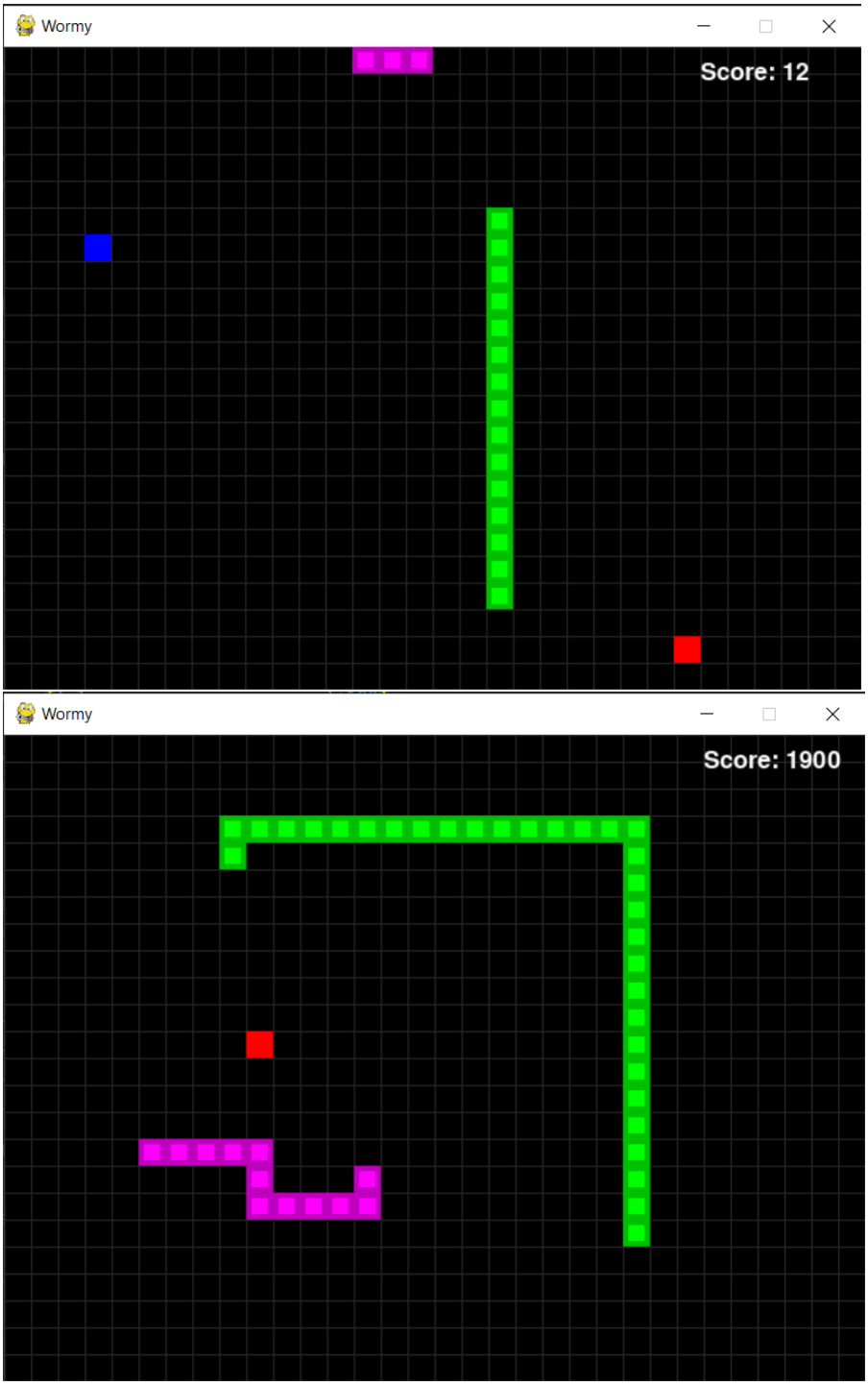
Додаваме код со кој што овозможуваме исцртување и трепкање на златното и синото јаболко. Треперењето зависи од променливата flashIsOn чија вредност се менува така што во секоја сунда 0.7 секунди е True а останатите 0.3 секунди е False.

```
# Change for requirement 2:
# drawScore(len(wormCoords) - 3)
drawScore((player_score - enemy_score) * 100)
# the total score is calculated as the difference between player
score and enemy score, multiplied by 100
```

Ја менуваме логиката за пресметување на поените (score) така што сека ќе се пресметуваат како разлика помеѓу поените на играчот (player_score) и поените на непријателот (enemy_score), помножена со 100. Со ова се стимулира играчот освен што ќе тежнее кон собирање на повеќе бодови, да тежнее и кон спречување на непријателот да собира бодови така што играчот би ги јадел јаблоката пред тие да бидат изедени од непријателот но исто така и поради тоа што кога играчот ќе го допре непријателот, непријателот добива поени а во обратен случај не се стимулира играчот да го избегнува непријателскиот црв.

```
# Change for requirement 2:
# the function is changed so it can draw an apple in a color passed
as an argument
# def drawApple(coord):
def drawApple(coord, color=RED):
    x = coord['x'] * CELLSIZE
    y = coord['y'] * CELLSIZE
    appleRect = pygame.Rect(x, y, CELLSIZE, CELLSIZE)
    # Change for requirement 2:
    # pygame.draw.rect(DISPLAYSURF, RED, appleRect)
    pygame.draw.rect(DISPLAYSURF, color, appleRect)
```

Функцијата DrawApple ја менуваме така што ќе овозможува исцртување на јаболка со боја која се праќа како аргумент.



3. На екранот што се појавува кога играта ќе заврши треба да се додадат две копчиња, "Start from the beginning" и "Quit". Кога играчот ќе кликне на првото копче, играта треба да почне од почеток (без да се појави почетниот екран). Кога играчот ќе кликне на второто копче, треба да се исклучи играта.

```
# Change for requirement 3:
def showGameOverScreen():
    # gameOverFont = pygame.font.Font('freesansbold.ttf', 150)
    # gameSurf = gameOverFont.render('Game', True, WHITE)
```

```
    # overSurf = gameOverFont.render('Over', True, WHITE)
    # gameRect = gameSurf.get_rect()
    # overRect = overSurf.get_rect()
    # gameRect.midtop = (WINDOWWIDTH / 2, 10)
    # overRect.midtop = (WINDOWWIDTH / 2, gameRect.height + 10 + 25)

    # DISPLAYSURF.blit(gameSurf, gameRect)
    # DISPLAYSURF.blit(overSurf, overRect)
    # drawPressKeyMsg()
    # pygame.display.update()
    # pygame.time.wait(500)
    # checkForKeyPress()  # clear out any key presses in the event
queue
    #
    # while True:
    #     if checkForKeyPress():
    #         pygame.event.get()  # clear event queue
    #         return

    # the gameOverFont decreased, game and over surf combined into
one gameOver surf
    gameOverFont = pygame.font.Font('freesansbold.ttf', 100)
    gameOverSurf = gameOverFont.render('Game Over', True, WHITE)
    gameOverRect = gameOverSurf.get_rect()
    gameOverRect.midtop = (WINDOWWIDTH / 2, 10)

    # added startAgain surf that represent a start again button
    buttonsFont = pygame.font.Font('freesansbold.ttf', 30)
    startAgainSurf = buttonsFont.render('Start from beggining', True,
WHITE)
    startAgainRect = startAgainSurf.get_rect()
    startAgainRect.topleft = (30, WINDOWHEIGHT / 2 + 50)

    # added quit surf that represent a quit button
    quitSurf = buttonsFont.render('Quit', True, WHITE)
    quitRect = quitSurf.get_rect()
    quitRect.topright = (WINDOWWIDTH - 100, WINDOWHEIGHT / 2 + 50)

    DISPLAYSURF.blit(gameOverSurf, gameOverRect)
    DISPLAYSURF.blit(startAgainSurf, startAgainRect)
    DISPLAYSURF.blit(quitSurf, quitRect)

    pygame.display.update()
    checkForKeyPress()  # clear out any key presses in the event
queue

    while True:
        for event in pygame.event.get():  # event handling loop
            if event.type == QUIT:
                terminate()
            if event.type == MOUSEBUTTONUP:
                if startAgainRect.collidepoint(event.pos):  # check
if start again is clicked
                    return  # this is followed up by runGame()
                elif quitRect.collidepoint(event.pos):  # check if
quit is clicked
                    terminate()  # quit the game
```

За да се иплементира ова барање потребно е да се промени функцијата showGameOverScreen така што најпрво фонтот на gameOverFont ќе го намалиме и ќе ги комбинираме текстуалните објекти Game и over во еден GameOver објект со цел да ги

собере и дополнителните копчиња. Потоа додаваме текстуални објекти StartAgain и Quit кои ќе ги претставуваат копчињата и ги прикажуваме на екран. Потоа влегуваме ве while true циклус од кој што се излегува при клик на едно од двете копчиња. При клик на Again завршува циклусот а со тоа и оваа функција по што потоа од main() ќе се повика runGame() функцијата. При клик на Quit се повикува функцијата terminate која ја терминира играта.