Sadržaj

1. Uvod

U ovome radu pokušati ću objasniti funkcioniranje rada adaptibilnog agenta i prikazati rad takvog agenta na primjeru igrice Pac-Man.

Pac-Man je ime za arkadnu igru koju je razvila japanska tvrtka Namco dok je za tržište u SAD-u licenciju i distribuciju držala tvrtka Midway. Igra je izašla u Japanu 22. svibnja 1980. Prema žanru spada u arkadne igre i puzzle. [1]

Arkadne igre je naziv za sve zabavne igre gdje se igra može započeti nakon što se umetne novac ili žeton u stroj. Arkadne igre obično su smještene u posebnim prostorijama s ostalim arkadnim igrama u tzv. arkadnim igraonicama, ili pojedinačni strojevi nalaze se u zabavnim parkovima, gostionama, kafićima itd. [2]

Igra je implementirana uz pomoć Python programskog jezika. Igru je bilo potrebno instalirati i uključiti kroz terminal Free Python Games za postavljanje već gotovog okruženja, koje je kasnije dorađeno uz pomoć vektora i boja. U tu svrhu koristi se naredba *$ python3 -m pip install freegames*, a nakon toga u .py datoteci potrebno je uključiti instaliranu biblioteku uz pomoć *import freegames*. Agent je implementiran uz pomoć *SPADE* biblioteke o čemu ću nešto više reći i prikazati na primjeru kasnije u radu.

2. Teorija o agentima i Pac-Man igrica

2.1. Agent

Riječ „agent“ je izvedena iz latinskog glagola agere, što znači činiti, raditi ili voditi. U svakodnevnom govoru, agentom se imenuju osobe ili stvari koje čine nešto za nas ili umjesto nas, implicirajući da te osobe ili stvari imaju slobodu odlučivanja o tome koja radnja (ako je ima) je primjerena [3]. Agent je računalni sustav koji je u stanju ponašati se autonomno u nekom okružju kako bi postigao ciljeve svog oblikovanja [4].

Dakle agentom možemo smatrati sustav koji opaža svoju okolinu i onda djeluje na nju. Odnosno sustav koji svoju okolinu i njene podatke kao ulazne vrijednosti uzima i obrađuje, te stvara nove izlazne vrijednosti.

Okolina agenata može biti statična (nepromjenjiva tijekom vremena) ili dinamična (promjenjiva u vremenu). Razvoj sustava zasebnih agenata (engl. single agent) značajno je lakši kada je okolina agenta statična jer su takvi sustavi lakše objašnjivi strogim matematičkim formulama, a time i lakši za upravljanje. Sustavi višestrukih agenata sami po sebi stvaraju dinamičko okruženje iz pogleda svakog zasebnog agenta što uzrokuje njihovu težu integraciju u sustave. [5]

Jedan od glavnih razloga korištenja ovakvog sustava su složeni zadatci. Integracijom ovakvih sustava u nekoj okolini zapravo dobivamo mogućnost rješavanja onih najsloženijih zadataka (računalne igrice, transportni sustavi, mreže, grafičke aplikacije, ...). Ukoliko imamo više agenata u nekoj okolini potrebno je staviti naglasak na njihovu komunikaciju, kako ne bi dolazilo do određenih konflikata i time krivo odrađenih zadataka. Interakciju agenata razlikujemo s obzirom na njihove ciljeve i resurse. Ciljevi označavaju rješenje zadataka koji agent pokušava izvršiti i ukoliko ima više agenata, njihovi ciljevi mogu biti usklađeni (zajedno postižu cilj; surađuju) ili neusklađeni. Resursi su oznaka za okolinu agenta koja mora biti osigurana prema zadanim parametrima, kako bi agent bio u mogućnosti ispravno obaviti svoj posao. Kao primjer resursa možemo spomenuti vrijeme (vremenski prostor), dostupnost potrebnih informacija, procesor, ...

2.2. Adaptibilni agent

Agent predstavlja model ili algoritam čiji je glavni zadatak učenje za izvršenje zadatka. Takvo učenje se naziva podržano ili pojačano učenje (eng. Reinforcement learning) i ono je područje strojnog učenja koje se bavi načinom na koji agenti u smislu algoritama uče kako izvršiti neki zadatak te za to bivaju „nagrađeni”. Zbog sposobnosti učenja, uz agente se veže pojam inteligencije (AI – Artificial intelligence), tj. pojmovi kao što su mentalna karakteristika, sposobnost učenja iz iskustva, prilagodba na nove situacije i snalaženje u novim situacijama.

Dakle agenti povlače pojedine akcije odnosno donose odluke u okolini koju možemo opisati kao „svijet” agenta koji kontrolira ono što agent opaža i ono što dobiva kao povratnu

informaciju s obzirom na izvršenje danog zadatka.

Postoje brojne vrste agenata poput reaktivnog agenta, proaktivnog agenta, mobilnog agenta i sl. U ovome radu, kako je to već navedeno, radi se o adaptibilnom agentu.

Adaptibilni agent je agent koji se adaptira, odnosno prilagođava okolini u kojoj je postavljen i unutar koje mora izvršiti neki zadatak učenjem.

2.3. Pac-Man igra

Pac-Man igra u smislu Python igre višeagentnih sustava implementirana je uz pomoć jednog agenta.

Dijelovi igre su: agent i okolina (okolina + resursi). U ovome smislu okolinu čini labirint u kojem se pacman kreće, duhovi čiji je zadatak uhvatiti pacman-a, hrana koju sakuplja pacman i koja se broji u konačan rezultat koji onda daje završno rješenje igre.

Agent je implementiran na dva načina. Prilikom pokretanja igre, korisnik može odabrati koju vrstu agenta želi pokrenuti. Razlikuje se agent koji je ujedno korisnik i pokreće pacman-a tipkama na tipkovnici, te samostalni agent koji se nasumično kreće u labirintu i paralelno uči o okolini (adaptira se).

Na posljetku možemo vidjeti koji agent je bolje obavio posao, pogledom u datoteku *Rezultat.txt*.

Cilj igre je da pacman pojede što više hrane, a da ga pri tome ne ulovi duh.

Način na koji je igra Pac-Man implementirana slijedi u nastavku rada.

3. Kritički osvrt

Kao osnovnu literatura pri izradi igre Pac-Man koristila sam:

- Saito, Sean, i ostali. Python Reinforcement Learning Projects. Packt Publishing, 2018.

  Gore navedena literatura govori o tome koliko je danas Reinforcement Learning popularno i predstavlja područje podložno velikim promjenama u smislu napretka unutar strojnog učenja. Ova literatura dala mi je uvid u korištenje mogućih biblioteka i primjera rada takvih agenata.

  U knjizi se spominju brojni algoritmi pojačanog učenja i prikazani su načini njihove implementacije. Područja primjene literature su: igre, te obrade slika, teksta i videa. Također se dobiva uvid u tehnologije pojačanog učenja poput TensorFlow i OpenAI Gym. U literaturi je igra Pac-Man nabrojana pod Atari igre, tj. američke programerske tvrtke video igara. Koraci za ovakvu implementaciju su izgradnja Atari emulatora, priprema podataka, primjena Depp Q-learning-a (npr. algoritmi), implementacija nekih dodatnih dijelova i na kraju izvođenje eksperimenata (pokretanje programa).

  Literatura je izdana za ljude profesionalce koji imaju za želju dodatno razvijati svoje znanje, tako da nisam isključivo koristila ovu literaturu obzirom da sam početnik.

- Kane, Frank. Hands-on Data Science and Python Machine Learning: Perform Data Mining and Machine Learning Efficiently Using Python and Spark. Packt, 2017.

  Daje uvid u strojno učenje, biblioteke, korake izrade programa kao i prethodna literatura.

  Za razliku od prethodne u ovoj literaturi baš postoje primjeri izrade algoritma za Pac-Man igru, primjerice na stranici 266 gdje se objašnjava Q-learning. Primjerice imamo Pac-Man i njegovu okolinu (brojni uvjeti). Određena akcija ima vrijednost Q, te Q ovisi o kretnji Pac-Man-a (npr. negativan Q = dodir duha). Vrijednost Q kreće od nula pa do svake druge vrijednosti obzirom na stanje koje Pac-Man može poprimiti. Nakon toga Pac-Man istražuje labirint, ujedno sa svim lošim događajima koji mu se mogu dogoditi (npr. dodir duha). Time Pac-Man uči o okolini na način da se pri takvim događajima mijenja vrijednost Q s obzirom na stanje u kojem se nalazio. Dobri događaji kao npr. hranjenje, povećavaju vrijednost Q.

4. Implementacija

4.1. Opis sustava igre

Pri izradi igre koristila sam se uz pomoć primjera *Turtle* Pac-Man igri na internetu.

Pri izradi Pac-Man igre korištene su tri osnovne biblioteke:

- spade - biblioteka uz pomoć koje pozivam State ponašanje kojim implementiram agenta. Agent se sastoji od tri State ponašanja. U prvom gdje se proizvoljno kreće dok ga duh ne ulovi. Nakon toga prelazi iz stanja jedan u stanje dva gdje bježi ukoliko mu duh prilazi (orijentacija prema duhu), a potom iz stanja dva prelazi u stanje tri. Stanje tri je stanje u kojem se agent kreće s obzirom na duha i na hranu (orijentacija prema duhu i hrani), odnosno skrenut će u onaj dio labirinta gdje ima hrane.
- freegames - biblioteka uz pomoć koje implementiram okolinu agenta, tj. labirint i brojanje sakupljene hrane. Pri tome se koristim vektorima za slaganje labirinta i za odabir početne pozicije agenta i duhova.
- turtle - modul pomoću kojeg implementiram samu igru, odnosno kretanje agenta i duhova, te zaslon okoline. Neke od metoda koje pozivam iz navedene biblioteke su: clear(), reset(), ontimer(), update(), setup(), title(), . . .

Ostale korištene biblioteke:

```
from spade.agent import Agent
from spade import quit_spade
from random import randint
import argparse
from ast import literal_eval
from time import sleep
import functools
import inspect
import traceback
import time
import turtle
from math import *
from tkinter import *
from tkinter import ttk
from random import choice
from turtle import *
from freegames import floor, vector
from datetime import datetime
```

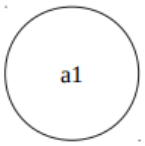
4.2. Matrica, tablice i vizualizacija

4.2.1. Matrica suradnje agenta

Cijeli sustav se temelji na jednom agentu (pacman). Pa s obzirom na to mrežna matrica izgleda:

𝔸	a1
a1	0

Prema matrici graf je slijedeći:

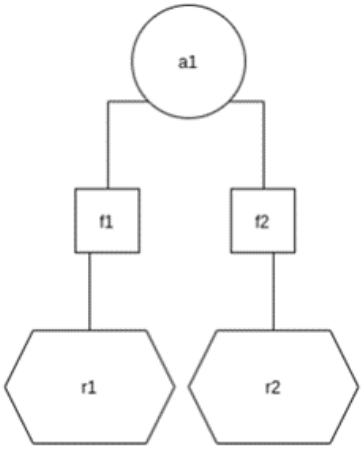


4.2.2. Matrica potrebe agenata za resursima

Obzirom na trenutni jednoagentni rad agent koristi resurse okoline, tj. labirint (r1) i hranu (r2). Ovdje nećemo ubrajati duhove kao resurse jer njega agent ne koristi da bi opstao u igri, već u dodiru sa duhom agent je izgubio u igri. Duhovi jesu jedan od resursa okoline, ali u ovome slučaju ne resurs za potrebe agenta. Matrica potrebe agenata za resursima izgleda:

$\mathbb{P}$	a1
r1	1
r2	1

Prema tome graf izgleda:



Funkcije f1 i f2 definiramo na sljedeći način: Oznakom T označavamo vremenski trenutak u kojem razmatramo sustav, a sa oznakom v označavamo korištenje resursa (njegova pozicija), te v1 poziciju pacman-a. Znači da funkcija glasi:

$$f_{1,2}(T) = ako\ u\ trenutku\ T\ vrijedi\ v = v1$$

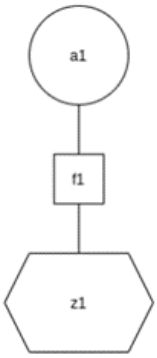
Odnosno, ako se u nekom trenutku pacman nalazi na resursu r1 on ga koristi (u igri je to cijelo vrijeme, jer se nalazi u labirintu). Jednako vrijedi i za resurs r2 koji označava hranu. Ako je pozicija pacman-a na r2, on koristi resurs hrane na način da se povećava finalni rezultat za jedan, a na toj poziciji hrana nestaje jer ju je pacman sakupio.

4.2.3. Matrica odgovornosti agenata za zadatke

U ovom sustavu agent ima samo jedan zadatak, a to je da sakupi što više hrane, po mogućnosti sve. Matrica odgovornosti agenta za zadatke prema tome izgleda:

∅	a1
z1	1

S obzirom na matricu, graf izgleda:

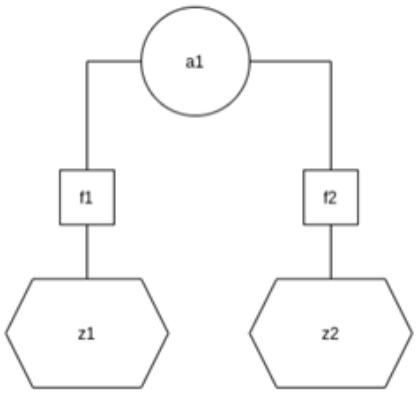


Funkciju f1 definiramo na sljedeći način: Oznakom T označavamo vremenski trenutak u kojem razmatramo sustav, a sa oznakom v označavamo stanje zadatka (početno stanje je 0), te v1 stanje pacman-a (početno stanje je 1). Znači da funkcija glasi:

$$f_1(T) = ako\ u\ trenutku\ T\ vrijedi\ v = v1.$$

Odnosno, ukoliko nema hrane u labirintu, v iz nula mijenja stanje u 1, te je zadatak z1 izvršen. Također možemo podijeliti trenutni sustav s obzirom na stanja (učenje) i zadatke. U tom slučaju glavni zadatak je da pacman sakupi svu hranu u labirintu. Također mora paziti da ga duh ne uhvati i bježati od duha ukoliko mu se približava. U tom slučaju matrica i graf izgledaju:

∅	a1
z1	1



Funkcija f1 ostaje jednaka, dok f2 se računa s obzirom na poziciju duha i pacmana. Zadatak je uspješno izvršen ukoliko v kao pozicija pacmana nikada ne bude jednaka v* označenoj kao poziciji duha.

4.3. Implementacija okoline

Prije svega potrebno je definirati osnovne varijable, to su:

- stanje (rezultat igre)
- put, writer (varijable turtle biblioteke postavljene vidljivosti na False, odnosno ne prikaz)
- aim, pacman, pacman2, duh (entiteti igre inicijalizirani vektorima)
- polje (postavljanje labirinta po kojem se kreću pacman, pacman2 i duhovi)

```
stanje = {'rezultat': 0}
put = Turtle(visible=False)
writer = Turtle(visible=False)
aim = vector(5, 0)
pacman = [[vector(-40, -80), vector(0, 5)]]
pacman2 = vector(-40, -80)
duh = [
    [vector(-180, 160), vector(5, 0)],
    [vector(-180, -160), vector(0, 5)],
    [vector(100, 160), vector(0, -5)],
    [vector(100, -160), vector(-5, 0)],
]
polje = [
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
    0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
    0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
    0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
    0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0,
    0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
    0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
    0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
    0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
    0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
    0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
    0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
    0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0,
    0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0,
    0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0,
    0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
    0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
]
```

U nastavku imamo prikazane metode, tj. funkcije square, offset, valid, world koje služe za stvaranje okoline odnosno labirinta i hrane. Postavljanje puta u kojem će se pacman i duhovi kretat.

4.4. Implementacija okoline

4.4.1. Implementacija

Kao što je to već navedeno, u igri postoje dvije vrste igranja:

```
def square(x, y):
    #crtanje kvadrata puta uz (x, y)
    put.up()
    put.goto(x, y)
    put.down()
    put.begin_fill()
    for count in range(4):
        put.forward(20)
        put.left(90)
    put.end_fill()

def offset(point):
    #vrati offset tocke u polju
    x = (floor(point.x, 20) + 200) / 20
    y = (180 - floor(point.y, 20)) / 20
    index = int(x + y * 20)
    return index
```

```
def valid(point):
    #vrati True ako je validirana tocka u polju
    index = offset(point)
    if polje[index] == 0:
        return False
    index = offset(point + 19)
    if polje[index] == 0:
        return False
    return point.x % 20 == 0 or point.y % 20 == 0

def world():
    #stvaranje labirinta
    Screen().title("VAS - PACMAN")
    bgcolor('black')
    put.color('blue')
    for index in range(len(polje)):
        tile = polje[index]
        if tile > 0:
            x = (index % 20) * 20 - 200
            y = 180 - (index // 20) * 20
            square(x, y)
            if tile == 1:
                put.up()
                put.goto(x + 10, y + 10)
                put.dot(2, 'yellow')
```

- samostalni agent
- korisnik kao agent

Kretnja samostalnog agenta implementirana je na način:

```
def move():
    #kretnja pacmana i duhova - samostalno
    writer.undo()
    writer.write(stanje['rezultat'])

    clear()

    for ve in pacman:
        index = offset(ve[0])

    if polje[index] == 1:
        polje[index] = 2
        stanje['rezultat'] += 1
        x = (index % 20) * 20 - 200
        y = 180 - (index // 20) * 20
        square(x, y)
    elif polje==0:
        print("POBJEDA, ovo je kraj igre!")

    for point, course in pacman:
        if valid(point + course):
            point.move(course)
        else:
            options = [
                vector(5, 0),
                vector(-5, 0),
                vector(0, 5),
                vector(0, -5),
            ]
            plan = choice(options)
            course.x = plan.x
            course.y = plan.y
        up()
        goto(point.x + 10, point.y + 10)
        dot(20, 'yellow')

    for point, course in duh:
        for point2, course2 in pacman:
            opposite=point.y-point2.y
            adjacent=point.x-point2.x
```

```
        try:
            angle = atan(adjacent/opposite)
            if point2.x < point.x:
                backward(100)
        except ZeroDivisionError:
            angle = 0
```

```
    if valid(point + course):
        point.move(course)
    else:
        options = [
            vector(5, 0),
            vector(-5, 0),
            vector(0, 5),
            vector(0, -5),
        ]
        plan = choice(options)
        course.x = plan.x
        course.y = plan.y
    up()
    goto(point.x + 10, point.y + 10)
    dot(20, 'white')

update()
```

```
for point, course in duh:
    if abs(ve[0] - point) < 20:
        print("UDAR, izgubio sam.")
        f = open("Rezultat.txt", "a+")
        f.write("\ns: " + str(stanje['rezultat']))
        bye()
        return

ontimer(move, 100)
```

Dakle duhovi i pacman kreću se nasumično po labirintu. Crvenim okvirom naznačeno je pacman-ovo sakupljanje hrane, te ukoliko hrane više nema pacman je pobjedio. Žutim okvirom naznačena je kretnja pacman-a, a zelenim duha. Sivom bojom je naznačeno izbjegavanje duhova, dakle kada pacman osjeti duha u određenom krugu krene se kretati unatrag, odnosno bježati od duha. Plavom bojom označen je gubitak u igri, odnosno duh je uhvatio pacman-a. U ovome dijelu se bilježi konačni rezultat samostalnog agenta u datoteku *Rezultat.txt*.

Na sličan način je implementiran i agent kao korisnik, odnosno igra u kojoj pacman-a pokreće korisnik na tipke:

```
    def move2():
        #kretnja pacmana i duhova - tipke
        writer.undo()
        writer.write(stanje['rezultat'])

        clear()

        if valid(pacman2 + aim):
            pacman2.move(aim)

        index = offset(pacman2)

        if polje[index] == 1:
            polje[index] = 2
            stanje['rezultat'] += 1
            x = (index % 20) * 20 - 200
            y = 180 - (index // 20) * 20
            square(x, y)
        elif polje==0:
            print("POBJEDA, ovo je kraj igre!")

        up()
        goto(pacman2.x + 10, pacman2.y + 10)
        dot(20, 'red')

        for point, course in duh:
            if valid(point + course):
                point.move(course)
            else:
                options = [
                    vector(5, 0),
                    vector(-5, 0),
                    vector(0, 5),
                    vector(0, -5),
                ]
                plan = choice(options)
                course.x = plan.x
                course.y = plan.y

            up()
            goto(point.x + 10, point.y + 10)
            dot(20, 'white')

update()

for point, course in duh:
    if abs(pacman2 - point) < 20:
        print("UDAR, izgubio sam.")
        f = open("Rezultat.txt", "a+")
        now = datetime.now()
        dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
        f.write("\n" + dt_string)
        f.write("\nt: " + str(stanje['rezultat']))
        bye()
        return

ontimer(move2, 100)
```

Za prikaz koda vrijedi sve kao i u prethodnom prikazu kod samostalnog agenta, uz iznimku da kod pacman-a nemamo implementirani dio sa samostalnom kretnjom i izbjegavanjem duhova, već je za pacman-a implementirani dio prikazan u crvenom okviru. Za ovakvu vrstu igre bilo je potrebno implementirati funkciju promjene s obzirom na pokrenute tipke:

```
def change(x, y):
    #mjenjanje pacman-a s obzirom na tipke
    if valid(pacman2 + vector(x, y)):
        aim.x = x
        aim.y = y
```

Agent je implementiran na način da je inicijaliziran funkcijom kretnje (s obzirom na vrstu

pokretanja):

```
class Igra(Agent):
    def __init__(self, *args, vrsta, **kwargs):
        super().__init__(*args, **kwargs)
        self.vrsta = vrsta
        self.say("Krećem sa igrom!")
        if self.vrsta in ["tipka", "t"]:
            move2()
        else:
            move()

    def say(self, msg):
            print(f"{self.name}: {msg}")
```

A dio za izvršavanje programa izgleda na niže zadani način:

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="Primjer pokretanja: #python3 pacman.py -v t -jid agent@rec.foi.hr -pwd tajna")
    parser.add_argument("-v", "--vrsta", type=str, help="Vrsta agenta (s - samostalno ili t - tipkovnica)")
    parser.add_argument("-jid", type=str, help="JID agenta")
    parser.add_argument("-pwd", type=str, help="Lozinka agenta", default="tajna")
    args = parser.parse_args()

    agent = Igra(args.jid, args.pwd, vrsta=args.vrsta)
    agent.start()

    setup(800, 500, 300, 100)
    hideturtle()
    tracer(False)
    writer.goto(160, 160)
    writer.color('white')
    writer.write(stanje['rezultat'])
    listen()
    onkey(lambda: change(5, 0), 'Right')
    onkey(lambda: change(-5, 0), 'Left')
    onkey(lambda: change(0, 5), 'Up')
    onkey(lambda: change(0, -5), 'Down')
    world()

    input("Press ENTER to exit.\n")
    agent.stop()
    quit_spade()
```

Na početku se uz pomoć ArgumentParser definiraju svi potrebni argumenti za pokretanje skripte. Crveni okvir označava pokretanje agenta, a plavi isključivanje agenta.

Metode pozvane između označavaju otvaranje dijaloga i postavljanje okoline sa duhovima i pacman-om. Dok *onkey* funkcije označavaju kretnju uz pomoć tipki ukoliko je pokrenuta takva vrsta igre.

4.4.2. Pokretanje

-v označava koja vrsta se pokreće. Sa *t* je označena vrsta „tipke“, a sa s označena je vrsta „samostalno“

-jid označava *id@xmpp server* na koji se povezujemo

-pwd označava lozinku za -jid

Za pokretanje igre potrebno je unijeti slijedeće argumente:

- Korisnik kao agent

  python3 pacman.py -v t -jid agent@rec.foi.hr -pwd tajna

- Samostalni agent

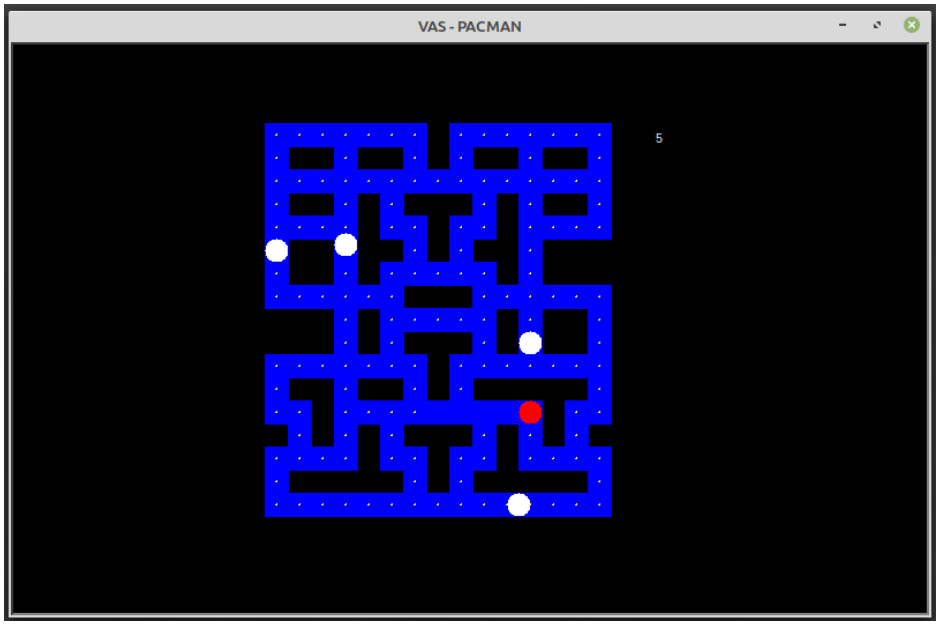
  python3 pacman.py -v s -jid agent@rec.foi.hr -pwd tajna

5. Prikaz rada aplikacije

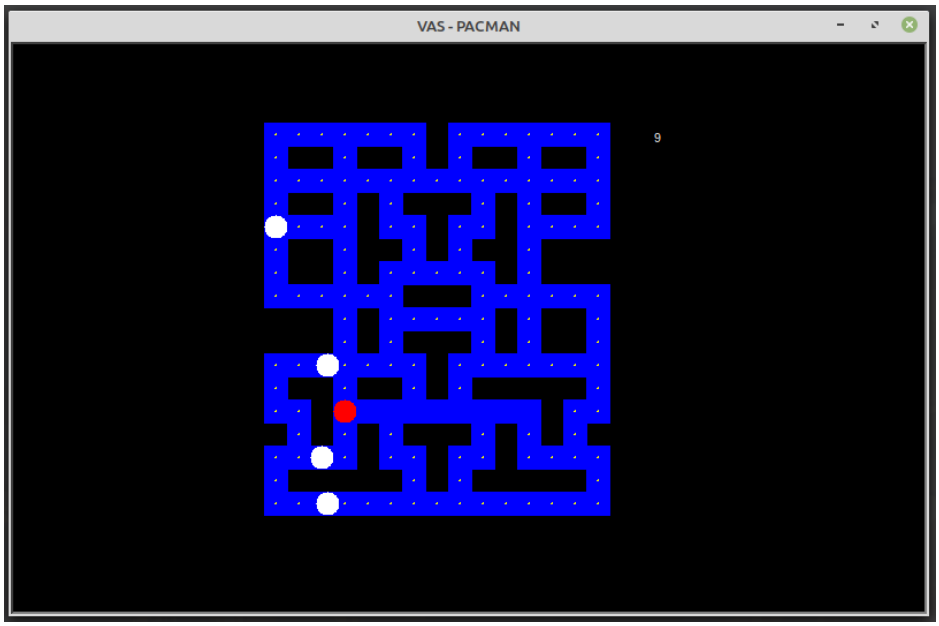
Na početku je potrebno pokrenuti vrstu agenta *t*.

```
oem@Paula:~/Desktop/pacman$ python3 pacman.py -v t -jid ppletes@foi.hr -pwd ppletes123
ppletes: Krećem sa igrom!
Press ENTER to exit.
```

Pokretanjem skripte se u terminalu ispisuje početak igre i mogućnost izlaska iz igre i isključivanja agenta klikom na *Enter*. Nakon što se pokrene agent, otvara se sučelje igre sa crvenim Pac-Man-om.



Kako se Pac-Man pokreće tipkama, tako se rezultat u desnom gorenjm uglu mijenja.



Kada se Pac-Man i duh dodirnu, zatvara se sučelje i ispisuje se kraj igre u terminalu.

```
oem@Paula:~/Desktop/pacman$ python3 pacman.py -v t -jid ppletes@foi.hr -pwd ppletes123
ppletes: Krećem sa igrom!
Press ENTER to exit.
UDAR, izgubio sam.
```

```
oem@Paula:~/Desktop/pacman$ python3 pacman.py -v t -jid ppletes@foi.hr -pwd ppletes123
ppletes: Krećem sa igrom!
Press ENTER to exit.
UDAR, izgubio sam.

oem@Paula:~/Desktop/pacman$
```

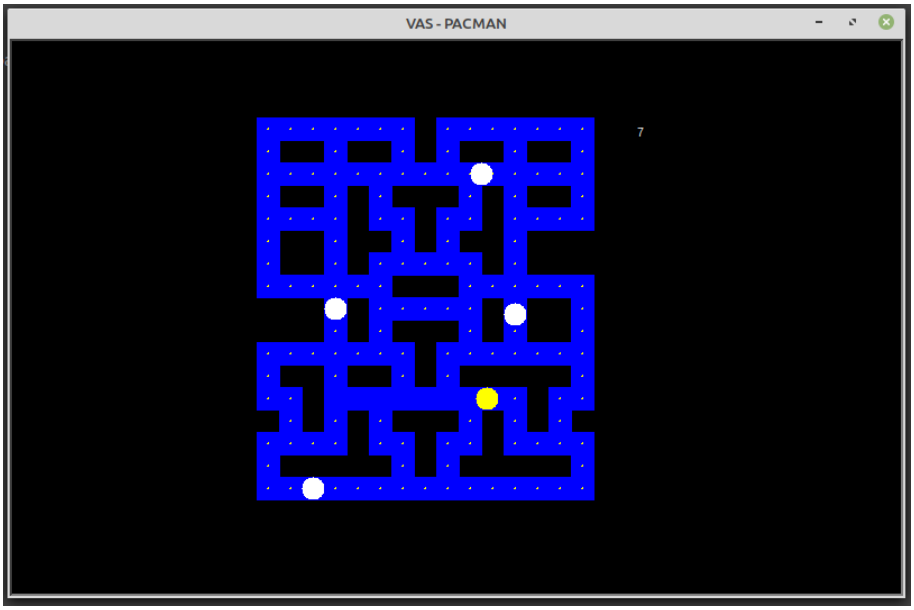
Klikom na *Enter*, isključuje se aktivni agent.

Nakon toga moguće je otvoriti *Rezultat.txt* i u njemu je zapisan konačan rezultat koji je postigao korisnik kao agent.

Rezultat.txt

```
1
2 23/01/2021 15:20:22
3 t: 9
```

Pokretanjem samostalnog agenta, potrebno je kod argumenta -v umjesto *t* napisati *s*. Nakon toga otvara se sučelje sa žutim Pac-Man-om.



Kada se duh i Pac-Man dodirnu, u terminalu se ispisuje jednaka poruka za kraj kao i u prethodnom primjeru.

```
oem@Paula:~/Desktop/pacman$ python3 pacman.py -v s -jid ppletes@foi.hr -pwd ppletes123
ppletes: Krećem sa igrom!
Press ENTER to exit.
UDAR, izgubio sam.
```

Nakon toga, datoteka *Rezultat.txt* izgleda na sljedeći način:

Rezultat.txt

```
1
2 23/01/2021 15:20:22
3 t: 9
4 s: 21
```

6. Zaključak

U dokumentaciji je prikazan jednostavan rad Pac-Man igre korištenjem agenta.

Prema usporedbi dvije vrste igranja igre, pobjedu odnosi korisnik kao agent, odnosno vrsta t. Samostalni agent se kreće nasumično, te svoju orijentaciju kretnje bira nakon dodira zida labirinta. To znači ako duh ide u njegovom smjeru (pri čemu Pac-Man ne zna čitati smjer duha), a on ga nakon udara u zid nema u svom krugu očitanog kao blizu, onda bira random smjer kretnje. Time može biti izazvan kraj igre, ukoliko Pac-Man odabere smjer u kojem se s druge strane kreće duh. Agent je naučio bježati od duha, ali pošto se ne radi o jako kompleksnom primjeru, agent isto tako može biti uhvaćen od duha. S druge strane korisnik kao agent vidi smjer kretnje duha, pa tako može unaprijed reagirati promjenom smjera Pac-Man-a i time ostvariti bolji rezultat.

Cilj ove igre je da Pac-Man pojede što više hrane, a da ga duh ne uhvati.

Međutim, svrha ove igre u znanstvenom smislu je prikazivanje načina implementacije jednostavnog adaptibilnog agenta, načina na koji je konfiguriran, kao i kako se izvršava i djeluje na okolinu igre, te usporedbu rezultata sa korisnikom kao agentom.

7. Popis literature

[1] „Pac-Man“. Wikipedia, 23. siječanj 2021. Wikipedia, https://en.wikipedia.org/w/index.php?title=Pac-Man&oldid=1002266977.

[2] ARKADNE IGRE - Sve što Neznam. https://sites.google.com/site/sveestoneznam/horor-igre/arkadne-igre. Pristupljeno 23. siječanj 2021.

[3] Nwana, H. S. (1996) Software Agents: An Overview. Cambridge University Press, Knowledge Engineering Review, Vol. 11, Br. 3, 205-244.

[4] Schatten, M., 2017. Višeagentni sustavi Inteligentni agenti. FOI, Višeagentni sustavi, Materijali sa predavanja, 2.

[5] Vlassis, Nikos. A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence. 1. ed, Morgan & Claypool, 2007.

[6] Grgić, Marijan, i Nada Vučetić. „Sustavi višestrukih agenata u kartografskoj generalizaciji“. Ekscentar, izd. 15, svibanj 2012., str. 100–03.

[7] „Inteligentni Agenti“. Prezi.Com, https://prezi.com/kdbr3izfbcds/inteligentni-agenti/. Pristupljeno 23. siječanj 2021.

Saito, Sean, i ostali. Python Reinforcement Learning Projects. Packt Publishing, 2018.

Kane, Frank. Hands-on Data Science and Python Machine Learning: Perform Data Mining and Machine Learning Efficiently Using Python and Spark. Packt, 2017.

Pacman — Free Python Games 2.3.2 documentation. http://www.grantjenks.com/docs/freegames/pacm Pristupljeno 15. siječanj 2021.