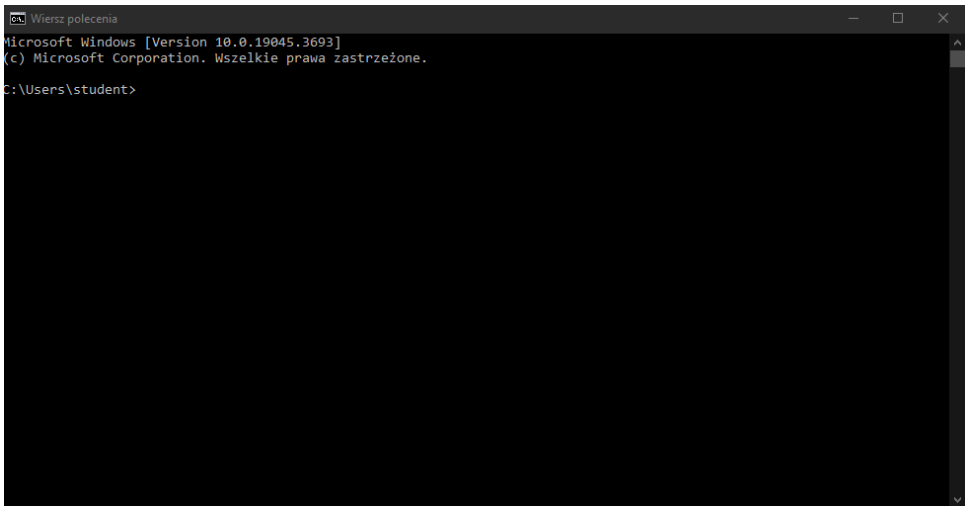
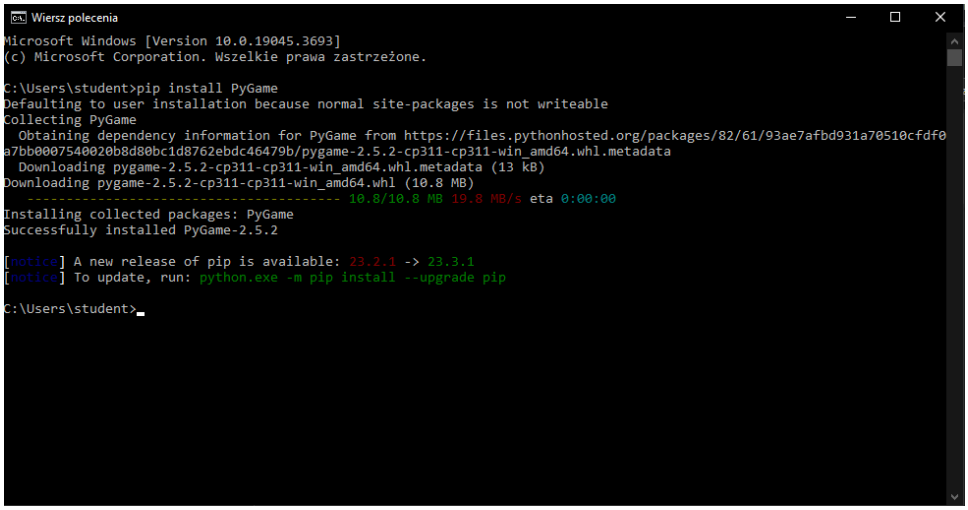
Sprawozdanie

Zad 1

Celem pierwszego zadanie było zainstalowanie biblioteki PyGame, używanej do tworzenia gier za pomocą języka Python. Aby zainstalować PyGame musimy otworzyć konsole cmd w systemie windows.



Python ma swój własny menadżer bibliotek nazywane pip. Aby zainstalować bibliotekę wystarczy napisać “pip install PyGame”. Otrzymujemy:

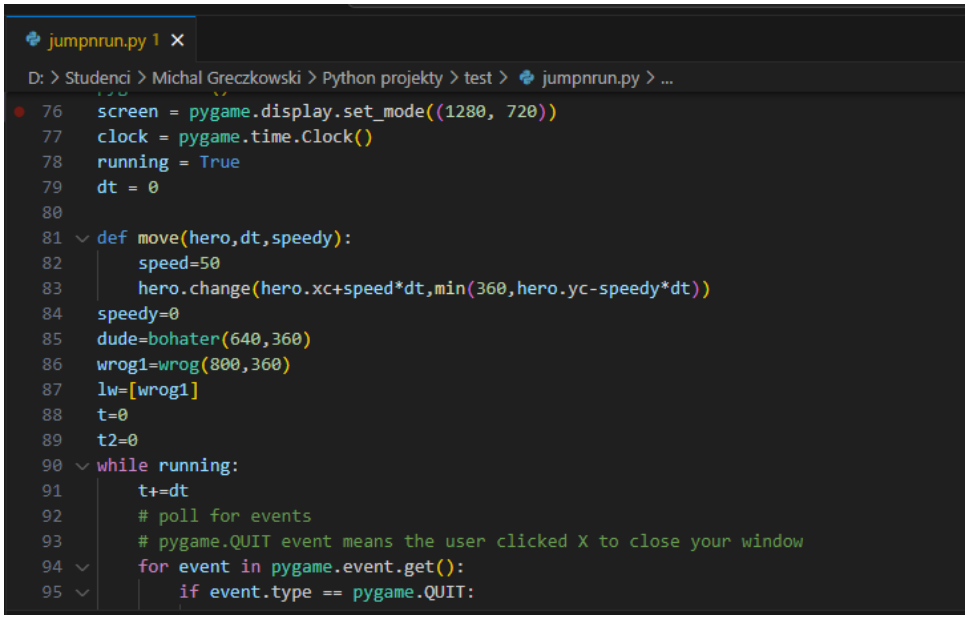


W ten sposób zainstalowaliśmy PyGame.

Zad 2

Celem zadania było utworzenie poziomu zawierającego kilku przeciwników w losowych pozycjach.

Otwieramy podstawowy program w visual studio:



Widzimy, że wrogowie są obiektem typu wrog, gdzie do obiektu przekazujemy pozycje x i y tam, gdzie chcemy, żeby został narysowany. W takm razie wystarczy wygenerować losowo pozycje, najlepiej zrobić to za pomocą polecenia randint zawartego w bibliotece random. Importujemy bibliotekę:

```
11  import random
```

Następnie, zamiast tworzyć osobno wrogów, możemy zautomatyzować ich tworzenie za pomocą funkcji:

```
120 def GenerateEnemies(amount):
121     templist = []
122     for i in range(amount):
123         e = 260 + (i*200)
124         templist.append(wrog(random.randint(e,1260), 360))
125     return templist
126
```

W funkcji umieszczono od razu metode randint, w ten sposób wrogowie zostaną utworzeni w różnych miejscach. Rysowanie wrogów również zautomatyzowano:

```
142 def DrawListOfObjects(list):
143     for i in list:
144         i.draw(screen)
```

W głównej pętli obsługującej zachowanie się programu wystarczy tylko użyć tych funkcji w następujący sposób:

```
177 lw=GenerateEnemies(5)
```

```
DrawListOfObjects(lw)
```

Zad 3

Celem zadania było sprawienie, aby wrogowie zaczęli się poruszać. Do tego celu wykorzystano metodę “change” z klasy “bohater”.

```
82      def change(self,xc,yc):
83          self.xc=xc
84          self.yc=yc
85          self.x1=self.xc-self.a
86          self.x2=self.xc+self.a
87          self.y1=self.yc-self.b
88          self.y2=self.yc+self.b
```

Teraz wystarczy użyć wcześniej zdefiniowanej funkcji move:

```
168 def move(hero,dt,speedy, speed):
169     hero.change(hero.xc+speed*dt,min(360,hero.yc-speedy*dt))
```

Została ona zmieniona w taki sposób, że można podać teraz speed przywołując tą funkcję, teraz aby poruszać wszystkimi obiektami można utworzyć funkcję używającą tej wczesniejszej funkcji:

```
156 def MoveEnemies(list):
157     for i in list:
158         move(i, dt, 0, random.randint(-50, -1))
```

W pętli odpowiadającej za działanie programu wystarczy teraz tą funkcje tylko przywołać.

```
238     MoveEnemies(lw)
```

Wrogowie powinni się teraz poruszać.

Zad 4

Celem zadania było wprowadzenie nowego obiektu, na który będzie mógł wspinać się bohater.

W przypadku tej klasy, nie potrzebujemy definiować żadnych skomplikowanych metod, wystarczy, że utworzymy konstruktor:

```
106 class platforma(rect):
107     def __init__(self, xc, yc):
108         self.a=100
109         self.b=3
110         self.xc=xc
111         self.yc=yc
112         rect.__init__(self,self.xc-self.a,self.yc-self.b,self.xc+self.a,self.yc+self.b,"green")
113         self.type = "platforma"
```

Aby wygenerować te platformy utworzono funkcję.

```
135 def GeneratePlatforms(amount):
136     templist = []
137     for i in range(amount):
138         e = 660 + (i*200)
139         templist.append(platforma(random.randint(e,1260), 340))
140     return templist
```

Pomimo tego, że funkcja ta przypomina tą użytą we wcześniejszym zadaniu, przez ograniczenia pygame nie ma możliwości połączenia tych funkcji w jedność.

Mając wygenerowane platformy możemy je narysować za pomocą funkcji DrawListOfObjects. Teraz musimy tylko utworzyć kolizje pomiędzy bohaterem a platformą. W klasie bohater z tego powodu definiujemy nowe metody, touch_platform będzie sprawdzała, czy bohater dotyka platformy, a check_status_platform będzie sprawdzała, czy statusem jest dotknięcie.

```
50      def touch_platform(self,other):
51          if other.type == "platforma":
52              return ((abs(self.xc-other.xc)<self.a+other.a) and self.yc<other.yc)
```

```
59      def check_status_platform(self, listofplatforms):
60          touched=False
61          a = 0
62          for i in listofplatforms:
63              touched=touched or self.touch_platform(i)
64              a+=1
65          if touched:
66              self.status="on_platform"
67              return a
```

Teraz wystarczy tylko przywołać wymagane metody a następnie sprawdzić status bohatera

```
212     dude.check_status_platform(list_of_platforms)
```

```
225     if dude.status=="on_platform":
226         speedy=0
227         dude.yc = 325
```

Przypisując bohaterowi yc = 325 możemy być pewni, że nie zatnie się on w środku platformy.

W ten sposób, utworzyliśmy platformy, narysowaliśmy je i utworzyliśmy sposób na wykrycie ich kolizji, bohater więc może na nie wskakiwać, i nie będzie spadał przez nie.

Zad 5

Celem zadania było utworzenie monet w grze. Tworzymy klasę moneta. W jej przypadku użyjemy metod znajdującej się w klasie bohater. W ten sposób nie musimy sprawdzać znowu obkiet bohatera, możemy sprawdzić poszczególne monety, czy nie były dotknięte.

```
90  class moneta(rect):
91      def __init__(self, xc, yc):
92          self.a=10
93          self.b=10
94          self.xc=xc
95          self.yc=yc
96          rect.__init__(self,self.xc-self.a,self.yc-self.b,self.xc+self.a,self.yc+self.b,"yellow")
97          self.status="notcollected"#notcollected,collected
98      def touch(self,other):
99          return ((abs(self.xc-other.xc)<self.a+other.a) and self.y2>=other.y1)
100     def check_status(self,hero):
101         touched = False
102         touched = touched or self.touch(hero)
103         if touched:
104             self.status="collected"
```

Oczywiście znowu tworzymy funkcje tworzącą nam listę tych obiektów

```
128  def GenerateCoins(amount):
129      templist = []
130      for i in range(amount):
131          e = 260 + (i*100)
132          templist.append(moneta(random.randint(e,1260), random.randint(300, 340)))
133      return templist
```

Tworzymy też funkcje, która będzie sprawdzała status wszystkich monet:

```
145  def CheckStatusOfCoins(list_of_objects, play_object, score):
146      a = 0
147      for i in list_of_objects:
148          i.check_status(play_object)
149          if i.status == "collected":
150              list_of_objects.pop(a)
151              score += 50
152          a += 1
153      return score
```

Funkcja ta też odpowiada, za usuwanie monet, kiedy mają status “collected” , oraz dodaniu punktów. Teraz wystarczy umieścić parę poleceń w głównej pętli. W ten sposób będziemy mogli wypisywać punkty dla gracza, narysować monety oraz sprawdzać, czy zostały dotknięte.

```
176  list_of_coins = GenerateCoins(5)
```

```
179  score = 0
180  font = pygame.font.Font(None, 64)
```

```
207      DrawListOfObjects(list_of_coins)
```

```
213      score = CheckStatusOfCoins(list_of_coins, dude, score)
```

```
203      score_text = font.render("Score:" + str(score), True, (5, 5, 5))
204      score_text_pos = score_text.get_rect(centerx=100, y=10)
205      screen.blit(score_text, score_text_pos)
```

Ostatni zrzut ekranu przedstawia rysowanie tekstu wraz z wynikiem dla gracza.

W ten sposób utworzono nową klasę obiektu, która jest w stanie być zebrana, generuje się w losowych pozycjach i kiedy zostanie zebrana dodaje 50 punktów i jest usuwana z gry.

Zad 6

Celem zadania było wprowadzenie celu wygranej w grze. W tym przypadku cel ustawiono na dojście do końca poziomu (prawy koniec ekranu). Wystarczyło więc sprawdzić, czy bohater znajduje się na krańcu ekranu, a następnie wypisać tekst w sposób podobny do napisu game over. Cały kod znajduje się w pętli głównej.

```
229      if dude.xc >= 1280:
230          text = font.render("You win!", True, (10, 10, 10))
231          textpos = text.get_rect(centerx=640, y=10)
232          screen.blit(text, textpos)
233          pygame.display.flip()
234          pygame.time.delay(1000*2)
235          running=False
```

Zad 7

Celem zadania było wprowadzenie realistycznego skoku. W tym przypadku skoku parabolicznego, musimy więc najpierw wprowadzić stałą grawitacyjną

```
182    g = 30
```

W tym przypadku stała jest bardzo duża, tak aby bohater nie skakał zbyt wysoko i w miare szybko opadał. Następnie wystarczy trochę przerobić wcześniej używanego polecenia if w następujący sposób:

```
196     keys = pygame.key.get_pressed()
197     if keys[pygame.K_UP] and dude.yc==360:
198         dt2=dt
199         speedy = 50
200     if speedy > 0 or dude.yc!=360:
201         speedy -= g*dt
```

Oczywiście speedy odpowiada naszej wartości vy a jak wiadomo, w ruchu parabolicznym wartość składowej vy = vy0 – g*t . Oczywiście w programowaniu t musimy uznać jako zmiane czasu, inaczej nie zauważylibyśmy ruchu.

W ten też sposób postać wykonuje skok paraboliczny.