دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)
دانشکده ریاضی و علوم کامپیوتر

پایان‌نامه کارشناسی‌ارشد
گرایش ریاضی

هوش مصنوعی - گزارش ۰۴ - پیاده سازی بازی دونفره با جستجوی تخاصمی

پایان‌نامه

نگارش
آترین حجت

استاد راهنما
نام کامل استاد راهنما

استاد مشاور
نام کامل استاد مشاور

فروردین ۱۴۰۰


بسم الله الرحمن الرحيم

صفحه فرم ارزیابی و تصویب پایان نامه- فرم تأیید اعضاء کمیته دفاع

در این صفحه فرم دفاع یا تایید و تصویب پایان نامه موسوم به فرم کمیته دفاع- موجود در پرونده آموزشی- را قرار دهید.

نکات مهم:

- نگارش پایان نامه/رساله باید به زبان فارسی و بر اساس آخرین نسخه دستورالعمل و راهنمای تدوین پایان نامه های دانشگاه صنعتی امیرکبیر باشد.(دستورالعمل و راهنمای حاضر)
- رنگ جلد پایان نامه/رساله چاپی کارشناسی، کارشناسی ارشد و دکترا باید به ترتیب مشکی، طوسی و سفید رنگ باشد.
- چاپ و صحافی پایان نامه/رساله بصورت پشت و رو(دورو) بلامانع است و انجام آن توصیه می شود.

دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)

به نام خدا

تعهدنامه اصالت اثر

تاریخ: فروردین ۱۴۰۰

اینجانب **آترین حجت** متعهد می‌شوم که مطالب مندرج در این پایان‌نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی اساتید دانشگاه صنعتی امیرکبیر بوده و به دستاوردهای دیگران که در این پژوهش از آنها استفاده شده است مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذکر گردیده است. این پایان‌نامه قبلاً برای احراز هیچ مدرک هم‌سطح یا بالاتر ارائه نگردیده است.

در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

کلیه نتایج و حقوق حاصل از این پایان‌نامه متعلق به دانشگاه صنعتی امیرکبیر می‌باشد. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس از این پایان نامه بدون موافقت کتبی دانشگاه صنعتی امیرکبیر ممنوع است. نقل مطالب با ذکر مآخذ بلامانع است.

آترین حجت

امضا

فهرست مطالب

عنوان صفحه

فهرست اشکال

شکل صفحه

فهرست جداول

جدول صفحه

فهرست نمادها

نماد	مفهوم
$n(G)$	تعداد رئوس گراف G
$deg_G(v)$	درجه‌ی راس v در گراف G
$nei(v)$	همسایه‌های راس v در گراف

فصل اول

شرح مسئله و روند کار

۱-۱ مقدمه

در این گزارش با استفاده از الگوریتم Minimax یک agent برای بازی Othello می‌سازیم. پیاده سازی و کد در اینجا قابل مشاهده می‌باشد. برای پیاده سازی engine بازی و رابط کاربری از Python و برای پیاده سازی الگوریتم از C++ استفاده شده است.

۱-۲ روش اجرا

برای اجرای برنامه نیاز به Python با حداقل ورژن 3.8 و gcc/g++ با c++17 دارید.

برای اجرای برنامه ابتدا نیاز به راه اندازی یک Virtual environment برای Python است. به این منظور در فولدر Report 04/game دستورات زیر را اجرا کنید.

```
cd Report\\ 04/game
python3 -m venv venv
```

برای activate کردن با توجه به سیتم عامل دستورات اینجا [۱] را اجرا کنید.

برای نصب پیشنیاز ها دستورات زیر را اجرا کنید.

```
python3 -m pip install -r requirements.txt
```

برای اضافه کردن ماژول Minimax باید کد c++ الگوریتم به ماژول Python تبدیل شود.

```
cd agent/MiniMaxCompEngine
python3 setup.py build
python3 setup.py install
cd ../../
```

سپس با اجرای فایل test01.py قادر به اجرای برنامه خواهید بود.

```
python3 test01.py
```

[۱] برای جزئیات بیشتر به اینجا مراجعه کنید

فصل دوم

پیاده‌سازی رابط کاربری و Engine بازی

۲-۱ Engine

کلاس Engine مسئولیت نگهداری موقعییت بازی را بر عهده دارد. اعزای این کلاس شامل boardSize, state, hisotry, availableMoves و توابع getAvailableMoves, makeMove می‌باشد. state یک dict شامل موقعیت کنونی بازی، امتیاز هر بازیکن، آخرین بازی کنی که حرکت کرده و اطلاعات پایان بازی می‌باشد. history یک لیست از موقعیت های کذشته و حرکات انجام شده است.

۲-۱-۱ تابع getAvailableMoves

تابع getAvailableMoves() با حرکت روی تمام خانه‌های خالی، چک می‌کند که اگر بازی‌کن کنونی این نقطه را انتخاب کند رنگ حداقل یک خانه‌ی همسایه تغییر می‌کند یا نه. سپس لیستی از حرکات ممکن برمیگرداند.

```
def getAvailableMoves(self):
    if self.availableMoves:
        return [tuple(i) for i in self.availableMoves]
    curPlayer =(PLAYERS.BLACK if
            self.state['lastPlayer'] ==PLAYERS.WHITE else PLAYERS.WHITE)
    availableMoves =[]

    for i in range(self.boardSize):
        for j in range(self.boardSize):
            if self.state['board'][i][j] !=PLAYERS.NONE:
                continue
            # Horizontal
            ok =False
            for jp in range(j +1, self.boardSize):
                if self.state['board'][i][jp] ==PLAYERS.NONE:
                    break
                elif self.state['board'][i][jp] ==curPlayer:
                    ok =jp >j +1
                    break
```

```
            if ok:
                availableMoves.append((i, j))
                continue
            for jp in range(j -1, -1, -1):
                if self.state['board'][i][jp] ==PLAYERS.NONE:
                    break
                elif self.state['board'][i][jp] ==curPlayer:
                    ok =jp <j -1
                    break
            if ok:
                availableMoves.append((i, j))
                continue


            # Vertical
            for ip in range(i +1, self.boardSize):
                if self.state['board'][ip][j] ==PLAYERS.NONE:
                    break
                elif self.state['board'][ip][j] ==curPlayer:
                    ok =ip >i +1
                    break
            if ok:
                availableMoves.append((i, j))
                continue
            for ip in range(i -1, -1, -1):
                if self.state['board'][ip][j] ==PLAYERS.NONE:
                    break
                elif self.state['board'][ip][j] ==curPlayer:
                    ok =ip <i -1
                    break
            if ok:
```

```
                availableMoves.append((i, j))
                continue

            # Diagonal
            for idel in range(-1, 2, 2):
                for jdel in range(-1, 2, 2):
                    for k in range(1, self.boardSize):
                        if i +idel *k >=self.boardSize or i +idel *k <0:
                            break
                        if j +jdel *k >=self.boardSize or j +jdel *k <0:
                            break
                        ip, jp =i +idel *k, j +jdel *k
                        if self.state['board'][ip][jp] ==PLAYERS.NONE:
                            break
                        elif self.state['board'][ip][jp] ==curPlayer:
                            ok =k >1
                            break
                    if ok: break
                if ok: break
            if ok:
                availableMoves.append((i, j))
                continue

        self.availableMoves =[tuple(i) for i in availableMoves]
        return availableMoves
```

۲-۱-۲ تابع makeMove

تابع makeMove اندیس از خروجی getAvailableMoves دریافت می‌کند و آن حرکت را برای بازی‌کن کنونی انجام می‌دهد. سپس چک می‌کند که آیا بازی به اتمام رسیده و آیا رقیب حرکت مجازی دارد یا نه و اگر حریف حرکت مجازی نداشت نوبت را به همان‌بازیکن بر میگرداند.

```
def makeMove(self, move):
    if self.state['gameEnded']:
        return False
    if move not in self.availableMoves:
        return False
    self.history.append({'move': move, 'board': self.state['board']})
    board =np.copy(self.state['board'])
    self.state['board'] =board
    curPlayer =(PLAYERS.BLACK if
           self.state['lastPlayer'] ==PLAYERS.WHITE else PLAYERS.WHITE)
    self.state['lastPlayer'] =curPlayer

    i, j =move
    board[i][j] =curPlayer
    for jp in range(j +1, self.boardSize):
        if self.state['board'][i][jp] ==PLAYERS.NONE:
            break
        elif self.state['board'][i][jp] ==curPlayer:
            for t in range(j +1, jp):
                self.state['board'][i][t] =curPlayer
            break

    for jp in range(j -1, -1, -1):
        if self.state['board'][i][jp] ==PLAYERS.NONE:
            break
        elif self.state['board'][i][jp] ==curPlayer:
            for t in range(j -1, jp, -1):
                self.state['board'][i][t] =curPlayer
            break


    # Vertical
```

```
for ip in range(i +1, self.boardSize):
   if self.state['board'][ip][j] ==PLAYERS.NONE:
      break
   elif self.state['board'][ip][j] ==curPlayer:
      for t in range(i +1, ip):
         self.state['board'][t][j] =curPlayer
      break
for ip in range(i -1, -1, -1):
   if self.state['board'][ip][j] ==PLAYERS.NONE:
      break
   elif self.state['board'][ip][j] ==curPlayer:
      for t in range(i -1, ip, -1):
         self.state['board'][t][j] =curPlayer
      ok =ip <i -1
      break

# Diagonal
for idel in range(-1, 2, 2):
   for jdel in range(-1, 2, 2):
      for k in range(1, self.boardSize):
         if i +idel *k >=self.boardSize or i +idel *k <0:
            break
         if j +jdel *k >=self.boardSize or j +jdel *k <0:
            break
         ip, jp =i +idel *k, j +jdel *k
         if self.state['board'][ip][jp] ==PLAYERS.NONE:
            break
         elif self.state['board'][ip][jp] ==curPlayer:
            for kp in range(1, k):
               self.state['board'][i +idel *kp][j +jdel *kp] =
                                                      curPlayer
            break
```

```
        self.availableMoves =None
        t =self.getAvailableMoves()

        ongoing =False
        blackScore, whiteScore =0, 0
        for ix, iy in np.ndindex((self.boardSize, self.boardSize)):
            if self.state['board'][ix, iy] ==PLAYERS.NONE:
                ongoing =True
            elif self.state['board'][ix, iy] ==PLAYERS.WHITE:
                whiteScore =whiteScore +1
            elif self.state['board'][ix, iy] ==PLAYERS.BLACK:
                blackScore =blackScore +1
        self.state['score']['black'], self.state['score']['white'] =
                                                blackScore, whiteScore
        if not ongoing or blackScore ==0 or whiteScore ==0:
            self.state['gameEnded'] =True

        if ongoing and len(t) ==0:
            # Skip One Move
            curPlayer =(PLAYERS.BLACK if
                    self.state['lastPlayer'] ==PLAYERS.WHITE else PLAYERS.WHITE)

            self.history.append({'move': None, 'board':
                                                    np.copy(self.state['board'])})
            self.state['lastPlayer'] =curPlayer

        return True
```

۲-۲ رابط کاربری CLI

برای پیاده سازی CLI [۱] از کتابخانه Rich [۲] استفاده شده که مسئولیت نشان دادن جدول و خروجی برنامه است. با اجرای این کد، کاربر ابتدا باید سایز برد و دو agent انتخاب کند. agent ها شامل کاربر و Minimax با عمق‌های متفاوت است. سپس تا زمانی که بازی تمام نشده با توجه به موقعیت engine به یک بازیکن اجازه‌ی حرکت می‌دهد.

۳-۲ Agent ها

هر Agent [۳] یک زیرکلاس AbstractAgent [۴] با دو تابع __init__(self, color) و move(self, engine) می‌باشد. به فراخوانی تابع move، یک اندیس از engine.getAvailableMoves باید بازگردانده شود.

```
class AbstractAgent:
   def __init__(self, color):
      self.color =color

   def move(self, engine):
      pass
```

[1] https://github.com/atrin-hojjat/Uni-AI-Course-Reports/blob/main/Report%2004/game/interface/CLI.py

[2] https://github.com/willmcgugan/rich

[3] https://github.com/atrin-hojjat/Uni-AI-Course-Reports/tree/main/Report%2004/game/agent

[4] https://github.com/atrin-hojjat/Uni-AI-Course-Reports/blob/main/Report%2004/game/agent/Agent.py

۲-۳-۱ Human Agent

کلاس Human Agent [۵] در هر حرکت یک prompt به کاربر نشان داده و شماره‌ی حرکت را از کاربر می‌پرسد. حرکت‌های ممکن و شماره‌هایشان در جدول نشان داده‌می‌شود.

```
from . import Agent
from rich.prompt import IntPrompt
from engine.OthelloEngine import PLAYERS

class HumanAgent(Agent.AbstractAgent):
    def __init__(self, color):
        self.color =color
        self.colorName ="Black" if PLAYERS.BLACK ==color else "White"

    def move(self, engine):
        moveNo =IntPrompt.ask(f"{self.colorName} to move")
        return moveNo
```

۲-۳-۲ Minimax Agent

کلاس MiniMaxAgent [۶] در ابتدا رنگ و عمق لازم را دریافت می‌کند. سپس برای هر حرکت موقعیت بازی را به تابع calculateMiniMax از ماژول MiniMax که به زبان C++ نوشته شده، ارسال می‌کند تا بهترین حرکت محاسبه شود. موقعیت بازی باید فرمت int32_t در C++ داشته باشد تا اطلاعات درست parse شوند.

```
from . import Agent
from rich.prompt import IntPrompt
from engine.OthelloEngine import PLAYERS
import numpy as np
```

[5] https://github.com/atrin-hojjat/Uni-AI-Course-Reports/blob/main/Report%2004/game/agent/Human.py

[6] https://github.com/atrin-hojjat/Uni-AI-Course-Reports/blob/main/Report%2004/game/agent/MiniMax.py

```
from numpy.ctypeslib import ndpointer
from MiniMax import calculateMiniMax

class MiniMaxAgent(Agent.AbstractAgent):
    depth =7
    def __init__(self, color, depth=7):
        self.color =color
        self.colorName ="Black" if PLAYERS.BLACK ==color else "White"
        self.depth =depth


    def move(self, engine):
        grid =np.copy(engine.state['board'].astype(np.int32))
        N =engine.boardSize

        result =calculateMiniMax(N, grid, self.color,
               self.depth)

        return engine.getAvailableMoves().index(result['move'])
```

۲-۳-۳ رجیستر کردن agentها

در فایل agent/__init__.py [۷] یک لیست از Agent های موجود برای صحولت کار CLI نگه داشته می‌شود. این لیست شامل یک Human Agent و یازده MiniMax Agent با عمق‌های متفاوت می‌باشد.

```
from . import Human, MiniMax


AGENTS =[
```

[7] https://github.com/atrin-hojjat/Uni-AI-Course-Reports/blob/main/Report%2004/game/agent/__init__.py

```
    ("Human", Human.HumanAgent),
    ("MiniMax agent(depth 2)", lambda col: MiniMax.MiniMaxAgent(color=col,
       depth=2)),
    ("MiniMax agent(depth 3)", lambda col: MiniMax.MiniMaxAgent(color=col,
       depth=3)),
    ("MiniMax agent(depth 4)", lambda col: MiniMax.MiniMaxAgent(color=col,
       depth=4)),
    ("MiniMax agent(depth 5)", lambda col: MiniMax.MiniMaxAgent(color=col,
       depth=5)),
    ("MiniMax agent(depth 6)", lambda col: MiniMax.MiniMaxAgent(color=col,
       depth=6)),
    ("MiniMax agent(depth 7)", lambda col: MiniMax.MiniMaxAgent(color=col,
       depth=7)),
    ("MiniMax agent(depth 8)", lambda col: MiniMax.MiniMaxAgent(color=col,
       depth=8)),
    ("MiniMax agent(depth 9)", lambda col: MiniMax.MiniMaxAgent(color=col,
       depth=9)),
    ("MiniMax agent(depth 10)", lambda col:
                                          MiniMax.MiniMaxAgent(color=col,
       depth=10)),
    ("MiniMax agent(depth 11 - up to 10 seconds)", lambda col:
                                          MiniMax.MiniMaxAgent(color=col,
       depth=11)),
    ("MiniMax agent(depth 12 - up to 100 seconds)", lambda col:
                                          MiniMax.MiniMaxAgent(color=col,
       depth=12)),
    ]
```

فصل سوم

پیاده‌سازی الگوریتم Minimax

منابع و مراجع

[1] Sasireka, A and Kishore, AH Nandhu. Applications of dominating set of a graph in computer networks. *Int. J. Eng. Sci. Res. Technol*, 3(1):170–173, 2014.

پیوست