ЛАБОРАТОРИСКА ВЕЖБА 4
30.12.2020

ПРОГРАМИРАЊЕ НА ВИДЕО ИГРИ И СПЕЦИЈАЛНИ ЕФЕКТИ

Wormy

Професор*:* д-р Катарина Тројачанец Динева
Студент*:* Кирил Зеленковски

1. ***Барање 1***

По 45 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.

Пред се за ова барање потребен е тајмер од 45 секунди за прикажување на втор црв. Овој тајмер е потребно да започне во исто време кога играта започнува и потоа да го прикаже вториот црв кој има случајно движење.

Имплементацијата на тајмерот беше слична како барањето со јаболките (жолто и плаво) од работата на час. Започнуваме со знаменце за дали црвот е прикажан, на почетокот црвот е со знаме “hidden” и доколку поминат 45 секунди црвот почнува да се прикажува.

Ова барање може да се разложи на 3 компоненти:

- Тајмер за втор црв
- Рандом движење на црв
- Колизија на црвите
  - Тип 1 колизија: Прв црв го допира втор црв со својата глава (расте сегмент)
  - Тип 2 колизија: Втор црв го допира првиот црв било каде по телото (расте сегмент)

Тајмер за втор црв

Тајмерот е всушност бројач кој почнува од 45 и се движи до 0, кога доаѓа на нула се испишува само порака да се избегнува црвот.

```
def runGame():
    # Set a random start point.
    startx = random.randint(5, CELLWIDTH - 6)
    starty = random.randint(5, CELLHEIGHT - 6)
    wormCoords = [{'x': startx,     'y': starty},
                  {'x': startx - 1, 'y': starty},
                  {'x': startx - 2, 'y': starty}]

    wormCoords2 = [{'x': startx,     'y': starty},
                  {'x': startx - 1, 'y': starty},
                  {'x': startx - 2, 'y': starty}]

    direction = RIGHT
    direction2 = LEFT

    start_ticks = pygame.time.get_ticks()  # start ticks
    counter = 45  # додај бројач за нов црв
    flag = 'hidden'  # знаме за црв; 'hidden'-скриено, 'shown'-прикажано
```

Се дефинира пред се знамето на “hidden”, бројачот на 45 и се зимаат почетните ticks од почетокот на играта. Кога ќе дојдат на 45 се менува знамето.

Имплементацијата започнува во главниот game loop:

```
# 45 секунди за втор црв функција
counter, start_ticks, flag = drawSecondWormTime(counter, seconds, start_ticks, flag)
# на излез добиваме 3 параметри од кој еден е знамето: ако е shown – цртај црв во следен loop
```

Функцијата враќа 3 аргументи кои ги процесира на следниот начин:

```
# Главна функција за втор црв
def drawSecondWormTime(counter, seconds, start_ticks, flag):
    # HIDDEN: крај го црвот
    if flag == 'hidden':
        # 1 проверка: дали тајмерот истече
        if seconds == counter:
            # да, истече
            text = ''
            # испиши текст горе десно под score
            yellowSurf = BASICFONT.render(text, True, WHITE)
            yellowRect = yellowSurf.get_rect()
            yellowRect.topleft = (WINDOWWIDTH - 300, 25)
            DISPLAYSURF.blit(yellowSurf, yellowRect)
            # ресетирај бројач на 45. значи 45 секунди нема, потоа цело време е тука
            counter = 5
            # најбитно: ресетирај го почетното време на бројачот за game over
            start_ticks = pygame.time.get_ticks()
            # смени знаме на shown за кога ќе излезе да почне да го покажува
            flag = 'shown'
            # врати вредности
            return counter, start_ticks, flag
        else:
            # не е истечен сеуште, започни тајмер (descending) за кога е следното
            display_timer = counter-seconds
            # испиши текст горе десно под score
            text = f'Second worm appears in: {str(display_timer)}s'
            yellowSurf = BASICFONT.render(text, True, WHITE)
            yellowRect = yellowSurf.get_rect()
            yellowRect.topleft = (WINDOWWIDTH - 285, 25)
            DISPLAYSURF.blit(yellowSurf, yellowRect)
            # врати вредности
            return counter, start_ticks, flag

    else:
        text = f'Avoid second worm!'
        yellowSurf = BASICFONT.render(text, True, WHITE)
        yellowRect = yellowSurf.get_rect()
        yellowRect.topleft = (WINDOWWIDTH - 225, 25)
        DISPLAYSURF.blit(yellowSurf, yellowRect)
        # врати вредности
        return counter, start_ticks, flag
```

Оваа функција се состои од 2 главни проверки: каква е состојбата на знамето и дали секундите се еднакви на тајмерот. Доколку е состојба да криење (“hidden”), тогаш тајмерот врти додека не се исти. Секундата кога се исти преминува знамето во состојба прикажување (“shown”) и започнува да испишува “Avoid second worm!”. Знамето е многу битно и тоа ќе го објаснам зошто во следните страни.

Рандом движење на втор црв

Ова движење зависи многу од знамето во претходната функција. Сега, веќе кога знамето е “shown” имаат поминато 45 секунди и можеме да прикажуваме црв. Ова го правиме така што црвот го претставуваме со листа од речници со x,y како клучеви и вредности кординати. Ваква листа има првиот црв, правиме со слично име за вториот која се вика **wormCoords2**.

Иницирана е со исти почетни 3 сегменти дефинирани внатре во играта (исти како првиот).

Сега прво што напрвив е после прикажување на црвот со трите сегменту му давав рандом насока од листа со насоки, со random.choice(directions), каде directions е листа од сите можни насоки (LEFT, UP, ...) и потоа до движев слично како првиот што е придвижен од нашата насока.

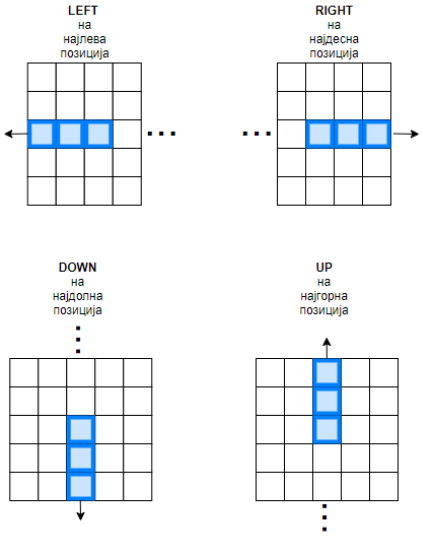
Но, иако ми делуваше доста логично на прв поглед, овде има грешка доколку не правиме **проверки** црвот за многу кусо време ќе **излезе** од играта односно:

- x кординатите на сегментите ќе бидат:
  - Негативни, т.е. лево од играта
  - поголеми од ширината (CELLWIDTH), т.е. десно од играта
- y кординатите на сегментите ќе бидат:
  - негативни, т.е. под играта
  - поголеми од висината (CELLHEIGHT), т.е. над од играта

Тука стана доста зезнато за имплементација на движење на вториот црв, бидејќи доколку даваме рандом вредности од листа со движења можни веројатно е дека ќе го снема брзо. Поради овој проблем, пробав да направам проверки со цел “регулирано ” случајно да го движиме, проверките се засноваат на каде е главата на вториот црв (првиот елемент од листата).

Секоја една “критична ситуација” ја разгранувам како сценарио кое резултира со можни насоки при таквии ситуации. Вакви “критични ситуации” се 4:

- Се наоѓаме најлево во игра а добиваме движење од случајен избор LEFT
- Се наоѓаме најдесно во игра а добиваме движење од случајен избор RIGHT
- Се наоѓаме најгоре во игра а добиваме движење од случајен избор UP
- Се наоѓаме најдолу во игра а добиваме движење од случајен избор DOWN

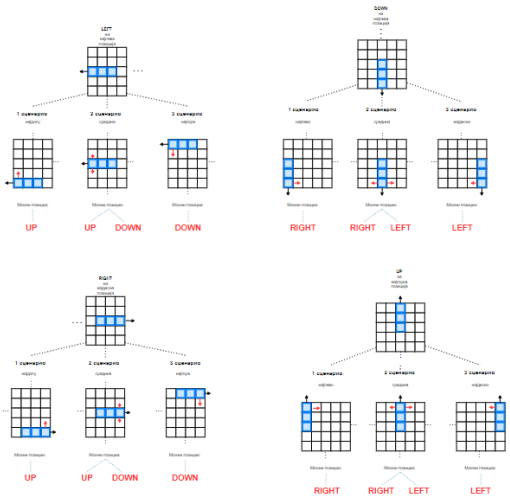


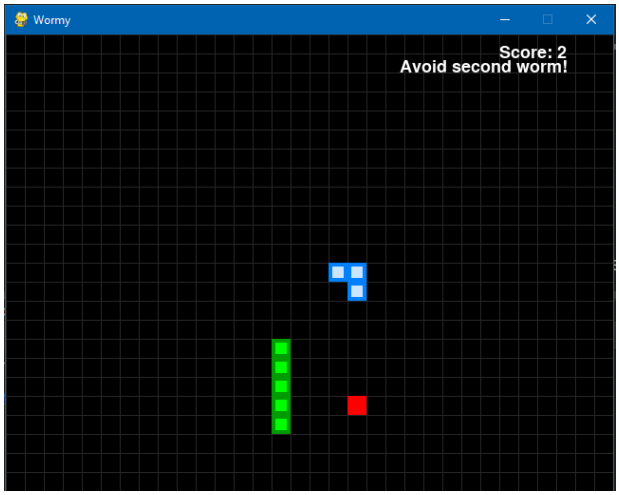
За цртање на сите следни цртежи како и овој користам: draw.io

Сега овие 4 проверки се во самиот код имплементирани како функција која изгледа вака (ги направив како collapse за да може да се гледа идејата ќе ја објаснам во целост подолу):

```
113         if flag == 'shown':
114             flag_direction = True
115             while flag_direction:
116                 direction2 = random.choice(directions)
117                 # најлево
118                 position_x = wormCoords2[0]['x']
119                 position_y = wormCoords2[0]['y']
120                 # најлево
121                 if direction2 == 'left' and position_x == 0:...
132                 # најдесно
133                 elif direction2 == 'right' and position_x == (CELLWIDTH-1):...
144                 # најдолу
145                 elif direction2 == 'up' and position_y == 0:...
156                 # најгоре
157                 elif direction2 == 'down' and position_y == (CELLHEIGHT-1):...
168                 # ако нема конфликтни ситуации, врати ја вредноста што е доделена прва
169                 direction2 = directions_dict[direction2]
170                 flag_direction = False
```

Значи самата функција ги прави овие проверки и потоа секоја една од овие критични ситуации е исполнета внатре во секоја итерација имаме разгранување на 3 можни сценарија кој се повеќе околку местоположбата на самиот црв (дали доколку е надоле е во средина или на десен ќош бидејќи ако е пр во десен ќош нема да има логика да му дадеме десно да се движи ќе избега со главата и повторно лоша проверка). Со таа цел го направив ова поголемо дрво:



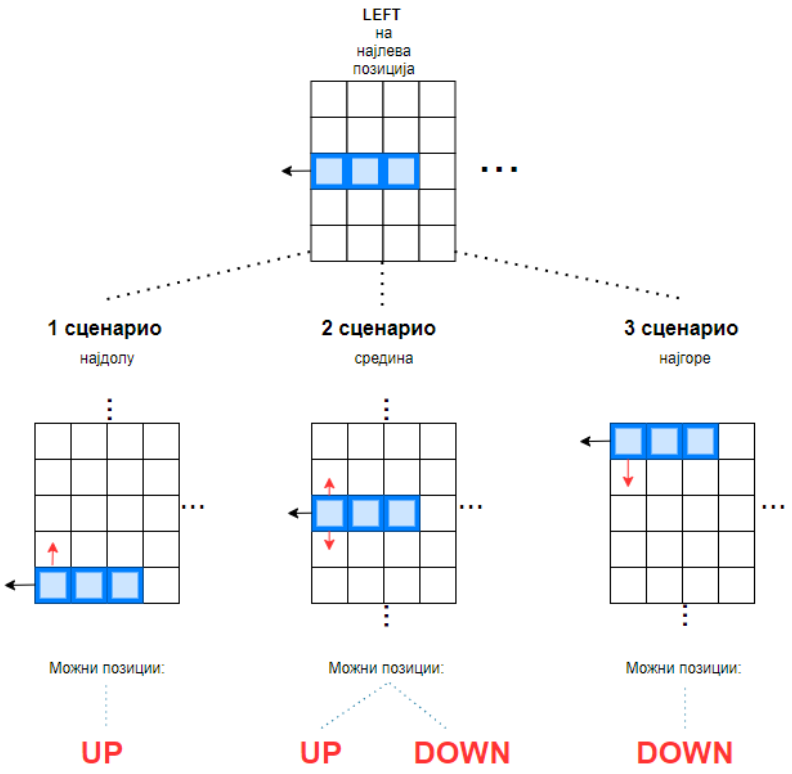


4 УСЛОВИ. 12 Сценарија.

Ова се сите можни “сценарија” предизвикани од нашите 4 “критични ситуации”. Доаѓа нешто како дрво на одлука. (ЛЕВО)

Црвот во скиците го направив да биде плав, гледав истата боја да е и во играта (ДЕСНО).

1 УСЛОВ: Добиена случајна избрана насока за втор црв LEFT а се наоѓа на <u>најлева</u> позиција



1 УСЛОВ. 1 Сценарио: *најдолу*

Ова е доколку црвот е најлево и во исто време најдолу, тука насока LEFT или DOWN нема логика бидејќи изглегува од игра, а насока RIGHT ќе има два дупликата сегмента и ја намалува должината на некој начин (барем визуелно). Од каде можна насока е **UP**

1 УСЛОВ. 2 Сценарио: *средина*

Ова е доколку црвот е најлево но не е ниту најгоре ниту најдоле. Тука врз основа на случаен избор се дава една од двете можни логички насоки: **UP** или **DOWN**

1 УСЛОВ. 3 Сценарио: *најдолу*

Идентично на првото само најгоре се наоѓа црвот. UP не е логично што значи единствена можна насока е **DOWN**

Овој услов искодиран изгледа вака:

```
121             if direction2 == 'left' and position_x == 0:  # најлево
122                 if position_y == 0:  # најлево, најдолу
123                     direction2 = UP
124                     break
125                 elif position_y == (CELLHEIGHT-1):  # најлево, најгоре
126                     direction2 = DOWN
127                     break
128                 else:  # најлево, но некаде на средина во игра
129                     direction2 = random.choice([UP, DOWN])
130                     direction2 = directions_dict[direction2]
131                     break
```

1 УСЛОВ. 1 Сценарио: Доколку сме најлево и најдолу даваме UP како следна насока и го прекинуваме циклусот кој е всушност додека не најдеме насока.

1 УСЛОВ. 2 Сценарио: Доколку сме во средина правиме листа од двете можни насоки за да имаме 50-50 шанси и со функцијата random.choice() добиваме една од тие две и повторно прекинуваме.

1 УСЛОВ. 3 Сценарио: Доколку сме најлево и најгоре даваме насока DOWN како следна и го прекинуваме циклусот.

2 УСЛОВ: Добиена случајна избрана насока за втор црв RIGHT а се наоѓа на <u>најдесна</u> позиција

2 УСЛОВ. 1 Сценарио: *најдолу*

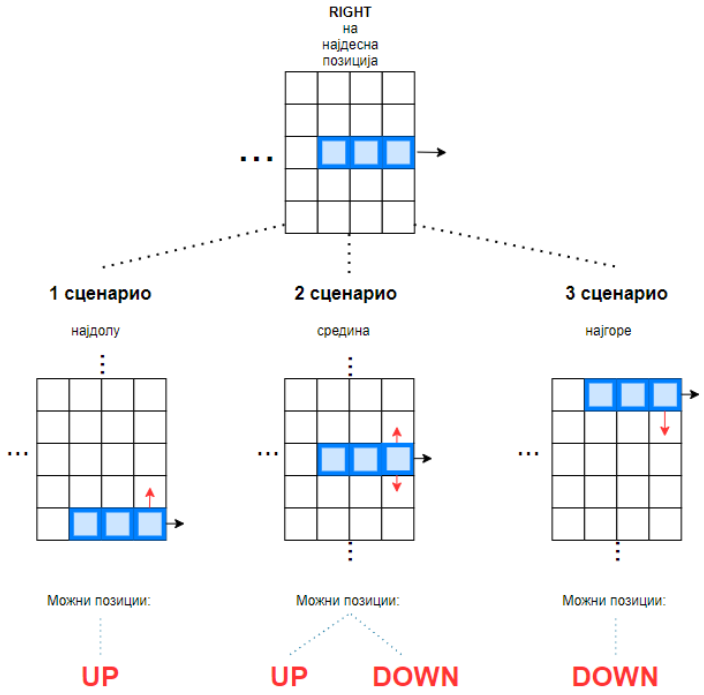
Ова е доколку црвот е најдесно и во исто време најдолу, исто како 1 УСЛОВ, и тука насока LEFT или DOWN немаат логика бидејќи изглегува од игра, а насока LEFT ќе има два дупликата сегмента и ја намалува должината на некој начин (барем визуелно). Од каде можна насока е **UP**

2 УСЛОВ. 2 Сценарио: *средина*

Ова е доколку црвот е најдесно но не е ниту најгоре ниту најдоле. Тука врз основа на случаен избор се дава една од двете можни логички насоки: **UP** или **DOWN**

2 УСЛОВ. 3 Сценарио: *најдолу*

Идентично на првото само најгоре се наоѓа црвот. UP не е логично што значи единствена можна насока е **DOWN**



Овој услов искодиран изгледа вака:

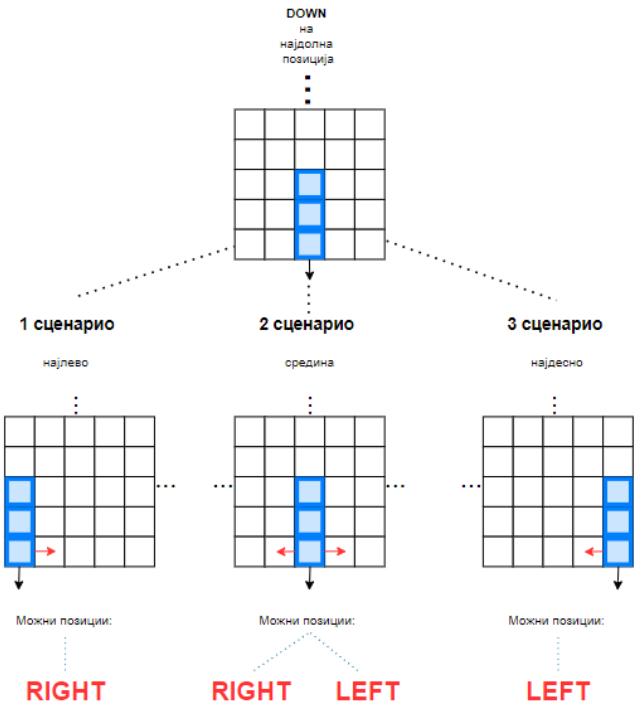
```
132                 # најдесно
133                 elif direction2 == 'right' and position_x == (CELLWIDTH-1):  # најдесно
134                     if position_y == 0:  # најдесно, најдолу
135                         direction2 = UP
136                         break
137                     elif position_y == (CELLHEIGHT-1):  # најдесно, најгоре
138                         direction2 = DOWN
139                         break
140                     else:  # најдесно, но некаде на средина во висина
141                         direction2 = random.choice([UP, DOWN])
142                         direction2 = directions_dict[direction2]
143                         break
```

2 УСЛОВ. 1 Сценарио: Доколку сме најдесно и најдолу даваме UP како следна насока и го прекинуваме циклусот.

2 УСЛОВ. 2 Сценарио: Доколку сме во најденсо во средина правиме листа од двете можни насоки, случајно бираме и повторно прекинуваме.

2 УСЛОВ. 3 Сценарио: Доколку сме најдесно и најгоре даваме насока DOWN како следна и го прекинуваме циклусот.

3 УСЛОВ: Добиена случајна избрана насока за втор црв DOWN а се наоѓа на <u>најдолна</u> позиција



<u>3 УСЛОВ. 1 Сценарио:</u> *најлево*

Ова е доколку црвот е најдоле и во исто време најлево слично како претходните услови ги бараме логичните наски тука е само една а таа е десно со што следна насока е **RIGHT**

<u>3 УСЛОВ. 2 Сценарио:</u> *средина*

Ова е доколку црвот е најдолу но не допира ѕидови и врз основа на случаен избор се дава една од двете можни логични насоки: **RIGHT** или **LEFT**

<u>3 УСЛОВ. 3 Сценарио:</u> *најдесно*

Идентично на првото само сме најдолу најдесно што значи единствена можна насока е **LEFT**

Овој услов искодиран изгледа вака:

```
156                 # најдолу
157                 elif direction2 == 'down' and position_y == (CELLHEIGHT-1):   # најдолу
158                     if position_x == 0:  # најдолу, најлево
159                         direction2 = RIGHT
160                         break
161                     elif position_x == (CELLWIDTH-1):  # најдолу, најдесно
162                         direction2 = LEFT
163                         break
164                     else:
165                         direction2 = random.choice([LEFT, RIGHT])
166                         direction2 = directions_dict[direction2]
167                         break
```

Иако на прв поглед изгледа грешно, не е. Обратно се се горе и долу во кодирање, ова е зависност од како е поставен кординатниот ситем.

<u>3 УСЛОВ. 1 Сценарио:</u> Доколку сме најдолу и најлево даваме RIGHT како следна насока и го прекинуваме циклусот.

<u>3 УСЛОВ. 2 Сценарио:</u> Доколку сме во најдолу во средина правиме листа од двете можни насоки, случајно бираме и повторно прекинуваме (RIGHT или LEFT)

<u>3 УСЛОВ. 3 Сценарио:</u> Доколку сме најдолу и најдесно даваме насока LEFT како следна и го прекинуваме циклусот.

4 УСЛОВ: Добиена случајна избрана насока за втор црв UP а се наоѓа на најгорна позиција

4 УСЛОВ. 1 Сценарио: *најлево*

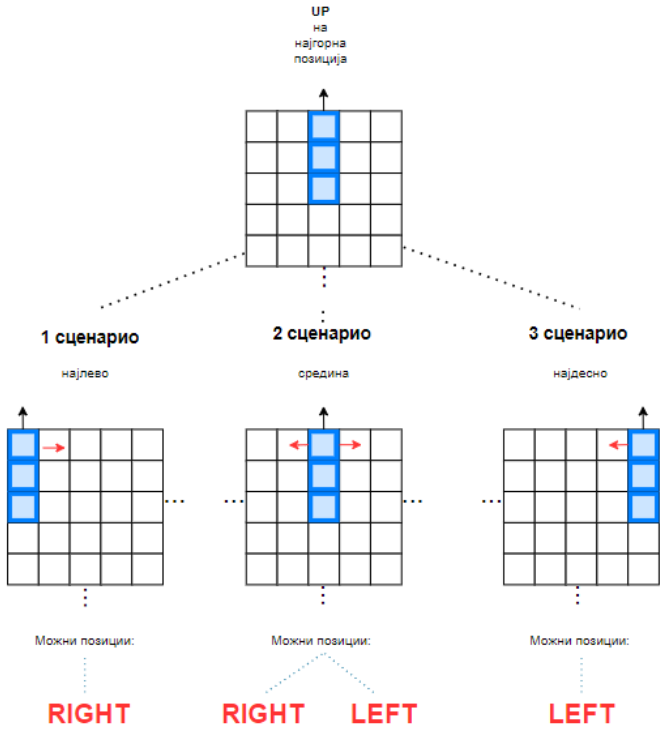
Ова е доколку црвот е најгоре и во исто време најлево слично како претходниот услов ги бараме логичните наски тука е само една а таа е десно со што следна насока е **RIGHT**

4 УСЛОВ. 2 Сценарио: *средина*

Ова е доколку црвот е најгоре но не допира ѕидови и врз основа на случаен избор се дава една од двете можни логични насоки: **RIGHT** или **LEFT**

4 УСЛОВ. 3 Сценарио: *најдесно*

Идентично на првото само сме најгоре најдесно што значи единствена можна насока е **LEFT**



Овој услов искодиран изгледа вака:

```
144                 # најгоре
145                 elif direction2 == 'up' and position_y == 0:  # најгоре
146                     if position_x == 0:  # најгоре, најлево
147                         direction2 = RIGHT
148                         break
149                     elif position_x == (CELLWIDTH-1): # најгоре, најдесно
150                         direction2 = LEFT
151                         break
152                     else:
153                         direction2 = random.choice([LEFT, RIGHT])
154                         direction2 = directions_dict[direction2]
155                         break
```

4 УСЛОВ. 1 Сценарио: Доколку сме најгоре и најлево даваме RIGHT како следна насока и го прекинуваме циклусот.

4 УСЛОВ. 2 Сценарио: Доколку сме во најгоре во средина правиме листа од двете можни насоки, случајно бираме и повторно прекинуваме (RIGHT или LEFT)

4 УСЛОВ. 3 Сценарио: Доколку сме најгоре и најдесно даваме насока LEFT како следна и го прекинуваме циклусот.

Со ова ги завршуваме сите можни 12 сценарија и доколку не влезе ни во еден од условите и не го прекине циклусот завршува со доделување на иницијалната насока од случајниот избор и потоа го прекинува циклусот (ред 170 – 171)

```
113     if flag == 'shown':
114         flag_direction = True
115         while flag_direction:
116             direction2 = random.choice(directions)
117             # најлево
118             position_x = wormCoords2[0]['x']
119             position_y = wormCoords2[0]['y']
120             # најлево
121             if direction2 == 'left' and position_x == 0:...
132             # најдесно
133             elif direction2 == 'right' and position_x == (CELLWIDTH-1):...
144             # најгоре
145             elif direction2 == 'up' and position_y == 0:...
156             # најдолу
157             elif direction2 == 'down' and position_y == (CELLHEIGHT-1):...
168 
169             # ако нема конфликтни ситуации, врати ја вредноста што е доделена прва
170             direction2 = directions_dict[direction2]
171             flag_direction = False
```

Црвот потоа се движи идентично како првиот црв со следниот блок комнади подоле во кодот:

```
216     if flag == 'shown':
217         # Втор црв движење
218         if direction2 == UP:
219             newHead2 = {'x': wormCoords2[HEAD2]['x'], 'y': wormCoords2[HEAD2]['y'] - 1}
220         elif direction2 == DOWN:
221             newHead2 = {'x': wormCoords2[HEAD2]['x'], 'y': wormCoords2[HEAD2]['y'] + 1}
222         elif direction2 == LEFT:
223             newHead2 = {'x': wormCoords2[HEAD2]['x'] - 1, 'y': wormCoords2[HEAD2]['y']}
224         elif direction2 == RIGHT:
225             newHead2 = {'x': wormCoords2[HEAD2]['x'] + 1, 'y': wormCoords2[HEAD2]['y']}
226 
```

И доколку е прикажан црвот се додава во листата со insert методот на почетокот новата глава доколку не се избрише сегмент (тоа го објаснувам подлоу)

```
237         if flag == 'shown':
238             wormCoords2.insert(0, newHead2)
239             drawWorm(wormCoords2, DARKBLUE, LIGHTBLUE)
240             # print(wormCoords2)
```

Поголемиот дел од дебагирањето за дали е океј потегот го правев со принтање на листата од речници за црвот (ред 240).

Колизија на црвите

Следно за имплементација е што се случува доколку црвите имаат колизија некаква. Имаме од самиот опис на барањето 2 типа:

- **Тип 1 колизија:** Главата од прв црв го допира втор црв (расте сегмент)
- **Тип 2 колизија:** Втор црв го допира првиот црв со било кој сегмент било каде по телото на првиот (расте сегмент)

Притоа многу важно доколку имаме колизија од 1 тип, црвот расте но SCORE останува **непроменет**. Затоа подоцна имаме бројач кој брои дали имаме колизија од тип 1 за да минусира.

Тип 1 колизија

Ова е доста едноставна проверка гледаме дали со својата глава привот црв го допира вториот некаде, доколку да поставуваме знамеце **hit_flag = False**. Кога ќе првиот го допира со својата глава вториот тогаш знамето е **True**.

Искодирано ова изгледа вака:

```
182         hit_flag = False  # знаме за проверка дали 1 црв го допира 2 (некаде по телото)
183         for wormBody in wormCoords2:
184             if wormCoords[HEAD]['x'] == wormBody['x'] and wormCoords[HEAD]['y'] == wormBody['y']:
185                 hit_flag = True
186                 break
```

Тип 2 колизија

Оваа ја имав замислено како матрица каде гледаме дали секој сегмент од вториот црв е еднаков на секој сегмент од првиот црв (матрица на соседство ако би се проверувало кај графови). Ако видиме дека е еднаков тогаш ново знамеце jump_flag од False го поставуваме на True.

Искодирано ова изгледа вака:

```
173         jump_flag = False  # знаме за проверка дали 2 црв го допира 1
174         for wormBody in wormCoords:  # врти ги сите сегменти на прв црв
175             for wormBody2 in wormCoords2:  # врти ги сите сегменти на втор црв
176                 if wormBody['x'] == wormBody2['x'] and wormBody['y'] == wormBody2['y']:  # проверка
177                     jump_flag = True
178                     break
179             if jump_flag:
180                 break
181
```

Сега зошто ни се потребни знамецата е повеќе кај проверките за бришење / оставање на сегменти. Имаме неколку можни ситуации:

- Прв црв до допира втор црв со глава со што автоматски втроиот црв го допира првиот:
  двата црва растат сегмент (т.е. не им се брише сегмент од крајот)
- Втор црв го допира првиот црв со своите други сегменти:
  вториот расте сегмент (т.е. на првиот се намалува сегмент од крај)
- Ако е прикажан вториот и нема колизии:
  ни еден не расте сегмент(т.е. и двата се намалуваат за сегмент)

Ова искодирано изгледа вака (**score_sub:** се зголемува кога првиот со главата го допира вториот и притоа го намалува резултатот за толку со цел да ја долови суштината, плавиот расте зелениот расте но резултатот е ист):

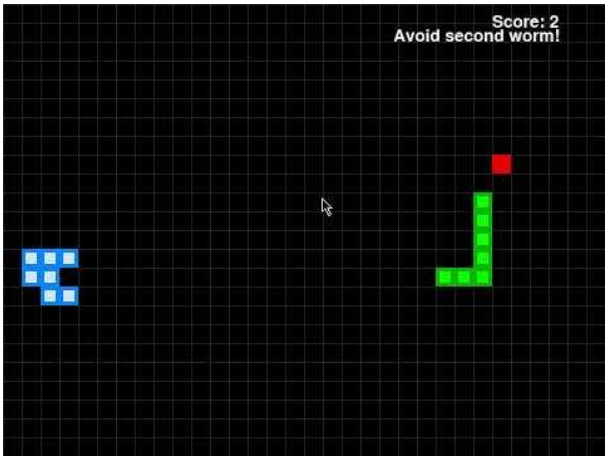
```
        # проверки за колизии
        if wormCoords[HEAD]['x'] == apple['x'] and wormCoords[HEAD]['y'] == apple['y']:
            # don't remove worm's tail segment
            apple = getRandomLocation()# set a new apple somewhere

        elif flag == 'shown' and jump_flag:  # проверка дали 2 го допира прв било каде
            if hit_flag:# тука уствари и 2та црва се допријаат, 1иот со глава, другиот со било кој сег
                # pass  # двата црва растат за сегмент
                score_sub += 1
            else:
                del wormCoords[-1]  # 2иот го допира 1иот продолжуваме за 1, но 1иот се намалува

        elif flag == 'shown':
            del wormCoords[-1]
            del wormCoords2[-1]  # remove worm's tail segment

        else:
            del wormCoords[-1]
```

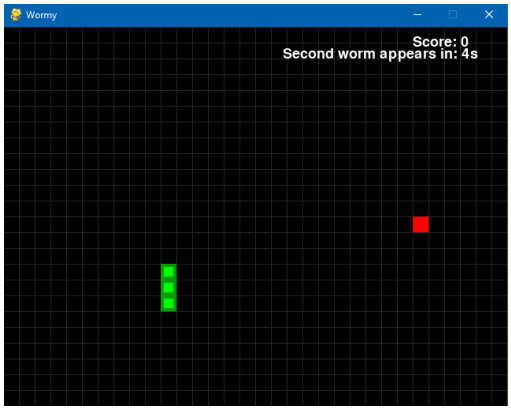
Со цел визуелно да ви покажам како работи кодот, снимив видео и го прикачив на YouTube. Линкот до видеото е следен:

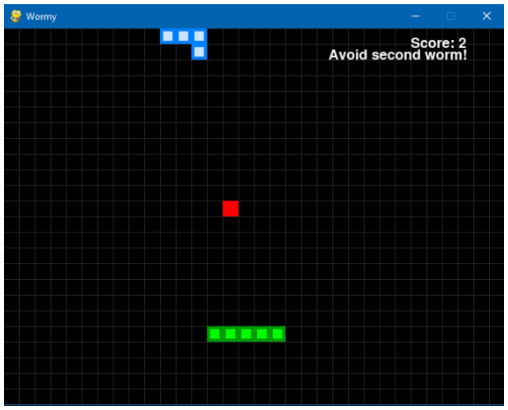


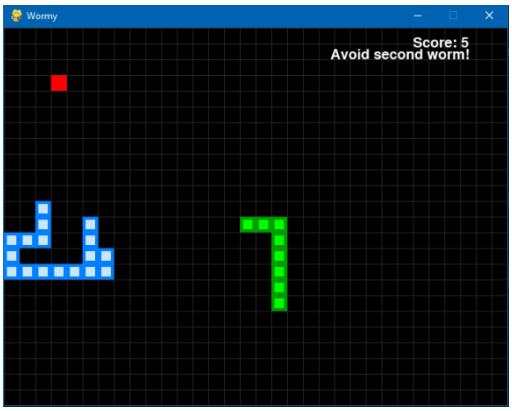
https://youtu.be/Z6ZnSMSbjZM

NOTE: Намалив тајмерот на видеото да почнува од 5 секунди место од 45 со цел да не прикачувам големо видео. Во крајниот код прикачувам со 45 секунди. Истотака на крајот од видеото по game over, повторно почнувам со цел да покажам дека секој пат се ресетира и големината на црвот и тајмерот за истиот.

Неколку screenshots од ова барање се во прилог:







2. ***Барање 2***

Додадете елементи што трепкаат (секој пат по три) на случајно избрани позиции со димензија од едно квадратче. Тие се појавуваат во времетраење од 5 секунди. Ако оригиналниот црв ги изеде, играчот добива дополнителни поени (по 2 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.

Ова барање е слично (барем јас така го имплементирав) како жолтото јаболко и плавото јаболко само што наместо едно ќе имаме повеќе.

Идејата е да се има посебни бројачи, почетни времиња за двата тајмери. Секој пат кога ќе се изеде елемент се генерира нов и ова иде со трепкање од 5 секунди (5 покажува, 5 не покажува).

Земи почетно време за елементи, намести бројач и постави знаменце:

```
78      # Тајмер црв
79      start_ticks_worm = pygame.time.get_ticks()  # start ticks црв
80      counter = 45  # додај бројач за нов црв 45сек
81      flag = 'hidden'  # знаме за црв; 'hidden'-скриено, 'shown'-прикажано
82
83      # Тајмер елементи
84      start_ticks_ele = pygame.time.get_ticks()  # start ticks елементи
85      counter_elements = 5 # додај бројач за елементи 5сек
86      flag_elements = 'hidden'
87
```

Земи три рандом променливи со рандом локации (технички треба да се проверува дали се покплопуваат две рандом локации но ја пуштив играта повеќе пати и немав 2 елемента на иста локација, ако има плус 4 поени за среќа):

```
92      # дефиниција на елементи
93      point_element1 = getRandomLocation()  # креирам прв елемент за плус поени
94      point_element2 = getRandomLocation()  # креирам прв елемент за плус поени
95      point_element3 = getRandomLocation()  # креирам прв елемент за плус поени
96
```

Дефинирај променливи за резултат:

- **Score_sub:** брои колку пати сме се удриле со плав црв за да намали
- **Score_plu:** брои колку плави елементи кои носат 2 поени сме изеле

```
97      # минусирање од резултат
98      score_sub = 0
99      # плус поени за резултат
100     score_plu = 0
```

Сега на самиот почеток од главниот циклус, ги зимаме времињата за тајмерот да работи. Притоа треба да се посебни бидејќи ако имаме start_ticks за двата исто тогаш едниот од бројачите најчесто тој што прв истекол ќе го ресетира другиот.

```
while True:# main game loop
    # земи време за црв
    seconds_worm = (pygame.time.get_ticks() - start_ticks_worm) / 1000
    seconds_worm = int(seconds_worm)
    # земи време за елементи
    seconds_ele = (pygame.time.get_ticks() - start_ticks_ele) / 1000
    seconds_ele = int(seconds_ele)

```

Потоа имаме проверка дали сме изеле едно од јаболките, ако сме изеле зголемуваме score_plu += 1:

```
# Проверки за плус елементи
if wormCoords[HEAD]['x'] == point_element1['x'] and wormCoords[HEAD]['y'] == point_element1['y']:
    score_plu += 1
    point_element1 = getRandomLocation()  # го изел првиот елемент, додели нова локација

elif wormCoords[HEAD]['x'] == point_element2['x'] and wormCoords[HEAD]['y'] == point_element2['y']:
    score_plu += 1
    point_element2 = getRandomLocation()  # го изел вториот елемент, додели нова локација

elif wormCoords[HEAD]['x'] == point_element3['x'] and wormCoords[HEAD]['y'] == point_element3['y']:
    score_plu += 1
    point_element3 = getRandomLocation()  # го изел третиот елемент, додели нова локација
```

```
if flag_elements == 'shown':
    drawApple(point_element1, GLAUCOUS)
    drawApple(point_element2, GLAUCOUS)
    drawApple(point_element3, GLAUCOUS)

```

Потоа имаме проверка дали истекол тајмерот, ако да ги покажува ако не ги скока.

Ова е доста слично како со знамето на првиот црв, разликата е само оваа функција:

```
# 45 секунди за втор црв функција
counter, start_ticks_worm, flag = drawSecondWormTime(counter,
                                                     seconds_worm,
                                                     start_ticks_worm,
                                                     flag)
# 5s on / 5s off
counter_elements, start_ticks_ele, flag_elements = drawElementsTime(counter_element
                                                                    seconds_ele,
                                                                    start_ticks_ele
                                                                    flag_elements)
```

Разликата е внатре во самата drawElementsTime() е тоа што сакаме периодично прикажување а во drawSecondWormTime() чекаме 45 и потоа цело време го имаме.

Формулата која е користам за пресметување на резултатот е следна:

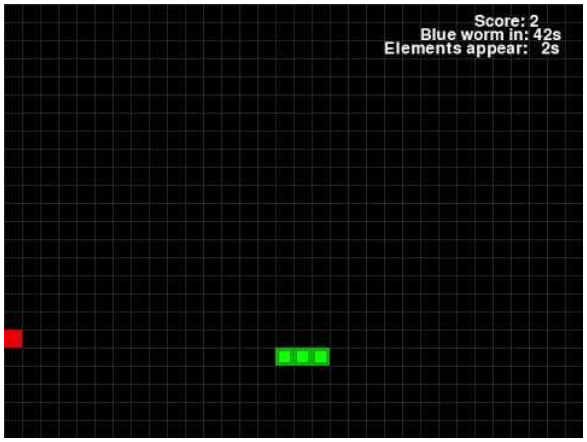
```
299         # пресметај резултат
300         drawScore(len(wormCoords) - 3 - score_sub + score_plu*2)
301
```

Каде:

- **len(wormCoords):** Вкупна должина на црв
- **-3:** ова 3ка се почетните 3 сегменти од почетокот на играта
- **score_sub:** ова е бројач колку сегменти се додадени каде играчот до удира плавиот црв
- **score_plu*2:** го множиме бројот на изедени елементи * 2 за да го зголемуваме резултот за 2 на секој изеден елемент

На следниот линк има видео каде барање 1 и барање 2 ги испробав:

https://www.youtube.com/watch?v=4AVPHMAo6qE



3. ***Барање 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.

Копчињата ги додавам во главната игра во функцијата каде се испишува Game Over.

Слично како game и over, додавам surface и rect објекти и потоа проверувам слично како во Slide Puzzle дали имаме rect.colidepoint(event.pos) доколку имаме извршуваме што и да имаме во условот.

Модифицираната функција:

```
475     def showGameOverScreen():
476         gameOverFont = pygame.font.Font('freesansbold.ttf', 150)
477         basicFont = pygame.font.Font('freesansbold.ttf', 40)
478         gameSurf = gameOverFont.render('Game', True, WHITE)
479         overSurf = gameOverFont.render('Over', True, WHITE)
480         gameRect = gameSurf.get_rect()
481         overRect = overSurf.get_rect()
482         gameRect.midtop = (WINDOWWIDTH / 2, 10)
483         overRect.midtop = (WINDOWWIDTH / 2, gameRect.height + 10 + 25)
484 
485         startSurf = basicFont.render('Start from the beginning', True, YELLOW)
486         quitSurf = basicFont.render('Quit', True, YELLOW)
487         startRect = startSurf.get_rect()
488         quitRect = quitSurf.get_rect()
489 
490         startRect.midtop = (WINDOWWIDTH / 2, 320)
491         quitRect.midtop = (WINDOWWIDTH / 2, 360)
492 
493         DISPLAYSURF.blit(gameSurf, gameRect)
494         DISPLAYSURF.blit(overSurf, overRect)
495 
496         DISPLAYSURF.blit(startSurf, startRect)
497         DISPLAYSURF.blit(quitSurf, quitRect)
498 
499         drawPressKeyMsg()
500         pygame.display.update()
501         pygame.time.wait(500)
502         checkForKeyPress()
503 
504         while True:
505             for event in pygame.event.get():  # event handling loop
506                 if event.type == MOUSEBUTTONUP:
507                     if startRect.collidepoint(event.pos):
508                         show_flag = False
509                         main(show_flag)
510                     elif quitRect.collidepoint(event.pos):
511                         terminate()
512                 if checkForKeyPress():
513                     pygame.event.get() # clear event queue
514                     return
515 
```

Во ред 509: ја повикувам главната функција но со flag. Ова знамеце игра улога во дали текстот што се врти ќе се испише или не.

Во ред 510: ја терминираме играта ако играчот кликне “Quit”

Знаменецто во главната функција го имплементирам вака:

```
46  show_flag = True
47
48  def main(flag=True):
49      global FPSCLOCK, DISPLAYSURF, BASICFONT
50
51      pygame.init()
52      FPSCLOCK = pygame.time.Clock()
53      DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))
54      BASICFONT = pygame.font.Font('freesansbold.ttf', 18)
55      pygame.display.set_caption('Wormy')
56
57      if flag:
58          showStartScreen()
59      while True:
60          runGame()
61          showGameOverScreen()
```

Секој пат кога ќе биде кликнато “Start from beginning” ќе се смени вредноста и ќе го скока прикажувањето на текстот.

Како и за другите барање снимив видео демонстрација и за ова (иако е поедноставно):

https://youtu.be/-sYOPGjvNxY