**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

|  |  |  |
| --- | --- | --- |
| СОГЛАСОВАНО  Старший преподаватель департамента программной инженерии факультета компьютерных наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Д. В. Пантюхин  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г. |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия»  профессор департамента программной инженерии, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2019г. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | | **КОМПЬЮТЕРНАЯ СТРАТЕГИЧЕСКАЯ ИГРА "NERV" НА UNITY**  **Текст программы**  **ЛИСТ УТВЕРЖДЕНИЯ**  **RU.17701729.04.01-01 12** **01-1-ЛУ** | | |
|  |  | |
| Исполнитель  студент группы БПИ175  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / И. О. Балбин/  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г. | |
|  | | |
|  | |  |

**Москва 2018**

|  |  |  |  |
| --- | --- | --- | --- |
| УТВЕРЖДЕН  RU.17701729.04.01-01 12 01-1-ЛУ |  | |  |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | | **КОМПЬЮТЕРНАЯ СТРАТЕГИЧЕСКАЯ ИГРА "NERV" НА UNITY**  **Текст программы**  **RU.17701729.04.01-01 12** **01-1**  **Листов 86** | | | |
|  | |  | |
|  | |
|
|

**Москва 2019**

1. **Текст Программы**
   1. **Файл app.py**
2. #!/usr/bin/env python
3. **from** flask **import** Flask, render\_template, Response, request, redirect, url\_for
4. **from** camera\_chess **import** Camera, CameraMode
5. app = Flask(\_\_name\_\_)
6. **def** gen(camera):
7. **while** True:
8. frame = camera.get\_frame()
9. **yield** (b'--frame\r\n'
10. b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
11. @app.route('/video\_feed')
12. **def** video\_feed():
13. **return** Response(gen(Camera("http://100.66.103.27:8080/video")),
14. mimetype='multipart/x-mixed-replace; boundary=frame')
15. @app.route('/', methods=['POST', 'GET'])
16. **def** index():
17. **if** request.method == "POST":
18. video\_source = request.form.get("video\_source")
19. Camera.set\_video\_source(video\_source)
20. **if** Camera.check\_video\_source():
21. **return** redirect(url\_for('ches\_chessboard'), code=302)
22. **return** render\_template('index.html')
23. @app.route('/conf/chessboard', methods=['GET', 'POST'])
24. **def** ches\_chessboard():
25. Camera.board\_corner = None
26. Camera.M = None
27. **if** request.method == "POST":
28. Camera.find\_corner()
29. **return** redirect(url\_for('set\_position'), code=302)
30. Camera.set\_mode(CameraMode.FRAME\_REAL)
31. **return** render\_template('conf\_chess.html')
32. @app.route('/conf/chess', methods=['POST', 'GET'])
33. **def** set\_position():
34. **if** request.method == "POST":
35. **return** redirect(url\_for('chess\_game'), code=302)
36. **return** render\_template('set\_position.html')
37. @app.route('/game/chessboard')
38. **def** chess\_game():
39. **if** Camera.camera\_mode != CameraMode.FRAME\_GAME:
40. Camera.set\_mode(CameraMode.FRAME\_GAME)
41. **return** render\_template('chess\_game.html')
42. **if** \_\_name\_\_ == '\_\_main\_\_':
43. app.run(host='0.0.0.0', debug=True)

**1.2. Файл base\_camera.py**

**import** time

**import** threading

**try**:

**from** greenlet **import** getcurrent **as** get\_ident

**except** ImportError:

**try**:

**from** thread **import** get\_ident

**except** ImportError:

**from** \_thread **import** get\_ident

**class** CameraEvent(object):

"""An Event-like class that signals all active clients when a new frame is

    available.

    """

**def** \_\_init\_\_(self):

self.events = {}

**def** wait(self):

"""Invoked from each client's thread to wait for the next frame."""

ident = get\_ident()

**if** ident **not** **in** self.events:

# this is a new client

# add an entry for it in the self.events dict

# each entry has two elements, a threading.Event() and a timestamp

self.events[ident] = [threading.Event(), time.time()]

**return** self.events[ident][0].wait()

**def** set(self):

"""Invoked by the camera thread when a new frame is available."""

now = time.time()

remove = None

**for** ident, event **in** self.events.items():

**if** **not** event[0].isSet():

# if this client's event is not set, then set it

# also update the last set timestamp to now

event[0].set()

event[1] = now

**else**:

# if the client's event is already set, it means the client

# did not process a previous frame

# if the event stays set for more than 5 seconds, then assume

# the client is gone and remove it

**if** now - event[1] > 5:

remove = ident

**if** remove:

**del** self.events[remove]

**def** clear(self):

"""Invoked from each client's thread after a frame was processed."""

self.events[get\_ident()][0].clear()

**class** BaseCamera(object):

thread = None # background thread that reads frames from camera

frame = None # current frame is stored here by background thread

last\_access = 0 # time of last client access to the camera

event = CameraEvent()

break\_thread = False

**def** \_\_init\_\_(self):

"""Start the background camera thread if it isn't running yet."""

**if** BaseCamera.thread **is** None:

BaseCamera.last\_access = time.time()

# start background frame thread

BaseCamera.thread = threading.Thread(target=self.\_thread)

BaseCamera.thread.start()

# wait until frames are available

**while** self.get\_frame() **is** None:

time.sleep(0)

@staticmethod

**def** get\_frame():

"""Return the current camera frame."""

BaseCamera.last\_access = time.time()

# wait for a signal from the camera thread

BaseCamera.event.wait()

BaseCamera.event.clear()

**return** BaseCamera.frame

@staticmethod

**def** frames():

""""Generator that returns frames from the camera."""

**raise** RuntimeError('Must be implemented by subclasses.')

@classmethod

**def** \_thread(cls):

"""Camera background thread."""

**print**('Starting camera thread.')

frames\_iterator = cls.frames()

**for** frame **in** frames\_iterator:

BaseCamera.frame = frame

BaseCamera.event.set() # send signal to clients

time.sleep(0)

# if there hasn't been any clients asking for frames in

# the last 10 seconds then stop the thread

**if** time.time() - BaseCamera.last\_access > 10 **or** BaseCamera.break\_thread:

frames\_iterator.close()

**print**('Stopping camera thread due to inactivity.')

BaseCamera.break\_thread = False

**break**

BaseCamera.thread = None

**1.3. Файл chess\_camera.py**

**import** time

**import** cv2

**import** chess

**import** numpy **as** np

**import** chess.svg

**from** PIL **import** Image

**from** io **import** BytesIO

**import** cairosvg

**from** base\_camera **import** BaseCamera

# Translation mode

**class** CameraMode(object):

FRAME\_REAL = 0

FRAME\_BUBBLE = 1

FRAME\_GAME = 2

**class** Camera(BaseCamera):

video\_source = ''

width = 400

board\_corner = None

M = None

board = None

camera\_mode = CameraMode.FRAME\_REAL

**def** \_\_init\_\_(self, source='', board = None):

Camera.set\_video\_source(source)

Camera.set\_board(board)

super(Camera, self).\_\_init\_\_()

@staticmethod

**def** set\_board(board):

**if** board **is** None:

board = chess.Board()

Camera.board = board

@staticmethod

**def** frame\_real():

camera = cv2.VideoCapture(Camera.video\_source)

cv2.waitKey(0)

**if** **not** camera.isOpened():

**raise** RuntimeError('Could not start camera.')

**while** True:

# read current frame

\_, frame = camera.read()

**if** Camera.M **is** **not** None:

frame = cv2.warpPerspective(frame, Camera.M, (Camera.width, Camera.width))

**yield** frame

# yield cv2.imencode('.jpg', frame)[1].tobytes()

camera.grab()

@staticmethod

**def** set\_mode(camera\_mode):

Camera.camera\_mode = camera\_mode

**if** BaseCamera.thread **is** **not** None:

BaseCamera.break\_thread = True

BaseCamera.thread = None

@staticmethod

**def** set\_video\_source(source):

**if** source.isdigit():

Camera.video\_source = int(source)

**else**:

Camera.video\_source = source

@staticmethod

**def** frame\_diff(frame\_other):

kernel = np.ones((5, 5,))

**for** frame **in** Camera.frame\_real():

frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

diff = cv2.absdiff(frame, frame\_other)

diff = cv2.threshold(diff, 10, 150, cv2.THRESH\_BINARY)[1]

diff = cv2.erode(diff, kernel)

diff = cv2.dilate(diff, kernel)

**yield** diff

@staticmethod

**def** frames():

**print**('camera mode:', Camera.camera\_mode)

**if** Camera.camera\_mode == CameraMode.FRAME\_REAL:

**for** frame **in** Camera.frame\_real():

**yield** cv2.imencode('.jpg', frame)[1].tobytes()

# return Camera.frame\_real()

# elif Camera.camera\_mode == CameraMode.FRAME\_BUBBLE:

#

# cap = cv2.VideoCapture(Camera.video\_source)

# \_, first\_img = cap.read()

#

# first\_img = cv2.cvtColor(first\_img, cv2.COLOR\_BGR2GRAY)

# for frame in Camera.frame\_diff(first\_img):

# yield cv2.imencode('.jpg', frame)[1].tobytes()

**elif** Camera.camera\_mode == CameraMode.FRAME\_GAME:

**for** frame **in** Camera.\_game():

# yield frame

# return Camera.\_game()

**yield** cv2.imencode('.jpg', frame)[1].tobytes()

**raise** ValueError('Bad camera\_mode')

@staticmethod

**def** find\_corner():

camera = cv2.VideoCapture(Camera.video\_source)

ret = False

res = None

**for** \_ **in** range(100):

ret, first\_frame = camera.read()

ret, res = cv2.findChessboardCorners(first\_frame, (7, 7))

**if** ret:

**break**

**if** **not** ret:

**raise** ValueError('Chessboard dont on board or dont empty ')

res = res[:, 0, :]

min\_x, max\_x = res[:, 0].min(), res[:, 0].max()

min\_y, max\_y = res[:, 1].min(), res[:, 1].max()

dx = (max\_x - min\_x) / 6

dy = (max\_y - min\_y) / 6

board\_corner = np.array([

[min\_x - dx, max\_y + dy],

[min\_x - dx, min\_y - dy],

[max\_x + dx, max\_y + dy],

[max\_x + dx, min\_y - dy],

])

Camera.board\_corner = np.vstack((res, board\_corner))

pst1 = np.float32(board\_corner)

pst2 = np.float32([[0, 0], [Camera.width, 0], [0, Camera.width], [Camera.width, Camera.width]])

Camera.M = cv2.getPerspectiveTransform(pst1, pst2)

@staticmethod

**def** \_hand\_on(frame\_cur, frame\_last, thr=0.1):

diff = Camera.\_get\_frame\_diff(frame\_cur, frame\_last)

size = diff.shape[0]

**return** (diff != 0).sum() / (size \* size) > thr

@staticmethod

**def** \_get\_frame\_diff(frame1, frame2):

diff = cv2.absdiff(frame1, frame2)

diff = cv2.threshold(diff, 10, 150, cv2.THRESH\_BINARY)[1]

kernel = np.ones((5, 5,))

diff = cv2.erode(diff, kernel)

diff = cv2.dilate(diff, kernel)

**return** diff

@staticmethod

**def** \_get\_move(frame\_cur, frame\_last, thr=0.1):

diff = Camera.\_get\_frame\_diff(frame\_cur, frame\_last)

step = diff.shape[0] // 8

change\_square = []

**for** i **in** range(8):

**for** j **in** range(8):

sub\_diff = diff[step \* i:step \* (i + 1), step \* j:step \* (j + 1)]

**if** (sub\_diff != 0).sum() / (step \* step) > thr:

change\_square.append([i, j])

moves = []

**for** j, i **in** change\_square:

**for** l, k **in** change\_square:

moves.append(chess.Move(i + j \* 8, k + l \* 8))

res\_move = None

**for** move **in** moves:

**if** move **in** Camera.board.legal\_moves:

res\_move = move

**break**

**return** res\_move

@staticmethod

**def** check\_video\_source():

**print**(f"Check {Camera.video\_source}")

camera = cv2.VideoCapture(Camera.video\_source)

\_, t = camera.read()

cv2.waitKey(0)

res = camera.isOpened()

# camera.release()

**return** res

@staticmethod

**def** \_game():

**print**('In game')

**if** Camera.board **is** None:

**raise** ValueError('Miss Board')

frame\_last = next(Camera.frame\_real())

frame\_last = cv2.cvtColor(frame\_last, cv2.COLOR\_BGR2GRAY)

hand\_on\_last = False

**for** frame **in** Camera.frame\_real():

frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

hand\_on\_cur = Camera.\_hand\_on(frame, frame\_last)

hand\_on\_change = hand\_on\_cur ^ hand\_on\_last

hand\_on\_last = hand\_on\_cur

# print("hand on" if hand\_on\_cur else "hand off")

**if** hand\_on\_change **and** **not** hand\_on\_cur:

**print**('yo')

time.sleep(2)

frame = next(Camera.frame\_real())

frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

cur\_move = Camera.\_get\_move(frame, frame\_last)

**print**(cur\_move)

Camera.board.push(cur\_move)

svg\_board = chess.svg.board(Camera.board)

svg\_board = cairosvg.svg2png(svg\_board)

temp\_buff = BytesIO()

temp\_buff.write(svg\_board)

temp\_buff.seek(0)

im = Image.open(temp\_buff)

background = Image.new('RGBA', im.size, (255, 255, 255))

background.paste(im, im)

im\_jpg = np.array(background.convert('RGB'), dtype=np.uint8)

**yield** im\_jpg

**1.4 Файл camera\_opencv.py**

**import** cv2

**from** base\_camera **import** BaseCamera

**class** Camera(BaseCamera):

video\_source= 0

**def** \_\_init\_\_(self, source=0):

Camera.set\_video\_source(source)

super(Camera, self).\_\_init\_\_()

@staticmethod

**def** set\_video\_source(source):

Camera.video\_source = source

@staticmethod

**def** frames():

camera = cv2.VideoCapture(Camera.video\_source)

**if** **not** camera.isOpened():

**raise** RuntimeError('Could not start camera.')

**while** True:

# read current frame

\_, img = camera.read()

# encode as a jpeg image and return it

**yield** cv2.imencode('.jpg', img)[1].tobytes()

**1.5. Файл base.html**

<!DOCTYPE html>

<**html** lang="en">

<**head**>

<**meta** charset="UTF-8">

<**title**>Title</**title**>

<**link** rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">

<**script** src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></**script**>

<**script** src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></**script**>

<**script** src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></**script**>

{% block header %}

{% endblock %}

</**head**>

<**body**>

{% block content %}

{% endblock %}

</**body**>

</**html**>

**1.6 файл chess\_game.html**

{% extends 'base.html' %}

{% block content %}

<**img** src="{{ url\_for('video\_feed') }}" class="img-thumbnail" alt="board" width="400", height="400">

{% endblock %}

**1.6 файл conf\_chess.html**

{% extends 'base.html' %}

{% block content %}

<**img** src="{{ url\_for('video\_feed') }}" class="img-thumbnail" alt="board" width="400", height="400">

<**form** method="POST">

<**div** class="form-check form-check-inline">

<**div** class="form-check">

<**input** class="form-check-input" type="checkbox" value="" id="defaultCheck1" onchange="document.getElementById('success').disabled = !this.checked;">

<**label** class="form-check-label" for="defaultCheck1">

Clear board on the Camera.

</**label**>

<**button** type="submit" disabled class="btn btn-success" id="success">Success</**button**>

</**div**>

</**form**>

{% endblock %}

**1.7 Файл index.html**

{% extends 'base.html' %}

{% block content %}

<**form** method="POST" class="form-group">

<**label** for="video\_source">'Write url for IP camera</**label**>

<**input** class="form-control" type="text" name="video\_source" id="video\_source" placeholder="Default input">

<**button** type="submit" class="btn btn-success">Done</**button**>

</**form**>

{% endblock %}

**1.8 Файл set\_position.html**

{% extends 'base.html' %}

{% block content %}

<**form** method="POST" class="form-group">

<**label** for="video\_source">'Write url for IP camera</**label**>

<**input** class="form-control" type="text" name="video\_source" id="video\_source" placeholder="Default input">

<**button** type="submit" class="btn btn-success">Done</**button**>

</**form**>

{% endblock %}

# **ПРИЛОЖЕНИЕ 1**

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. ГОСТ 19.101-77 Виды программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
2. ГОСТ 19.102-77 Стадии разработки. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3. ГОСТ 19.103-77 Обозначения программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
4. ГОСТ 19.104-78 Основные надписи. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
5. ГОСТ 19.105-78 Общие требования к программным документам. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
6. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
7. ГОСТ 19.401-78 Текст программы. Требования к содержанию и оформлению. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
8. ГОСТ 19.603-78 Общие правила внесения изменений. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
9. ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

**ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводительного документа и дата | Подпись | Дата |
| Изм. | Измененных | Замененных | Новых | Аннулированных |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |