Bosna i Hercegovina
Federacija Bosne i Hercegovine

Unsko – sanski kanton
Grad Cazin

JU „Gimnazija“ Cazin

Maturski rad

Igranje računalnih igara neuroevolucijom promjenjivih topologija

Cazin, 5. 4. 2022. godine

Mentor:
Silić Jasmin

Kandidat:
Škrgić Dado

Sadržaj

Sadržaj figura

1. Uvod

Moderne računalne igre su veoma kompleksne, realistične i u timskom okruženju. Kao posljedica, umjetna inteligencija u igrama je sve teži i zahtjevniji zadatak. Računalna inteligencija je obećavajuća tehnologija koja podržava razvoj umjetne inteligencije i poboljšava iskustvo igara. Dostupnost jeftinih računalnih resursa i nedavni radovi koji se bave neuroevolucijom [10, 11, 12] sugeriraju da su evolucionarne računalne tehnike učinkovite za razvoj zanimljivijih i privlačnijih računalnih igara. Pored toga, računalne igre su idealan testni scenario za evolucionarne tehnike jer pružaju kompleksno i realistično okruženje bez potrebe za skupim simulacijama i eksperimentima u stvarnom životu [13].
Zbog toga su u polju evolucionarnog računarstva organizirana znanstvena natjecanja posvećena računarskim igrama kao [14].
Ovaj rad se fokusira na primjenu Neuroevolucije promjenjivih topologija, poznatog neuroevolucijskog pristupa, za evoluciju neuralne mreže za igranje računalne igre Swing Copters, implementirane od nule.
Neuroevolucija, umjetna evolucija neuralnih mreza koja koristi genetske algoritme, je pokazala značajan potencijal u kompleksnim zadatcima podržanog učenja [1, 2, 3, 4, 5]. Neuroevolucija pretražuje prostor ponašanja neuralne mreže koja određeni zadatak obavlja dobro. Ovaj pristup rješavanju kompleksnih upravljačkih problema predstavlja alternativu statističkim metodama koje pokušavaju procijeniti korisnost pojedinih postupaka u određenim situacijama [6]. Neuroevolucija je prosperitetan pristup rješavanju problema podržanog učenja iz više razloga. Ranija istraživanja su pokazala da je neuroevolucija brži i učinkovitiji način strojnog učenja od drugih metoda podržanog učenja kao što su Prilagodljiva istraživačka kritika i Q-učenje [7, 8]. Budući da neuroevolucija traži ponašanje a ne vrijednost funkcije, učinkovita je u problemima sa kontinuiranim i visokodimenzionalnim stanjima prostora. Pored toga, pamćenje se jednostavno predstavlja kroz ponavljajuće veze u neuralnoj mreži, čineći neuroevoluciju prvim izborom za učenje nemarkovijanskih zadataka [1, 9].
U klasičnim neuroevolucijskim pristupima, topologija za evoluirajuću mrežu je izabrana prije samog početka eksperimenta. Mrežnu topologiju inače predstavlja skriveni sloj neurona gdje je svaki skriveni neuron povezan za svaki mrežni ulaz i svaki mrežni izlaz. Evolucija mreže traži prostor povezanih težina funkcije tako što dozvoljava reprodukciju mreža visokih performansi. Prostor težina se istražuje ukrštanjem mrežnih težina i mutacijom tereta jedne mreže. Prema tome, cilj neuroevolucije sa fiksnom topologijom je da optimizira veze težina koje određuju funkcionalnost mreže.
Unatoč tome i dalje ostaje osnovno pitanje: Imaju li promjenjive topologije zajedno sa težinama prednost nad promjenjivim težinama na fiksnoj topologiji?
Istraživači su u [2] prestavili uvjerljiv argument za evoluciju i topologija i težina, u kome su tvrdili da evoluirajuća struktura štedi vrijeme koje ljudi potroše na odluku o topologiji mreže za određeni neuroevolucijski problem. Iako većina neuroevolucijskih sistema koriste u potpunosti povezan skriveni sloj, odluka o broju skrivenih čvorova je proces pokušaja i pogreški. Podržali su svoj argument evolucijom topologije i težina umjetne neuralne mreže koja je riješila najteži referentni problem balansiranja motke do danas. Međutim, kasniji rezultati su zaključili da takva struktura nije bila potrebna za rješavanje teškog problema. Metoda sa fiksnom topologijom zvana Nadmetnuta podpopulacija [1] je riješila isti problem 5 puta brže tako što se je ponovno započinjala iznova sa nasumično generisanim brojem skrivenih neurona kad god bi zaglavila.

Na odabir teme su me potaknule činjenice da strojno učenje posljednjih godina bilježi ogroman rast industrijske primjene i samim tim predstavlja sve bitniji dio naših života. Od preporuka prilikom pretraživanja na internet tražilicama do autonomnih vozila strojno učenje je sve zastupljenije. Odlučio sam se specifičnije fokusirati na neuroevoluciju iz razloga što predstavlja jednostavniji oblik strojnog učenja za implementirati što mi je omogućilo da pored teorijskog dijela u radu i praktično predstavim mogućnosti strojnog učenja.

2. Neuroevolucija promjenjivih topologija

Neuroevolucija promjenjivih topologija je neuroevolucijska metoda dizajnirana da iskoristi prednosti strukture minimalizirajući dimenzionalnost prostora za pretragu povezanih težina. Predstavlja genetski algoritam koji generira umjetne neuronske mreže. Evolucijom algoritma mijenjaju se i strukture samih neuralnih mreža i težine s ciljem pronalaska optimalnog rješenja. Sastoji se od 3 osnovne tehnike: pamćenje gena kako bi se omogućilo ukrštanje topologija, izvršavanje specijalizacije kako bi se očuvale genetske inovacije i razvoj topologije na način da potiče što jednostavnija rješenja. Jednistvenost neuroevolucije promjenjivih topologija je u strukturi koja postaje kompleksnija što je optimalnija, povezujući genetske algoritme sa prirodnom evolucijom.

2.1. Zapis gena

Svaki genom sadrži listu povezujućih gena, od kojih se svaki odnosi na dva povezana čvorna gena (figura 1 [16]). Svaki povezujući gen specifira ulazni čvor, izlazni čvor, težinu veze, da li je ili nije izražen gen (uključujući bit), inovacijski broj, koji dozvoljava pronalazak odgovarajućih gena prilikom ukrštanja. Iako će se rad baviti evolucijom mreža sa jednim izlazom, neuroevolucija promjenjivih topologija se može koristiti za evoluciju mreža sa bilo kojim brojem ulaza ili izlaza.

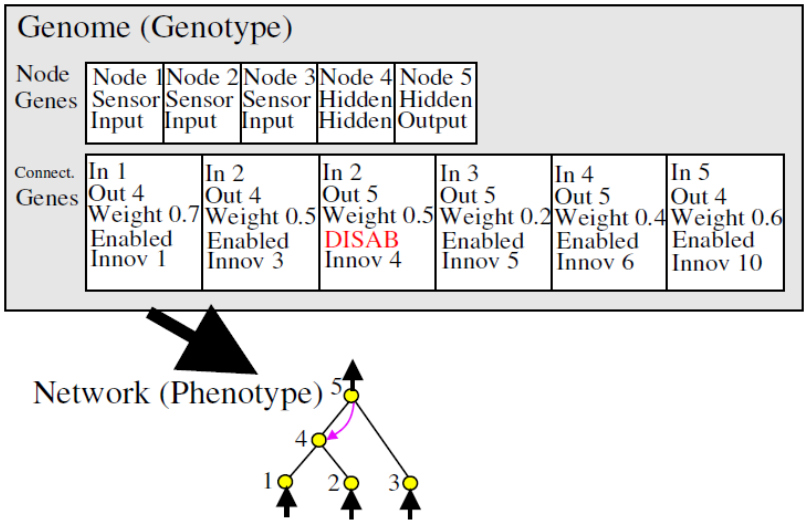

Figura 1: Mapiranje genoma u mrežu

Mutacija u neuroevoluciji promjenjivih topologija mijenja i težine veza i mrežnu strukturu. Težine veza mutiraju kao i u svakom drugom neuroevolucijskom sistemu, tako što se svaka veza mijenja ili ne. Strukturalne mutacije, koje proširuju genom, pojavljuju se na dva načina (figura 2 [16]).

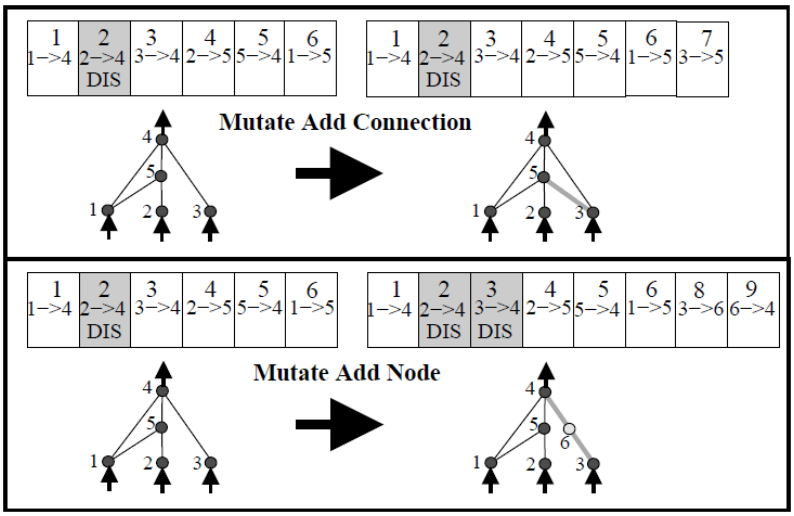

Figura 2: Dva tipa strukturalnih mutacija

U mutaciji dodavanja veza, jedinstvena novi povezujući gen se dodaje na dva ranije nepovezana čvora. U mutaciji dodavanja čvorova postojeća veza se raskida i novi čvor se postavlja na njeno mjesto. Stara veza se onemogućava a dvije nove se dodaju u genom. Ova metoda dodavanja čvorova je izabrana kako bi se novi čvorovi neposredno integrirali na mrežu.
Mutacijom se stvaraju genomi različitih veličina, ponekad sa potpuno različitim vezama na istim lokacijama. Naredno poglavlje objašnjava kako NEAT prelazi preko tako različitih genoma.

2.2. Praćenje gena

Kako bi se izvršilo ukrštanje, sustav mora odrediti koji geni odgovaraju bilo kojim članovima populacije. Ključno je za uočiti da dva gena koja imaju isto podrijetlo predstavljaju istu strukturu (iako mogu imati različite težine) budući da su naslijedili svojstva istog gena u prošlosti. Zbog toga sustav mora znati koji geni se podudaraju s kojima kako bi se pratilo podrijetlo svakog gena u sustavu.
Samo praćenje podrijetla gena ne zahtijeva mnogo računanja. Pojavom novog gena (strukturalnom mutacijom) global inovacijski broj se povećava i dodjeljuje tom genu. Inovacijski brojevi predstavljaju hronologiju svakog gena u sistemu.
Praćenje gena daje NEAT-u sposobnost da učinkovito rješava problem konkurentske konvencije za disparatne topologije [15]. Sustav zna koji geni odgovaraju kojim (figura 3 [16]). Geni koji ne odgovaraju ni jednom drugom su ili dislocirani ili suvišni, u zavisnosti od toga da li se pojavljuju u rasponu roditeljskih inovacijskih brojeva. Neodgovarajući geni nasljeđuju od roditelja s većom dobrotom, a u slučaju jednakosti, od svih roditelja nasumično. Na ovaj način, praćenje gena omogućuje ukrštanje bez potrebe za resursima zahtjevnom topološkom analizom.

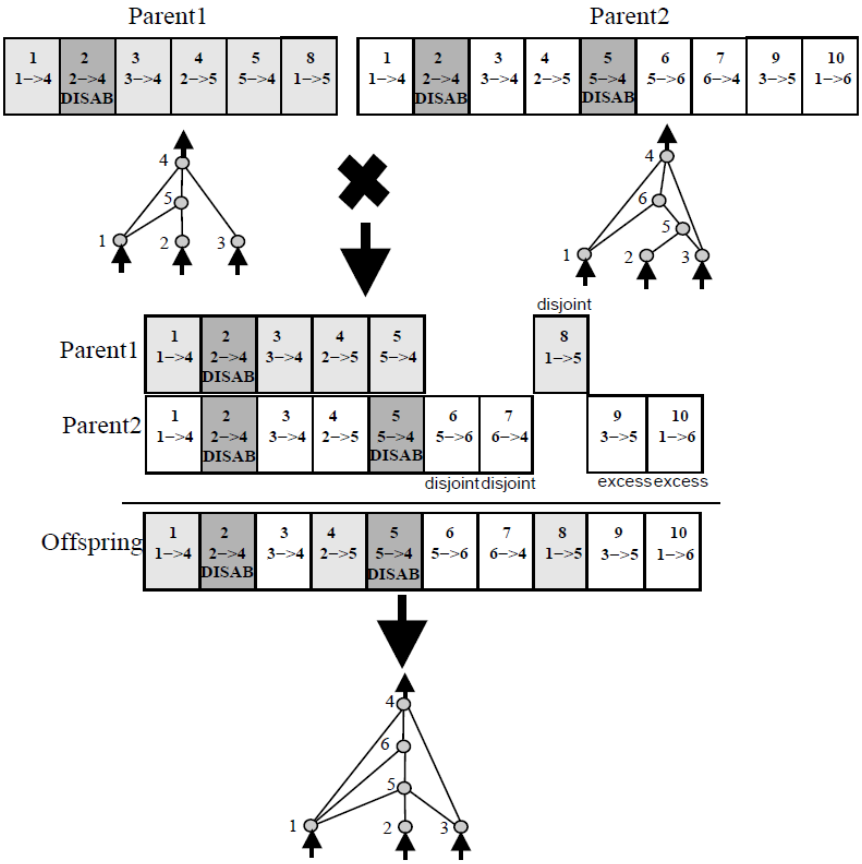

Figura 3: Usklađivanje genoma različitih topologija mreža koristeći inovacijske brojeve

2.3. Očuvanje inovacije kroz specijalizaciju

Dodavanje nove strukture u mrežu inače smanjuje dobrotu same mreže. Međutim, NEAT proširuje populaciju na način da se pojedinci natječu unutar svoje specijalizacije umjesto u cijeloj populaciji. Na ovaj način, topološke inovacije su zaštićene i imaju vremena za optimizaciju strukture prije natjecanja s drugim dijelovima populacije.

Povijesna obilježja omogućuju sustavu da podijeli populaciju u vrste na osnovu topološke sličnosti. Broj dislociranih i suvišnih gena između para genoma je mjera njihove kompatibilnosti. Što su dva genoma dislociranija, dijele manje evolucionarne povijesti i samim tim su manje kompatibilni. Udaljenost kojom mjerimo tu kompatibilnost možemo izračunati po funkciji za čije parametre koristimo broj viška gena E, broj neusklađenih gena D i prosječnu razliku u težinama usklađenih gena W: $\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 W$ [16]. N predstavlja broj gena u većem genomu s ciljem normalizacije izraza. Izračunavanje udaljenosti δ određuje pripadnost vrsti pomoću praga kompatibilnosti δ_t. Jedinka pripada određenoj vrsti kada je udaljenost od predstavnika te vrste manja od praga kompatibilnosti. S ciljem sprječavanja preklapanja vrsta, dati genom se dodijeljuje vrsti prvog reprezentativnog genoma s kojim je kompatibilan. Ukoliko nije kompatibilan ni s jednom postojećom vrstom, stvara se nova vrsta u kojoj taj genom postaje reprezentativan uzorak.

2.4. Minimaliziranje dimenzionalnosti

Umjetne neuralne mreže sa topološkom i težinskom evolucijom inače započinju inicijalnom populacijom nasumičnih topologija [2]. Takva topološka raznolikost se mora uvesti od početka budući da nova struktura često ne preživljava ove metode, što ne čuva inovaciju. Međutim, nije jasno da li je takva raznolikost nužna. Populacija nasumičnih topologija ima značajan udio struktura koje nisu preživjele niti jednu evaluaciju dobrote. Zbog toga ne možemo znati da li su takve strukture nužne. Neučinkovitost tog pristupa je u tome što se radi optimizacije mreže mora pretražiti veći broj dimenzija. Tom pretragom algoritam može trošiti mnogo resursa optimizirajući nepotrebno kompleksne strukture.

Za razliku od toga, NEAT počinje s populacijom mreže bez skrivenih čvorova. Budući da NEAT čuva inovaciju kroz specijalizaciju, može započeti na ovaj način i evoluirati novu strukturu po potrebi. Nove strukture se uvode s promjenom postojećih, i samo one strukture koje se označe dovoljno dobrima kroz evaluaciju dobrote preživaljavaju. Na ovaj način, NEAT pretražuje kroz minimalan broj dimenzija težine, čime značajno smanjuje broj generacija nužan za pronalazak rješenja.

3. Programsko ostvarenje

Izvorni kod projekta je podijeljen u nekoliko datoteka i klasa s ciljem lakše nadogradnje koda i bolje preglednosti. Svi hiperparametri su odvojeni u zasebnu datoteku kako bi se olakšalo ugađanje parametara. Implementacija je napisana u programskom jeziku Python. Python je zbog svoje jednostavnosti i fleksibilnosti trenutno najpopularniji jezik za implementaciju algoritama strojnog učenja. Iako je najveća mana za Python njegova brzina u odnosu na druge programske jezike, veliki broj programskih biblioteka za Python je implementirano u programskom jeziku C što smanjuje razliku u brzini izvođenja. Prednost je što postoji velik broj biblioteka za implementaciju raznih tipova strojnog učenja.

3.1. Korištene biblioteke

U projektu su korištene dvije programske biblioteke: PyGame [17] i Neat [18]. PyGame je korišten za implementaciju igre, dok je Neat korišten za implementaciju i ispitivanje algoritma strojnog učenja. Potrebno je stvoriti inačicu igre i proslijediti ju klasi zaduženoj za implementaciju samog algoritma. Algoritam će koristiti određene parametre proslijeđene u toku izvođenja programa za izlaz. Evaluirat će dobrotu agenta koja je potrebna za evoluciju mreže.
Izuzev main.py i imgs.py datoteka, ostale datoteke s izvornim kodom sadrže definiciju po jedne klase i njenu implementaciju.

3.2. Datoteka imgs.py

Imgs.py je datoteka zadužena za učitavanje slika potrebnih za grafički prikaz igre. Nakon učitavanja slika vrši skaliranje na potrebnu rezoluciju i sprema ih u varijablu kojoj druge datoteke mogu pristupiti za uporabu.

```
1  import pygame
2  import os
3  p1 = pygame.transform.scale2x(pygame.image.load(os.path.join("imgs",
   "pilot1.png")))
4  p2 = pygame.transform.scale2x(pygame.image.load(os.path.join("imgs",
   "pilot2.png")))
5  PILOT_IMGS = {p1 : p2, p2 : p1}
6  PIPE_IMG =
   pygame.transform.scale2x(pygame.image.load(os.path.join("imgs",
   "pipe.png")))
7  OBSTACLE_IMG =
   pygame.transform.scale2x(pygame.image.load(os.path.join("imgs",
   "obstacle.png")))
8  BASE_IMG =
   pygame.transform.scale2x(pygame.image.load(os.path.join("imgs",
   "base.png")))
9  BG_IMG =
   pygame.transform.scale2x(pygame.image.load(os.path.join("imgs",
   "bg.png")))
```

3.3. Klasa base

Base predstavlja bazu na kojoj pilot stoji na početku izvođenja igre i koja nakon nekoliko sekundi postaje nevidljiva. Isključivo je kozmetičke prirode i nije potrebna za samu mehaniku igre.

```
1  from imgs import BASE_IMG
2
3  class Base:
4      SPEED = 5
5      WIDTH = BASE_IMG.get_width()
6      IMG = BASE_IMG
7
8      def __init__(self):
9          self.y = self.WIDTH
10
11      def move(self):
12          self.y += 5
13
14      def draw(self, win):
15          win.blit(self.IMG, (0,self.y))
```

3.4. Klasa obstacle

Zadužena je za implementaciju pokretnih prepreka povezanih na cijevi. Sadrži funkcije potrebne za inicijalizaciju samog objekta, grafički prikaz i mehaniku kretanja prepreke.

```
1  from imgs import OBSTACLE_IMG
2  import pygame
3
4  class Obstacle:
5      SPEED = 5
6      IMG = OBSTACLE_IMG
7
8      def __init__(self, x, y):
9          self.x = x
10          self.y = y
11          self.y
12          self.tilt = 0
13          self.side = True
14          self.rotated_image = pygame.transform.rotate(self.IMG,
   self.tilt )
15          self.new_rect = self.rotated_image.get_rect(center =
   self.IMG.get_rect(topleft = (self.x + 8 + 9 * self.side, y -
   self.rotated_image.get_height() / 2 + 11)).center)
16
17
18      def draw(self, win, y, side):
19          rotated_image = pygame.transform.rotate(self.IMG, self.tilt
   )
20          new_rect = rotated_image.get_rect(center =
   self.IMG.get_rect(topleft = (self.x + 8 + 9 * side, y -
   rotated_image.get_height() / 2 + 11)).center)
21          win.blit(rotated_image, new_rect.topleft)
22          self.rotated_image = rotated_image
23          self.new_rect = new_rect
24
25      def move(self):
26          if self.side:
27              if self.tilt < 45:
28                  self.tilt += 5
29              else:
30                  self.side = not self.side
31                  self.tilt -= 5
32                  self.IMG = pygame.transform.flip(self.IMG, True,
   False)
33          else:
34              if self.tilt > -45:
35                  self.tilt -= 5
```

```
36              else:
37                  self.side = not self.side
38                  self.tilt += 5
39                  self.IMG = pygame.transform.flip(self.IMG, True,
   False)
40
41      def get_mask(self):
42          return pygame.mask.from_surface(self.img)
```

3.5. Klasa pipes

Većinu funkcionalnosti dijeli s klasom obstacle a pored toga sadrži funkcije za nasumično pozicioniranje cijevi i izračun sudara s pilotom što bi označilo kraj izvođenja trenutne instance programa.

```
1  from obstacle import Obstacle
2  from imgs import PIPE_IMG
3  import obstacle
4  import random
5  import pygame
6
7  class Pipe:
8      GAP = 200
9      SPEED = 5
10      def __init__(self, y):
11          self.y = y - PIPE_IMG.get_height()
12          self.dist = 0
13          self.left = 0
14          self.right = 0
15          self.PIPE_RIGHT = pygame.transform.flip(PIPE_IMG, True,
   True)
16          self.PIPE_LEFT = PIPE_IMG
17          self.passed = False
18          self.set_dist()
19          self.left_obstacle = Obstacle(self.dist - 55, self.y)
20          self.right_obstacle = Obstacle(self.right - 30, self.y)
21
22
23      def set_dist(self):
24          self.dist = random.randrange(100, 400)
25          self.left = self.dist - self.PIPE_LEFT.get_width()
26          self.right = self.dist + self.GAP
27
28      def move(self):
29          self.y += self.SPEED
30
```

```
31      def draw(self, win):
32          win.blit(self.PIPE_LEFT,(self.left, self.y))
33          win.blit(self.PIPE_RIGHT, (self.right, self.y))
34
35      def collide(self, pilot):
36          pilot_mask = pilot.get_mask()
37          left_mask = pygame.mask.from_surface(self.PIPE_LEFT)
38          right_mask = pygame.mask.from_surface(self.PIPE_RIGHT)
39
40          left_offset = (self.left - pilot.x, self.y -
   round(pilot.y))
41          right_offset = (self.right - pilot.x, self.y -
   round(pilot.y))
42
43          left_point = pilot_mask.overlap(left_mask, left_offset)
44          right_point = pilot_mask.overlap(right_mask, right_offset)
45
46          left_mask_obstacle =
   pygame.mask.from_surface(self.left_obstacle.IMG)
47          right_mask_obstacle =
   pygame.mask.from_surface(self.right_obstacle.IMG)
48
49          left_offset_obstacle =
   (self.left_obstacle.new_rect.topleft[0] - pilot.x,
   self.left_obstacle.new_rect.topleft[1] - round(pilot.y))
50          right_offset_obstacle =
   (self.right_obstacle.new_rect.topleft[0] - pilot.x,
   self.right_obstacle.new_rect.topleft[1] - round(pilot.y))
51
52          left_point_obstacle =
   pilot_mask.overlap(left_mask_obstacle, left_offset_obstacle)
53          right_point_obstacle =
   pilot_mask.overlap(right_mask_obstacle, right_offset_obstacle)
54
55
56          if left_point or right_point:
57              return True
58          if left_point_obstacle or right_point_obstacle:
59              return True
60          return False
```

3.6. Klasa pilots

Kao i ranije spomenute klase sadrži funkcije nužne za inicijalizaciju, grafički prikaz i kretanje. Pored toga sadrži i funkciju Pilots.turn kojom pilot mijenja smjer kretanja što predstavlja jedinu kontrolu koju igra ima. Za pozivanje ove funkcije bit će zadužena isključivo neuralna mreža.

```
1  import imgs
2  import pygame
3
4  class Pilot:
5      IMGS=imgs.PILOT_IMGS
6      MAX_ROTATION=25
7      ROT_VEL=20
8      ANIMATION_TIME=5
9
10      def __init__(self,x,y):
11          self.x=x
12          self.y=y
13          self.tilt=20
14          self.side = True
15          self.speed=0
16          self.height=self.y
17          self.img_count=0
18          self.img=self.IMGS[[pic for pic in self.IMGS][0]]
19
20      def turn(self):
21          self.side = not self.side
22          self.tilt = self.tilt
23          #pygame.transform.rotate(pygame.transform.flip(self.img,
   not self.side, 0))
24
25      def move(self):
26          self.x += 5 * int((self.side - 0.5) * 2)
27
28      def draw(self, win):
29          rotated_image =
   pygame.transform.rotate(pygame.transform.flip(self.img, not
   self.side, 0), self.tilt * int((self.side - 0.5) * 2))
30          new_rect = rotated_image.get_rect(center =
   self.img.get_rect(topleft = (self.x, self.y)).center)
31          win.blit(rotated_image, new_rect.topleft)
32          self.img = self.IMGS[self.img]
33
34      def get_mask(self):
35          return pygame.mask.from_surface(self.img)
```

3.7. Datoteka main.py

Main.py predstavlja jezgro projekta koje stvara instance potrebnih klasa i upravlja izvođenjem programa. Podijeljena je u nekoliko funkcija radi bolje organiziranosti koda.

```
1  from pygame.surfarray import pixels_green
2  from imgs import BG_IMG
3  import pygame
4  import time
5  import os
6  import random
7  import imgs
8  import pilots
9  import base
10  import pipes
11  import obstacle
12  import sys
13  import neat
14
15  WIN_WIDTH = 500
16  WIN_HEIGHT = 800
17  pygame.font.init()
18  FONT = pygame.font.SysFont('comicsans', 50)
19
20  gen = 0
21  scores = []
22  with open ('scores.txt', mode='w') as f:
23      pass
24
25  def get_mask(self):
26      return pygame.mask.from_surface(self.img)
27
28  def draw_window(win, pilot_list, pipes_list, base_list, score, gen):
29      win.blit(imgs.BG_IMG,(0, 0))
30
31      for pipe in pipes_list:
32          pipe.draw(win)
33          pipe.left_obstacle.draw(win, pipe.y, False)
34          pipe.right_obstacle.draw(win, pipe.y, True)
35          pipe.left_obstacle.move()
36          pipe.right_obstacle.move()
37
38      text = FONT.render("Rezultat: " + str(score), 1, (255, 255,
   255))
39      win.blit(text, (WIN_WIDTH - 10 - text.get_width(), 10))
```

```
40      text = FONT.render("Generacija: " + str(gen), 1, (255, 255,
   255))
41      win.blit(text, (10, 10))
42
43      base_list.draw(win)
44      for pilot in pilot_list:
45          pilot.draw(win)
46
47      pygame.display.update()
48
49  def main(genomes, config):
50      global gen
51      gen += 1
52      nets = []
53      ge = []
54      pilot_list = []
55
56      for _, g in genomes:
57          net = neat.nn.FeedForwardNetwork.create(g, config)
58          nets.append(net)
59          pilot_list.append(pilots.Pilot(230,500))
60          g.fitness = 0
61          ge.append(g)
62
63      base_list = base.Base()
64      pipes_list = [pipes.Pipe(0), pipes.Pipe(250)]
65      win = pygame.display.set_mode((WIN_WIDTH, WIN_HEIGHT))
66      clock = pygame.time.Clock()
67      run = True
68      score = 0
69      pipe_ind = 0
70      t = 0
71      while run:
72          clock.tick(1000)
73          for event in pygame.event.get():
74              if event.type == pygame.QUIT:
75                  run = False
76                  pygame.quit()
77                  quit()
78
79          if not pilot_list:
80              run = False
81              break
82
83          for index, pilot in enumerate(pilot_list):
```

```
84              pilot.move()
85              ge[index].fitness += 1
86
87              t = (t + 1) % 36
88              output = nets[index].activate(
89                  (pilot.y - pipes_list[pipe_ind].y,
90                  (pilot.x - pipes_list[pipe_ind].left),
91                  (pilot.x -
   pipes_list[pipe_ind].left_obstacle.new_rect.bottomright[0]),
92                  (pilot.y -
   pipes_list[pipe_ind].left_obstacle.new_rect.bottomright[1]),
93                  (pilot.side - 0.5) * 2
94                  ))
95
96              if output[0] < 0:
97                  if pilot.side:
98                      pilot.turn()
99              else:
100                  if not pilot.side:
101                      pilot.turn()
102
103         base_list.move()
104         rem = []
105         add_pipe = False
106         for pipe in pipes_list:
107             for i, pilot in enumerate(pilot_list):
108                 if pipe.collide(pilot):
109                     ge[i].fitness -= abs(pilot_list[i].x -
   (pipe.right - pipe.GAP // 2))
110                     pilot_list.pop(i)
111                     nets.pop(i)
112                     ge.pop(i)
113
114                 if not pipe.passed and pipe.y > pilot.y:
115                     pipe.passed = True
116                     add_pipe = True
117                     pipe_ind += 1
118
119             if pipe.y >= 800:
120                 rem.append(pipe)
121
122             pipe.move()
123
124         if add_pipe:
125             score += 1
```

```
126                 pipes_list.append(pipes.Pipe(0))
127
128                 for g in ge:
129                     g.fitness += 5
130
131             for p in rem:
132                 pipes_list.remove(p)
133                 pipe_ind -= 1
134
135             for i, pilot in enumerate(pilot_list):
136                 if pilot.y + pilot.img.get_height() >= 730 or pilot.y <
    0:
137                     pilot_list.pop(i)
138                     nets.pop(i)
139                     ge.pop(i)
140
141
142             draw_window(win, pilot_list, pipes_list, base_list, score,
    gen)
143
144         with open ('scores.txt', mode='a') as f:
145             f.write("{:4} {}\n".format(gen,score))
146
147     def run(config_path):
148
    config=neat.config.Config(neat.DefaultGenome,neat.DefaultReproductio
    n,neat.DefaultSpeciesSet,neat.DefaultStagnation,config_path)
149
150         p=neat.Population(config)
151         p.add_reporter(neat.StdOutReporter(True))
152         stats=neat.StatisticsReporter()
153         p.add_reporter(stats)
154
155         winner=p.run(main)
156
157     if __name__ == '__main__':
158         input()
159         local_dir = os.path.dirname(__file__)
160         config_path = os.path.join(local_dir, 'config-feedforward.txt')
161         run(config_path)
162         while True:
163             print(1)
164         print(scores)
```

3.7.1 Funkcija main.draw_window

Ova funkcija kao parametre prima pilote, prepreke, rezultat i generaciju i zadužena je za grafički prikaz. Implementirana je s ciljem preglednog koda tako da pored ispisa trenutnog rezultata i generacije samo poziva funkcije za grafički prikaz objekta iz ranije objašnjenih klasa.

3.7.2 Funkcija main.main

Predstavlja jezgo izvođenja programa, tj. jednu instancu igre. Prije while petlje za izvođenje igre stvara nove neuralne mreže, po jednu za svaku instancu igre koja će se odvijati paralelno. Za svaku sličicu u toku izvođenja igre šalje ulaz neuralnoj mreži i postupa po naredbi izlaza koja može biti promijeniti smijer kretanja ili ne. Pored toga prati napredak pilota, povećava im dobrotu ili briše u slučaju sudara s preprekom.

4. Evolucija neuralne mreže

Agenta prestavlja umjetna neuralna mreža koja ima 5 ulaznih neurona, jedan izlazni neuron i unutrašnju strukturu koju će odrediti evolucija mreže. Izlaz mreže će odrediti kojim smjerom se igrač treba kretati. Ulazi za neuralnu mrežu su:

- Trenutni smjer kretanja
- Udaljenost do najbliže prepreke po y osi
- Udaljenost od lijevog kraja prepeke po x osi – za evoluciju mreže je dovoljna jedna udaljenost do prepreke po x osi, do lijevog ili desnog kraja budući da je razmak konstantan
- Udaljenost od donjeg desnog ugla pokretne prepreke po x osi
- Udaljenost od donjeg desnog ugla pokretne prepreke po y osi

Na figuri možemo vidjeti četiri ulaza dok je peti ulaz, trenutni smjer kretanja, spremljen u varijabli objekta igrača.

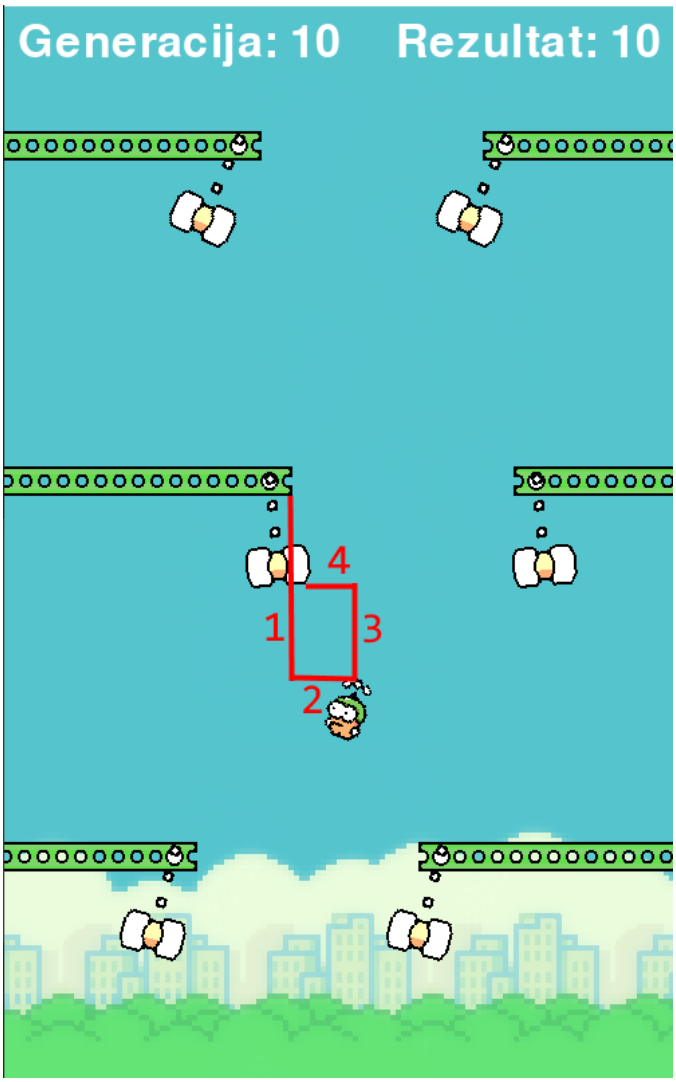

Figura 4: Prikaz ulaza mreže na zaslonu igre

Kada agent dobije informacije od okoliša (ulazi mreže), računa vjerojatnost kretanja u određenom smjeru. Ova vjerojatnost određuje kojim se je smjerom potrebno kretati. Na figuri je prikazan dijagram neuralne mreže koja ima 5 ulaznih neurona, 1 izlazni i 10 u skrivenom sloju. Broj ulaznih i izlaznih neurona je fiksan, dok je broj neurona u skrivenom sloju i sami broj skrivenih slojeva varijabilan u zavisnosti od evolucije mreže.

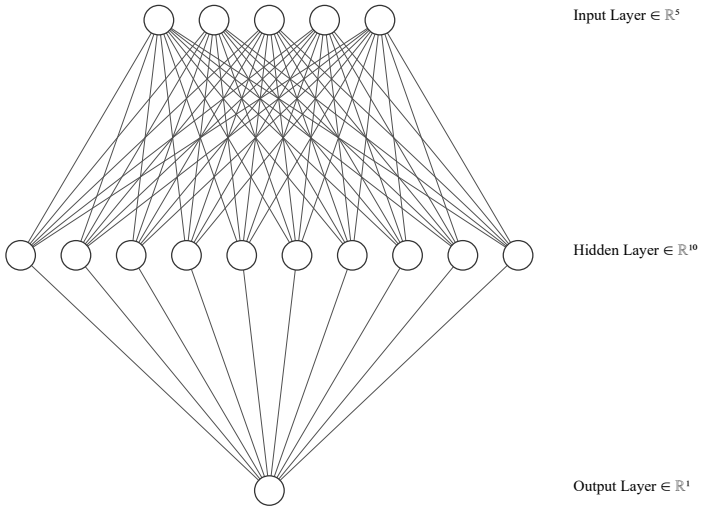

Figura 5: Dijagram neuralne mreže sa brojem ulaza i izlaza potrebnim za evoluciju igre

Ukoliko izlaz mreže traži da se igrač kreće udesno ili ulijevo, a već se ne kreće tih smjerom, izvršit će se promjena smjera kretanja.

4.1. Konfiguracija NEAT parametara

Za konfiguraciju parametara korištene su sljedeće vrijednosti, kojima se mogu dobiti slični rezultati:

- Populacija (population): Korištena je veličina populacije od 25 jedinki. Dovoljno je velika da obezbijedi raznolikost jedinki, a nije prevelika što omogućava brže izvršenje generacija.
- Elitizam (elitism): Izabran je elitizam od 5 jedinki što znači da 5 najboljih jedinki, tj. 20% svake generacije direktno napreduje u iduću. Ova vrijednost je dovoljna za očuvanje inovacije a nije prevelika što bi prouzrokovalo stagnaciju.
- Granica preživljavanja (survival_threshold): Predstavlja udio populacije koji će se ukrštati za stvaranje novih jedinki iduće populacije. Postavljen je na 0.1, tj. 10%.

Za ostale parametre korištene su zadane vrijednosti biblioteke neat-python.

5. Rezultati

Budući da je igra jednostavna za upravljanje, ima samo jednu kontrolu koja predstavlja izlaz neuralne mreže, algoritam je relativno brzo evoluirao mrežu da igra optimalno. Figura 6 predstavlja maksimalne rezultate po svakoj generaciji u jednom izvršenju programa. U figuri možemo vidjeti da je prvih 12 generacija rezultat najbolje jedinke bio svega 0 ili 1. Nasumična generacija težina je već za 13 generaciju generirala težine kojima je neuralna mreža postigla rezultat 8. Daljnjim ukrštanjem jedinki iz generacije 13 dobila se je još bolja neuralna mreža koja je u generaciji 14 postigla rezultat 78. Tad je program prestao s radom budući da je pronašao optimalnu mrežu (u konfiguraciji postavljeno na mrežu koja ostvari rezultat od minimalno 50). Iako je postigla rezultat od 78 a nije nastavila beskonačno igrati, mreža se smatra optimalnom za igranje jer je razlog gubitka na 79. poenu bio rubni slučaj koji neuralna mreža nije mogla predvidjeti i za koji zbog samo prirode nasumične generacije nije nužno moguć za proći (figura 7).

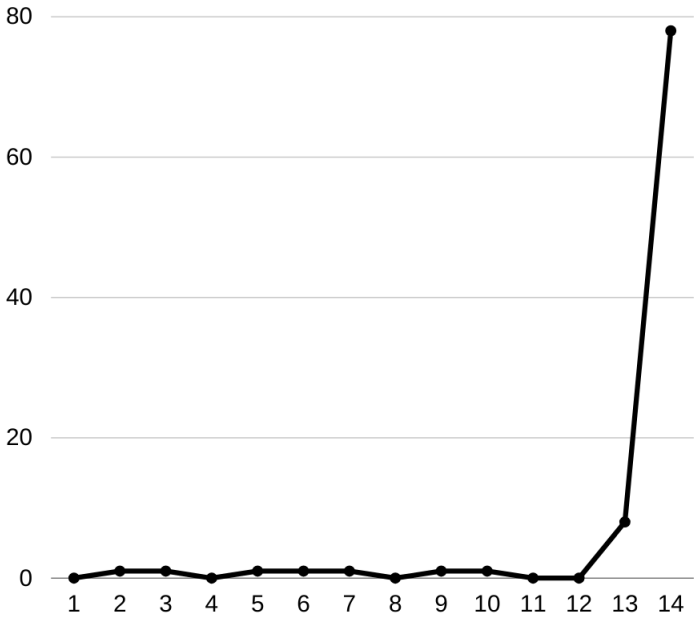

Figura 6: Maksimalni rezultati svake generacije u testom izvršavanju programa

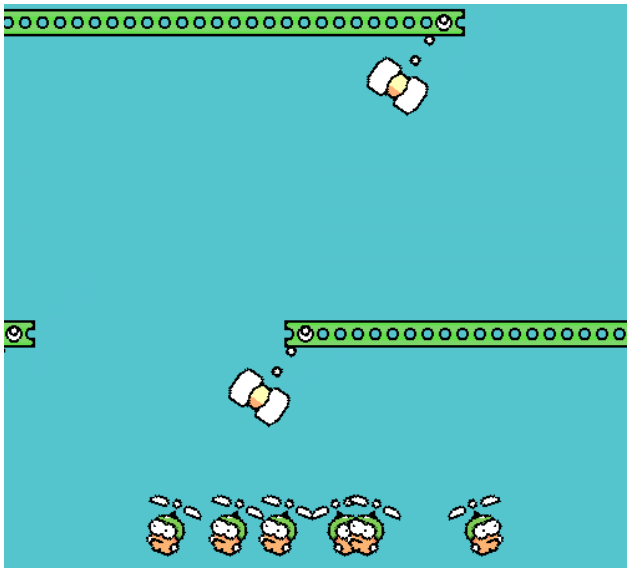

Figura 7: Rubni slučaj koji je nemoguće proći

6. Zaključak

NEAT postepeno evoluira strukturu neuralne mreže počevši od minimalnog početka. Manje strukture, kao struktura mreže iz projekta, se optimiziraju brže, i tu NEAT pronalazi rješenja brže od drugih neuroevolucijskih pristupa. Kada NEAT pronađe strukturu koja radi bilo kako, on ju eksponencijalno poboljšava što možemo vidjeti u rezultatima gdje je u samo dvije generacije rezultat igre skočio sa 0 na 78.
Što je mreža, tj. igra koja oblikuje mrežu, kompleksnija, NEAT-u će trebati više vremena za optimizaciju ukoliko do optimizacije ikako dođe. U mrežama sa izrazito kompleksnim strukturama NEAT će često zapeti u beskonačnoj petlji iz koje ga nasumične generacije ne mogu izvući i zbog toga nije pogodan za evoluciju tih mreža gdje ga drugi neuroevolucijski pristupi nadigravaju.

Literatura

[1] Gomez, F. and Miikkulainen, R. (1999). Solving non-Markovian control tasks with neuroevolution. In Dean, T., editor, Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, pages 1356–1361, Morgan Kaufmann, San Francisco, California.

[2] Gruau, F., Whitley, D., and Pyeatt, L. (1996). A comparison between cellular encoding and direct encoding for genetic neural networks. In Koza, J. R. et al., editors, Genetic Programming 1996: Proceedings of the First Annual Conference, pages 81–89, MIT Press, Cambridge, Massachusetts.

[3] Moriarty, D. E. and Miikkulainen, R. (1997). Forming neural networks through efficient and adaptive co-evolution Evolutionary Computation, 5(4):373–399.

[4] Potter, M. A. and De Jong, K. A. (1995). Evolving neural networks with collaborative species. In Oren, T. I. and Birta, L. G., editors, Proceedings of the 1995 Summer Computer Simulation Conference, pages 340–345, Society for Computer Simulation, San Diego, California..

[5] Whitley, D. et al. (1993). Genetic reinforcement learning for neurocontrol problems. Machine Learning, 13:259–284.

[6] Kaelbling, L. P., Littman, M., and Moore, A.W. (1996). Reinforcement learning: A survey. Journal of Artificial Intelligence, 4:237–285.

[7] Moriarty, D. E. and Miikkulainen, R. (1996). Efficient reinforcement learning through symbiotic evolution. Machine Learning, 22:11–32.

[8] Moriarty, D. E. (1997). Symbiotic Evolution of Neural Networks in Sequential Decision Tasks. Ph.D. thesis, Department of Computer Sciences, The University of Texas at Austin. Technical Report UT-AI97-257.

[9] Gomez, F. and Miikkulainen, R. (2002). Learning robust nonlinear control with neuroevolution. Technical Report AI02-292, Department of Computer Sciences, The University of Texas at Austin, Austin, Texas.

[10] Krzysztof Wloch and Peter J. Bentley. Optimising the performance of a formula one car using a genetic algorithm. In Proceedings of Eighth International Conference on Parallel Problem Solving From Nature, pages 702–711, 2004.

[11] Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. Real-time neuroevolution in the nero video game. IEEE Transactions on Evolutionary Computation, 9(6):653–668, 2005.

[12] Steffen Priesterjahn, Kramer, Alexander Weimer, and Andreas Goebels. Evolution of human-competitive agents in modern computer games. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC), 2007.

[13] Julian Togelius. Optimization, Imitation and Innovation: Computational Intelligence and Games. PhD thesis, Department of Computing and Electronic Systems, University of Essex, Colchester, UK, 2007.

[14] Simon M. Lucas. Ms pac-man competition. SIGEVOlution, 2(4):37–38, 2007.

[15] N. J Radcliffe. Genetic set recombination and its application to neural network topology optimisation. Neural computing and applications, 1(1):67ↀ90, 1993.

[16] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. Evolutionary Computation, 10(2):99–127, 2002.

[17] https://www.pygame.org/news

[18] https://neat-python.readthedocs.io/en/latest/

[19] https://github.com/Skrgich/swingcopters

KONSULTACIJSKI LIST

Datum	Sadržaj konsultacija	Potpis mentora

Datum predaje maturskog rada _____

Mišljenje profesora-mentora o radu:

Ocjena maturskog rada _____( )

Datum odbrane maturskog rada: _____

Komisija u sastavu: Potpis članova komisije:

1. Predsjednik_____ _____
2. Ispitivač _____ _____
3. Stalni član _____ _____

Pitanja na usmenom obrazloženju rada:

1. _____

   _____

2. _____

   _____

3. _____

   _____

Ocjena usmenog obrazloženja rada: _____ ( )

Ocjena rada: _____( )

M.P.