Veriliş Tarihi: 29.11.2018

Son Teslim Tarihi:27.12.2018

*YAPAY ZEKA-YZM 3217*

“Yapay Zeka Tabanlı Pacman Oyunu 2.Projesi”

HAZIRLAYANLAR:

152805010-Cennet Gizem Söylemez

152805009-Bekir Furkan Zadegil

142803015- ZülfüNur Eşit

“Yapay Zeka Tabanlı Pacman Oyunu 2.Projesi”

İçindekiler Tablosu

PROJENİN ADI 2

PROJE KONUSU

PROJE AMACI 2

**YAZILIM GELİŞTİRME SÜREÇLERİ** 2

PLANLAMA 3

PLANLAMA AŞAMASINDA YAPILAN ÇALIŞMALAR 3

**TASARIM VE GERÇEKLEŞTİRME** **3**

**VERİ KATOLOĞU**............................................................................................................4-5

PROJE KODLARI 1.Proje..……………………………………………………………....6-38

PROJE KODLARI 2.Proje..……………………………………………………….........39-56

Projenin Adı : “Yapay Zeka Tabanlı Pacman Oyunu”

Projenin Konusu: Proje kapsamında gerçekleştirilen Pacman etmeninin, labirent içerisinde yol bulabilme, belirli bir konuma ulaşabilme ve etkin bir biçimde yiyecekleri toplayabilmesi amaçlanmaktadır.

Projenin Amacı: MINIMAX Tabanlı PACMAN Etmeni, Alfa-Beta Budama Tabanlı PACMAN Etmeni, Expectimax Tabanlı PACMAN Etmeni kullanarak bu kavramları daha iyi öğrenmek .

Proje Metni: 1. MINIMAX Tabanlı PACMAN Etmeni: Hayaletlere karşı oyun oynayan PACMAN etmeninin “MINIMAX” algoritması ilkelerine göre eylemlerini gerçekleştirmesi. MINIMAX gerçekleştirimi, herhangi sayıda hayaletin varlığında çalışabilir olmalıdır. Dolayısı ile gerçekleştirimdeki MINIMAX ağacı, her bir hayalet için birden fazla “MIN” katmanına sahip olmalıdır. Gerçekleştirim, MINIMAX oyun ağacını, her çalıştırıldığında rastgele belirlenen belirli bir derinliğe kadar açarak, eylemlerine karar vermelidir. MINIMAX oyununda aşamaların değerlendirilmesi için uygun bir değerlendirme fonksiyonu belirlenip kullanılmalıdır.

2. Alfa-Beta Budama Tabanlı PACMAN Etmeni: Hayaletlere karşı oyun oynayan PACMAN etmeninin “Alfa-Beta Budama” algoritması ilkelerine göre eylemlerini gerçekleştirmesi. Gerçekleştirim, herhangi bir sayıda hayaletin varlığında çalışabilir olmalıdır.

3. Expectimax Tabanlı PACMAN Etmeni: Hayaletlere karşı oyun oynayan PACMAN etmeninin “EXPECTIMAX” algoritması ilkelerine göre eylemlerini gerçekleştirmesi.

# YAZILIM GELİŞTİRME SÜREÇLERİ

* Planlama
* Çözümleme
* Tasarım Ve Gerçekleştirme

# PLANLAMA:

Yazılım geliştirme sürecinin ilk unsuru planlamadır. Proje Planının temel amacı proje sahibinin gereksinimlerini listelemektir.

# Proje Planlama Aşamasında Yapılan Çalışmalar

**1-Proje Kaynaklarının Belirlenmesi:**

**Proje Ekibinin Oluşturulması:**

Projemiz 3 kişilik grup üyelerinden

1-Cennet Gizem Söylemez

2-Bekir Furkan Zadegil

3-Zülfü Nur Eşit

**2-Proje Zamanlaması:**

Projemiz 29 Kasım - 27 Aralık tarihleri arasında yapılmıştır. Beraber belli günler belirleyip buluşulup proje bitirilmiştir.

**3-Yazılım Araçları:**

* Pycharm ile yazılmıştır.

# **ÇÖZÜMLEME**

Çözümleme aşaması Üretimin ilk basamağıdır.

Üretim süreci çözümleme aşamasıyla başlamıştır. Gerekli olan işlevsel veya işlevsel olmayan gereksinimler belirlenip projede üretime geçilmiştir.

# **TASARIM VE GERÇEKLEŞTİRME**

Yazılım tasarımı, bir sorunun yazılım çözümü için, sorun giderme ve planlama aşamasıdır. Gerçekleştirme aşamasından önce yapılan tasarım aşamasıyla gerçekleştirme aşamasının ilk girişi yapılır. Tasarım şekli belirlendikten sonra gerçekleştirilmeye başlanır.

VERİ YAPILARI KATALOĞU

Kullanılan Veri Yapıları:

* MiniMax Algoritma
* Alfa-Beta Budama
* Expectimax

# **MiniMax algoritma:**

Bilgisayar mühendisliğinde, yapay zeka konusunda kullanılan bir karar ağacı türüdür. Aslında minimax ağaçları bilgisayar bilimlerine işletme bilimindeki oyun teorisinden (game theory) girmiştir.

Temel olarak sıfır toplamlı bir oyunda (zero sum game), yani birisinin kaybının başka birisinin kazancı olduğu (veya tam tersi) oyunlarda karar vermek için kullanılışlıdırlar. Örneğin çoğu masa oyunu (satranç, othello, tictactoe gibi) veya çoğu finansal oyunlar (borsa gibi) veya çoğu kumar oyunları sıfır toplamlı oyunlar arasında sayılabilir (yani birisinin kaybı başka birisinin kazancıdır ve sonuçta toplam sıfır olur).

Yukarıda bahsedilen bu oyunlarda doğru karar verilmesini sağlayan minimax ağacı basitçe kaybı asgariye indirmeye (mimize etmeye) ve dolayısyıla kazancı azamiye çıkarmaya (maximize etmeye) çalışır.

Ağaç basitçe her [düğümde (node)](http://www.bilgisayarkavramlari.com/2008/08/01/dugum-node/) farklı alternatiflerin değerlerini hesaplar. Son düğümden (yapraklardan ,leaf) yukarıya doğru değerleri seçerek gelir ve en sonunda bütün ağaçtaki en doğru seçenek seçilmiş olur.

**ALFA-BETA BUDAMA:**

Yazının konusu olan alfa beta budaması (alpha beta prunning) [minimax ağaçlarında (minimax trees)](http://www.bilgisayarkavramlari.com/2009/04/29/minimax-agaclari-minimax-tree/)kullanılır. Oyundaki hamle sayısı sınırlı olduğundan bilgisayarın hesaplaması açısından sorun oluşturmaz. Ancak hamle sayısının çok fazla olduğu satranç veya GO gibi oyunlar düşünülürse bütün hamlelerin hesaplanması bilgisayarlar için (günümüz teknolojisinde hâlâ ) imkansızdır. Bu durumda hesaplamayı kolaylaştırmak için bu ağaç üzerinde bir iyileştirilmeye gidilmesi gerekir.

İşte tam bu noktada alfa beta budaması devreye girer. Ağacın bazı dallarının işlenmesi ve hamlelerinin sonucunun hesaplanması gereksizdir. Alfa beta budaması bu dalları tespit ederek budar. Yani bu dalları ve altlarını hesaplamayarak performans artışı hedeflenir.

**EXPECTİMAX ALGORİTMA:**

Bir bekleyişim ağacı, iki oyunculu sıfır toplamlı oynayan yapay zeka sistemlerinde kullanılmak üzere özel bir minimax oyun ağacının bir varyasyonudur.

# YAZILIM GELİŞTİRME İÇİN HARCANAN GÜNLER

**Grup Elemanları Görev Paylaşımı:**

* Cennet Gizem Söylemez: 7 gün
* Bekir Furkan Zadegil:7 gün
* Zülfü Nur Eşit :5 gün

Kodlar 1.Proje:

Pacman.py:

**import** sys  
**import** logging  
**import** logging.handlers  
**from** BaseHTTPServer **import** BaseHTTPRequestHandler, HTTPServer  
**from** optparse **import** OptionParser  
**from** SocketServer **import** ThreadingMixIn  
**from** threading **import** Event  
**from** threading **import** activeCount  
**import** select  
**import** socket  
**from** urlparse **import** urlparse, urlunparse  
**from** types **import** FrameType, CodeType  
**from** signal **import** signal, SIGINT  
  
**try**:  
 **import** yaml  
**except** ImportError:  
  
USAGE = **"""\  
pacman  
  
pacman [SECENEKLER] ...  
  
description: çapraz etki alanı uygulamasını çivi altına almak için bir python web proxy sunucusu  
gelişme. çapraz etki alanı için en önemli araçlardan biri  
dirilişlerden beri uygulama araç kemeri n 'bok.  
SEÇENEKLER:  
 --host=HOST proxy sunucusunu bağlamak için ana bilgisayar adı.  
 --port=PORT proxy sunucusunu bağlamak için bağlantı noktası.  
 --max-threads Maksimum aktif iplik sayısı.  
 --log-level çıkış seviyesini kaydetme  
 --proxy-config proxy yapılandırması yaml dosyası.  
"""**CONFIG\_PATH = **"pacman.yaml"***# pylint #E501 disable*LOG\_FMT = **"[%(asctime)-12s] %(levelname)-8s {%(name)s \  
%(threadName)s} %(message)s "  
  
  
def** load\_proxy\_config(path):  
  
 **with** open(path, **'r'**) **as** f:  
 result = yaml.safe\_load(f)  
 f.close()  
 **return** result  
  
  
**class** ProxyHandler(BaseHTTPRequestHandler):  
  
 **def** \_\_init\_\_(self, \*args, \*\*kwargs):  
 BaseHTTPRequestHandler.\_\_init\_\_(self, \*args, \*\*kwargs)  
 self.protocol = **"HTTP/1.0"** self.protocol\_scheme = **'http'** self.\_proxy\_config = **None** self.\_proxy\_rules = **None** self.\_pacfile\_config = **None** @property  
 **def** proxy\_config(self):  
 **if not** hasattr(self, **'\_proxy\_config'**) **or** self.\_proxy\_config **is None**:  
 self.\_proxy\_config = load\_proxy\_config(CONFIG\_PATH)  
 **return** self.\_proxy\_config  
  
 @property  
 **def** proxy\_rules(self):  
 **if not** hasattr(self, **'\_proxy\_rules'**) **or** self.\_proxy\_rules **is None**:  
 self.\_proxy\_rules = self.proxy\_config.get(**'proxy\_rules'**)  
 **return** self.\_proxy\_rules  
  
 @property  
 **def** pacfile\_config(self):  
 **if not** hasattr(self, **'\_pacfile\_config'**) **or** self.\_pacfile\_config **is None**:  
 self.\_pacfile\_config = self.proxy\_config.get(**'pacfile'**)  
 **return** self.\_pacfile\_config  
  
  
 **def** handle(self):  
 *"""  
 tüm istekleri ele alır  
 """* (ip, port) = self.client\_address  
 self.server.logger.info(**"handling request from ip: {}"**.format(ip))  
  
 **if** hasattr(self, **'allowed\_clients'**) **and** ip **not in** self.allowed\_clients:  
 self.raw\_requestline = self.rfile.readline()  
 **if** self.parse\_request():  
 self.send\_error(403)  
  
 **else**:  
 **return** BaseHTTPRequestHandler.handle(self)  
  
  
 **def** render\_pacfile(self):  
 *"""  
Proxy pacfile, proxy yapılandırmasında tanımlardan oluşur.  
 """* proxy\_rule = **"""  
 if (shExpMatch(url, "\*{name}\*"))  
 return "PROXY {forward\_host}:{forward\_port}";  
 """** proxy\_rules = [proxy\_rule.format(\*\*host) **for** host **in** self.proxy\_rules]  
  
 body = **"""  
 fonksiyon FindProxyForURL(url, host) {  
 {}  
 return "DIRECT";  
 }  
 """**.format(**''**.join(proxy\_rules))  
  
 self.wfile.write(body)  
  
  
 **def** urlparse(self, scheme, host, path, params, query, fragment):  
 *"""  
 urlparse () 'nin sonucunu bir dict olarak ayrıştırır.  
 """* **return** {  
 **'scheme'**: scheme,  
 **'host'**: host,  
 **'path'**: path,  
 **'params'**: params,  
 **'query'**: query,  
 **'fragment'**: fragment}  
  
  
 **def** urlunparse(self, request):  
 urlunparse(  
 (**''**, **''**, request[**'path'**], request[**'params'**], request[**'query'**], **''**))  
  
  
 **def** do\_GET(self):  
 *"""  
 bir GET isteği işleyebilir  
 """* request = self.urlparse(\*urlparse(self.path, self.protocol\_scheme))  
  
 **if** self.is\_pacfile\_path():  
 **return** self.render\_pacfile()  
  
 **if** request[**'fragment'**]:  
 self.send\_error(400, **"unsupported url: {}"**.format(self.path))  
 **return  
  
 if** request[**'scheme'**] != **'http'**:  
 self.send\_error(400, **"unsupported url: {}"**.format(self.path))  
 **return  
  
 if not** request[**'host'**]:  
 self.send\_error(400, **"bad url: {}"**.format(self.path))  
 **return** self.send\_proxy\_request(request, self.proxy\_rules[request[**'host'**]])  
  
  
 **def** is\_pacfile\_path(self):  
 *"""  
 Geçerli istek yolu pacfile ise bir boolean döndürür.  
 """* **return** self.path.lower() == self.pacfile\_config[**'path'**].lower()  
  
  
 **def** send\_proxy\_request(self, request, forward\_proxy):  
 *"""  
 vekil bir istekte bulunur.  
 """* body = **''** length = self.headers.get(**"Content-Length"**, 0)  
 **if** length > 0:  
 body = self.rfile.read(int(length))  
 self.server.logger.debug(  
 **"(content-length {}): "**.format(length, body))  
  
 proxy\_s = self.create\_proxy\_socket()  
 **try**:  
 **if** self.proxy\_socket\_connect(proxy\_s, forward\_proxy.port):  
 self.log\_request()  
  
 proxy\_s.send(**"{} {} {}\r\n"**.format(  
 self.command, self.urlunparse(request), self.protocol))  
  
 self.headers[**'Connection'**] = **'close'  
 del** self.headers[**'Proxy-Connection'**]  
 self.server.logger.debug(**'URL: {} request'**.format(self.path))  
  
 [proxy\_s.send(**"{}: {}\r\n"**.format(\*kv)) **for** kv **in** self.headers.items()]  
  
 proxy\_s.send(**"\r\n{}\r\n"**.format(body))  
 self.socket\_rw(proxy\_s)  
 **except** Exception, e:  
 self.server.logger.exception(e)  
  
 **finally**:  
 proxy\_s.close()  
 self.connection.close()  
  
  
 **def** create\_proxy\_socket(self):  
 **return** socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
  
  
 **def** proxy\_socket\_connect(self, proxy\_s, port):  
 self.server.logger.debug(**"connect to {}:{}"**.format(proxy\_s, port))  
 **try**:  
 proxy\_s.connect(port)  
  
 **except** socket.error, arg;  
 self.server.logger.exception(e)  
 **try**:  
 msg = arg[1]  
 **except**:  
 msg = arg  
 self.send\_error(404, msg)  
 **return** 0  
  
 **return** 1  
  
  
 **def** socket\_rw(self, proxy\_s, max\_idle\_ticks=20):  
 iw = [self.connection, proxy\_s]  
 ow = []  
 timer = 0  
  
 **while** 1:  
 timer += 1  
 (ins, \_, exs) = select.select(iw, ow, iw, 1)  
 **if** exs:  
 **break  
  
 if** ins:  
 **for** i **in** ins:  
 **if** i **is** proxy\_s:  
 out = self.connection  
 **else**:  
 out = proxy\_s  
  
 data = i.recv(8192)  
 **if** data:  
 out.send(data)  
 timer = 0  
  
 **if** timer == max\_idle\_ticks:  
 **break** do\_PUT = do\_GET  
 do\_HEAD = do\_GET  
 do\_POST = do\_GET  
 do\_DELETE = do\_GET  
 do\_OPTIONS = do\_GET  
  
  
**class** ProxyHTTPServer(ThreadingMixIn, HTTPServer):  
 **def** \_\_init\_\_(self, server\_address, request\_handler\_class):  
 HTTPServer.\_\_init\_\_(self, server\_address, request\_handler\_class)  
 self.logger = self.create\_logger()  
  
 **def** create\_logger(self):  
 rv = logging.getLogger(self.\_\_class\_\_.\_\_name\_\_.split(**'.'**)[-1])  
 rv.setLevel(logging.INFO)  
 handler = logging.StreamHandler()  
 formatter = logging.Formatter(LOG\_FMT)  
 handler.setFormatter(formatter)  
 rv.addHandler(handler)  
 **return** rv  
  
  
**class** ProxyContext(object):  
 **def** \_\_init\_\_(self, host, port, max\_threads):  
  
 self.host = host  
 self.port = port  
 self.max\_threads = max\_threads  
 self.active\_thread\_count = 0  
 self.handler\_class = ProxyHandler  
 self.set\_sigint\_handler()  
  
  
 @property  
 **def** hostname(self):  
 **return** (self.host, self.port)  
  
  
 **def** run(self):  
 self.server = ProxyHTTPServer(self.hostname, self.handler\_class)  
  
 host, port = self.server.socket.getsockname()[:2]  
 self.server.logger.info(  
 **"serving http on host: {} port: {}"**.format(host, port))  
  
 self.active\_thread\_count = 0  
  
 **while not** self.sigint\_event.isSet():  
 **try**:  
 self.server.handle\_request()  
 self.active\_thread\_count = activeCount()  
 self.server.logger.info(  
 **"active thread count: {}"**.format(self.active\_thread\_count))  
  
 **if** self.active\_thread\_count >= self.max\_threads:  
 self.on\_max\_threads()  
  
 **except** select.error, err:  
 self.server.logger.debug(**"select.error: {}"**.format(e))  
 self.on\_sigint(\*err)  
  
 self.server.logger.info(**"proxy server shutdown.."**)  
  
  
 **def** on\_max\_threads(self):  
 self.server.logger.warn(  
 **"maksimum etkin iş parçacığına ulaşıldı. sayımı sıfırla .."**)  
 *#* ***todo: Buraya gelmek için daha fazla eylem?*** self.active\_thread\_count = 0  
  
  
 **def** set\_sigint\_handler(self):  
 *"""  
 'KeyboardInterrupt' olaylarını dinlemek için bir sinyali bağlar.  
 """* self.sigint\_event = Event()  
 signal(SIGINT, self.sigint\_handler)  
  
  
 **def** sigint\_handler(self, sigint, frame):  
 *"""  
 KeyboardInterrupt etkinlikleri için olay işleyici yöntemi.  
 """* self.server.logger.debug(  
 **"SIGINT event received: {}, {}"**.format(sigint, frame))  
  
 **while** frame **and** isinstance(frame, FrameType):  
 **if** frame.f\_code **and** isinstance(frame.f\_code, CodeType):  
 *# bu ProxyContext örneğini bulmak için yığından gider* **if "self" in** frame.f\_code.co\_varnames:  
 \_self = frame.f\_locals[**"self"**]  
 **if** isinstance(\_self, ProxyContext):  
 \_self.sigint\_event.set()  
 exit(0)  
 frame = frame.f\_back  
  
  
 **def** on\_sigint(self, code, msg, \*args):  
  
 self.server.logger.debug(**"on\_sigint, kodu: {} msg: {}"**.format(code, msg))  
  
 **if** code != 4 **and not** self.sigint\_event.isSet():  
 self.server.logger.critical(**"on\_sigint hatası, kodu: {}, {}"**.format(code, msg))  
 self.fail(msg)  
  
  
**def** fail(message, \*args):  
 print >> sys.stderr, **'error:'**, message.format(\*args),  
 exit(1)  
  
  
**def** usage():  
 print >> sys.stdout, USAGE,  
  
  
parser = OptionParser(  
 description=**"""description: çapraz etki alanı oluşturmak için bir python web proxy sunucusu  
uygulama geliştirme."""**,  
 prog=**"pacman"**,  
 usage=**"%prog [OPTIONS]"**)  
  
parser.add\_option(  
 **"--host"**, dest=**"host"**, default=**"localhost"**,  
 help=**"proxy sunucusunu bağlamak için ana bilgisayar adı."**)  
  
parser.add\_option(  
 **"--port"**, dest=**"port"**, default=3128,  
 help=**"proxy sunucusunu bağlamak için bağlantı noktası."**)  
  
parser.add\_option(  
 **"--max-threads"**, dest=**"max\_threads"**, default=1000,  
 help=**"aktif ipliklerin maksimum sayısı."**)  
  
parser.add\_option(  
 **"--proxy-config"**, dest=**"proxy\_config"**, default=**"pacman.yaml"**,  
 help=**"proxy pacfile yapılandırması dosyası."**)  
  
parser.add\_option(  
 **"--log-level"**, dest=**"log\_level"**, metavar=**"LEVEL"**,  
 help=**"çıkış seviyesi kaydediliyor."**)  
  
  
**def** main():  
 (options, args) = parser.parse\_args()  
  
 ProxyContext(  
 options.host, options.port, options.max\_threads).run()  
 exit(0)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()

game.py:

**from** util **import** \*  
**from** util **import** raiseNotDefined  
**import** time, os  
**import** traceback  
  
**try**:  
 **import** boinc  
 \_BOINC\_ENABLED = **True  
except**:  
 \_BOINC\_ENABLED = **False  
  
class** Agent:  
  
 **def** \_\_init\_\_(self, index=0):  
 self.index = index  
  
 **def** getAction(self, state):  
  
 raiseNotDefined()  
  
**class** Directions:  
 NORTH = **'North'** SOUTH = **'South'** EAST = **'East'** WEST = **'West'** STOP = **'Stop'** LEFT = {NORTH: WEST,  
 SOUTH: EAST,  
 EAST: NORTH,  
 WEST: SOUTH,  
 STOP: STOP}  
  
 RIGHT = dict([(y,x) **for** x, y **in** LEFT.items()])  
  
 REVERSE = {NORTH: SOUTH,  
 SOUTH: NORTH,  
 EAST: WEST,  
 WEST: EAST,  
 STOP: STOP}  
  
**class** Configuration:  
  
 **def** \_\_init\_\_(self, pos, direction):  
 self.pos = pos  
 self.direction = direction  
  
 **def** getPosition(self):  
 **return** (self.pos)  
  
 **def** getDirection(self):  
 **return** self.direction  
  
 **def** isInteger(self):  
 x,y = self.pos  
 **return** x == int(x) **and** y == int(y)  
  
 **def** \_\_eq\_\_(self, other):  
 **if** other == **None**: **return False  
 return** (self.pos == other.pos **and** self.direction == other.direction)  
  
 **def** \_\_hash\_\_(self):  
 x = hash(self.pos)  
 y = hash(self.direction)  
 **return** hash(x + 13 \* y)  
  
 **def** \_\_str\_\_(self):  
 **return "(x,y)="**+str(self.pos)+**", "**+str(self.direction)  
  
 **def** generateSuccessor(self, vector):  
  
 x, y= self.pos  
 dx, dy = vector  
 direction = Actions.vectorToDirection(vector)  
 **if** direction == Directions.STOP:  
 direction = self.direction *# There is no stop direction* **return** Configuration((x + dx, y+dy), direction)  
  
**class** AgentState:  
 *"""  
 AgentStates bir ajanın durumunu (yapılandırma, hız, korkmuş, vb.) Tutar.  
 """* **def** \_\_init\_\_( self, startConfiguration, isPacman ):  
 self.start = startConfiguration  
 self.configuration = startConfiguration  
 self.isPacman = isPacman  
 self.scaredTimer = 0  
  
 **def** \_\_str\_\_( self ):  
 **if** self.isPacman:  
 **return "Pacman: "** + str( self.configuration )  
 **else**:  
 **return "Ghost: "** + str( self.configuration )  
  
 **def** \_\_eq\_\_( self, other ):  
 **if** other == **None**:  
 **return False  
 return** self.configuration == other.configuration **and** self.scaredTimer == other.scaredTimer  
  
 **def** \_\_hash\_\_(self):  
 **return** hash(hash(self.configuration) + 13 \* hash(self.scaredTimer))  
  
 **def** copy( self ):  
 state = AgentState( self.start, self.isPacman )  
 state.configuration = self.configuration  
 state.scaredTimer = self.scaredTimer  
 **return** state  
  
 **def** getPosition(self):  
 **if** self.configuration == **None**: **return None  
 return** self.configuration.getPosition()  
  
 **def** getDirection(self):  
 **return** self.configuration.getDirection()  
  
**class** Grid:  
  
 **def** \_\_init\_\_(self, width, height, initialValue=**False**, bitRepresentation=**None**):  
 **if** initialValue **not in** [**False**, **True**]: **raise** Exception(**'Grids can only contain booleans'**)  
 self.CELLS\_PER\_INT = 30  
  
 self.width = width  
 self.height = height  
 self.data = [[initialValue **for** y **in** range(height)] **for** x **in** range(width)]  
 **if** bitRepresentation:  
 self.\_unpackBits(bitRepresentation)  
  
 **def** \_\_getitem\_\_(self, i):  
 **return** self.data[i]  
  
 **def** \_\_setitem\_\_(self, key, item):  
 self.data[key] = item  
  
 **def** \_\_str\_\_(self):  
 out = [[str(self.data[x][y])[0] **for** x **in** range(self.width)] **for** y **in** range(self.height)]  
 out.reverse()  
 **return '\n'**.join([**''**.join(x) **for** x **in** out])  
  
 **def** \_\_eq\_\_(self, other):  
 **if** other == **None**: **return False  
 return** self.data == other.data  
  
 **def** \_\_hash\_\_(self):  
 base = 1  
 h = 0  
 **for** l **in** self.data:  
 **for** i **in** l:  
 **if** i:  
 h += base  
 base \*= 2  
 **return** hash(h)  
  
 **def** copy(self):  
 g = Grid(self.width, self.height)  
 g.data = [x[:] **for** x **in** self.data]  
 **return** g  
  
 **def** deepCopy(self):  
 **return** self.copy()  
  
 **def** shallowCopy(self):  
 g = Grid(self.width, self.height)  
 g.data = self.data  
 **return** g  
  
 **def** count(self, item =**True** ):  
 **return** sum([x.count(item) **for** x **in** self.data])  
  
 **def** asList(self, key = **True**):  
 list = []  
 **for** x **in** range(self.width):  
 **for** y **in** range(self.height):  
 **if** self[x][y] == key: list.append( (x,y) )  
 **return** list  
  
 **def** packBits(self):  
  
 bits = [self.width, self.height]  
 currentInt = 0  
 **for** i **in** range(self.height \* self.width):  
 bit = self.CELLS\_PER\_INT - (i % self.CELLS\_PER\_INT) - 1  
 x, y = self.\_cellIndexToPosition(i)  
 **if** self[x][y]:  
 currentInt += 2 \*\* bit  
 **if** (i + 1) % self.CELLS\_PER\_INT == 0:  
 bits.append(currentInt)  
 currentInt = 0  
 bits.append(currentInt)  
 **return** tuple(bits)  
  
 **def** \_cellIndexToPosition(self, index):  
 x = index / self.height  
 y = index % self.height  
 **return** x, y  
  
 **def** \_unpackBits(self, bits):  
  
 cell = 0  
 **for** packed **in** bits:  
 **for** bit **in** self.\_unpackInt(packed, self.CELLS\_PER\_INT):  
 **if** cell == self.width \* self.height: **break** x, y = self.\_cellIndexToPosition(cell)  
 self[x][y] = bit  
 cell += 1  
  
 **def** \_unpackInt(self, packed, size):  
 bools = []  
 **if** packed < 0: **raise** ValueError, **"pozitif bir tam sayı olmalı"  
 for** i **in** range(size):  
 n = 2 \*\* (self.CELLS\_PER\_INT - i - 1)  
 **if** packed >= n:  
 bools.append(**True**)  
 packed -= n  
 **else**:  
 bools.append(**False**)  
 **return** bools  
  
**def** reconstituteGrid(bitRep):  
 **if** type(bitRep) **is not** type((1,2)):  
 **return** bitRep  
 width, height = bitRep[:2]  
 **return** Grid(width, height, bitRepresentation= bitRep[2:])  
  
**class** Actions:  
  
 *# Talimatlar* \_directions = {Directions.NORTH: (0, 1),  
 Directions.SOUTH: (0, -1),  
 Directions.EAST: (1, 0),  
 Directions.WEST: (-1, 0),  
 Directions.STOP: (0, 0)}  
  
 \_directionsAsList = \_directions.items()  
  
 TOLERANCE = .001  
  
 **def** reverseDirection(action):  
 **if** action == Directions.NORTH:  
 **return** Directions.SOUTH  
 **if** action == Directions.SOUTH:  
 **return** Directions.NORTH  
 **if** action == Directions.EAST:  
 **return** Directions.WEST  
 **if** action == Directions.WEST:  
 **return** Directions.EAST  
 **return** action  
 reverseDirection = staticmethod(reverseDirection)  
  
 **def** vectorToDirection(vector):  
 dx, dy = vector  
 **if** dy > 0:  
 **return** Directions.NORTH  
 **if** dy < 0:  
 **return** Directions.SOUTH  
 **if** dx < 0:  
 **return** Directions.WEST  
 **if** dx > 0:  
 **return** Directions.EAST  
 **return** Directions.STOP  
 vectorToDirection = staticmethod(vectorToDirection)  
  
 **def** directionToVector(direction, speed = 1.0):  
 dx, dy = Actions.\_directions[direction]  
 **return** (dx \* speed, dy \* speed)  
 directionToVector = staticmethod(directionToVector)  
  
 **def** getPossibleActions(config, walls):  
 possible = []  
 x, y = config.pos  
 x\_int, y\_int = int(x + 0.5), int(y + 0.5)  
  
 *# Kılavuz noktaları arasında, tüm aracılar düz devam etmelidir* **if** (abs(x - x\_int) + abs(y - y\_int) > Actions.TOLERANCE):  
 **return** [config.getDirection()]  
  
 **for** dir, vec **in** Actions.\_directionsAsList:  
 dx, dy = vec  
 next\_y = y\_int + dy  
 next\_x = x\_int + dx  
 **if not** walls[next\_x][next\_y]: possible.append(dir)  
  
 **return** possible  
  
 getPossibleActions = staticmethod(getPossibleActions)  
  
 **def** getLegalNeighbors(position, walls):  
 x,y = position  
 x\_int, y\_int = int(x + 0.5), int(y + 0.5)  
 neighbors = []  
 **for** dir, vec **in** Actions.\_directionsAsList:  
 dx, dy = vec  
 next\_x = x\_int + dx  
 **if** next\_x < 0 **or** next\_x == walls.width: **continue** next\_y = y\_int + dy  
 **if** next\_y < 0 **or** next\_y == walls.height: **continue  
 if not** walls[next\_x][next\_y]: neighbors.append((next\_x, next\_y))  
 **return** neighbors  
 getLegalNeighbors = staticmethod(getLegalNeighbors)  
  
 **def** getSuccessor(position, action):  
 dx, dy = Actions.directionToVector(action)  
 x, y = position  
 **return** (x + dx, y + dy)  
 getSuccessor = staticmethod(getSuccessor)  
  
**class** GameStateData:  
  
 **def** \_\_init\_\_( self, prevState = **None** ):  
 *"""  
Bir öncekinden bilgileri kopyalayarak yeni bir veri paketi oluşturur.  
 """* **if** prevState != **None**:  
 self.food = prevState.food.shallowCopy()  
 self.capsules = prevState.capsules[:]  
 self.agentStates = self.copyAgentStates( prevState.agentStates )  
 self.layout = prevState.layout  
 self.\_eaten = prevState.\_eaten  
 self.score = prevState.score  
 self.\_foodEaten = **None** self.\_capsuleEaten = **None** self.\_agentMoved = **None** self.\_lose = **False** self.\_win = **False** self.scoreChange = 0  
  
 **def** deepCopy( self ):  
 state = GameStateData( self )  
 state.food = self.food.deepCopy()  
 state.layout = self.layout.deepCopy()  
 state.\_agentMoved = self.\_agentMoved  
 state.\_foodEaten = self.\_foodEaten  
 state.\_capsuleEaten = self.\_capsuleEaten  
 **return** state  
  
 **def** copyAgentStates( self, agentStates ):  
 copiedStates = []  
 **for** agentState **in** agentStates:  
 copiedStates.append( agentState.copy() )  
 **return** copiedStates  
  
 **def** \_\_eq\_\_( self, other ):  
 *"""  
 Karşılaştırılacak iki durum sağlar.  
 """* **if** other == **None**: **return False** *#* ***TODO Check for type of other* if not** self.agentStates == other.agentStates: **return False  
 if not** self.food == other.food: **return False  
 if not** self.capsules == other.capsules: **return False  
 if not** self.score == other.score: **return False  
 return True  
  
 def** \_\_hash\_\_( self ):  
 **for** i, state **in** enumerate( self.agentStates ):  
 **try**:  
 int(hash(state))  
 **except** TypeError, e:  
 print e  
 *#hash(state)* **return** int((hash(tuple(self.agentStates)) + 13\*hash(self.food) + 113\* hash(tuple(self.capsules)) + 7 \* hash(self.score)) % 1048575 )  
  
 **def** \_\_str\_\_( self ):  
 width, height = self.layout.width, self.layout.height  
 map = Grid(width, height)  
 **if** type(self.food) == type((1,2)):  
 self.food = reconstituteGrid(self.food)  
 **for** x **in** range(width):  
 **for** y **in** range(height):  
 food, walls = self.food, self.layout.walls  
 map[x][y] = self.\_foodWallStr(food[x][y], walls[x][y])  
  
 **for** agentState **in** self.agentStates:  
 **if** agentState == **None**: **continue  
 if** agentState.configuration == **None**: **continue** x,y = [int( i ) **for** i **in** nearestPoint( agentState.configuration.pos )]  
 agent\_dir = agentState.configuration.direction  
 **if** agentState.isPacman:  
 map[x][y] = self.\_pacStr( agent\_dir )  
 **else**:  
 map[x][y] = self.\_ghostStr( agent\_dir )  
  
 **for** x, y **in** self.capsules:  
 map[x][y] = **'o'  
  
 return** str(map) + (**"\nScore: %d\n"** % self.score)  
  
 **def** \_foodWallStr( self, hasFood, hasWall ):  
 **if** hasFood:  
 **return '.'  
 elif** hasWall:  
 **return '%'  
 else**:  
 **return ' '  
  
 def** \_pacStr( self, dir ):  
 **if** dir == Directions.NORTH:  
 **return 'v'  
 if** dir == Directions.SOUTH:  
 **return '^'  
 if** dir == Directions.WEST:  
 **return '>'  
 return '<'  
  
 def** \_ghostStr( self, dir ):  
 **return 'G'  
 if** dir == Directions.NORTH:  
 **return 'M'  
 if** dir == Directions.SOUTH:  
 **return 'W'  
 if** dir == Directions.WEST:  
 **return '3'  
 return 'E'  
  
 def** initialize( self, layout, numGhostAgents ):  
 *"""  
Bir düzen dizisinden ilk oyun durumunu oluşturur  
 """* self.food = layout.food.copy()  
 self.capsules = layout.capsules[:]  
 self.layout = layout  
 self.score = 0  
 self.scoreChange = 0  
  
 self.agentStates = []  
 numGhosts = 0  
 **for** isPacman, pos **in** layout.agentPositions:  
 **if not** isPacman:  
 **if** numGhosts == numGhostAgents: **continue** *# Max hayaletler zaten ulaştı* **else**: numGhosts += 1  
 self.agentStates.append( AgentState( Configuration( pos, Directions.STOP), isPacman) )  
 self.\_eaten = [**False for** a **in** self.agentStates]  
  
**class** Game:  
 *"""  
Oyun, kontrol akışını yönetir, eylemleri ajanlardan talep eder.  
 """* **def** \_\_init\_\_( self, agents, display, rules, startingIndex=0, muteAgents=**False**, catchExceptions=**False** ):  
 self.agentCrashed = **False** self.agents = agents  
 self.display = display  
 self.rules = rules  
 self.startingIndex = startingIndex  
 self.gameOver = **False** self.muteAgents = muteAgents  
 self.catchExceptions = catchExceptions  
 self.moveHistory = []  
 self.totalAgentTimes = [0 **for** agent **in** agents]  
 self.totalAgentTimeWarnings = [0 **for** agent **in** agents]  
 self.agentTimeout = **False  
  
 def** getProgress(self):  
 **if** self.gameOver:  
 **return** 1.0  
 **else**:  
 **return** self.rules.getProgress(self)  
  
 **def** \_agentCrash( self, agentIndex, quiet=**False**):  
 **if not** quiet: traceback.print\_exc()  
 self.gameOver = **True** self.agentCrashed = **True** self.rules.agentCrash(self, agentIndex)  
  
 OLD\_STDOUT = **None** OLD\_STDERR = **None  
  
 def** mute(self):  
 **if not** self.muteAgents: **return  
 global** OLD\_STDOUT, OLD\_STDERR  
 **import** cStringIO  
 OLD\_STDOUT = sys.stdout  
 OLD\_STDERR = sys.stderr  
 sys.stdout = cStringIO.StringIO()  
 sys.stderr = cStringIO.StringIO()  
  
 **def** unmute(self):  
 **if not** self.muteAgents: **return  
 global** OLD\_STDOUT, OLD\_STDERR  
 sys.stdout.close()  
 sys.stderr.close()  
 sys.stdout = OLD\_STDOUT  
 sys.stderr = OLD\_STDERR  
  
  
 **def** run( self ):  
 *"""  
 Oyun oynamak için ana kontrol döngüsü.  
 """* self.display.initialize(self.state.data)  
 self.numMoves = 0  
  
 *# oyunun başlangıç ​​öğrenicilerini bilgilendir* **for** i **in** range(len(self.agents)):  
 agent = self.agents[i]  
 **if not** agent:  
 *# Bu boş bir ajan, yüklenemedi  
 # diğer takım kazanır* self.\_agentCrash(i, quiet=**True**)  
 **return  
 if** (**"registerInitialState" in** dir(agent)):  
 self.mute()  
 **if** self.catchExceptions:  
 **try**:  
 timed\_func = TimeoutFunction(agent.registerInitialState, int(self.rules.getMaxStartupTime(i)))  
 **try**:  
 start\_time = time.time()  
 timed\_func(self.state.deepCopy())  
 time\_taken = time.time() - start\_time  
 self.totalAgentTimes[i] += time\_taken  
 **except** TimeoutFunctionException:  
 print **"Ajan %d başlangıçta zaman tükendi!"** % i  
 self.unmute()  
 self.agentTimeout = **True** self.\_agentCrash(i, quiet=**True**)  
 **return  
 except** Exception,data:  
 self.unmute()  
 self.\_agentCrash(i, quiet=**True**)  
 **return  
 else**:  
 agent.registerInitialState(self.state.deepCopy())  
 *#* ***TODO: bu toplam süreyi aşabilir mi?*** self.unmute()  
  
 agentIndex = self.startingIndex  
 numAgents = len( self.agents )  
  
 **while not** self.gameOver:  
 *# Sonraki ajanı getir* agent = self.agents[agentIndex]  
 move\_time = 0  
 skip\_action = **False  
 if 'observationFunction' in** dir( agent ):  
 self.mute()  
 **if** self.catchExceptions:  
 **try**:  
 timed\_func = TimeoutFunction(agent.observationFunction, int(self.rules.getMoveTimeout(agentIndex)))  
 **try**:  
 start\_time = time.time()  
 observation = timed\_func(self.state.deepCopy())  
 **except** TimeoutFunctionException:  
 skip\_action = **True** move\_time += time.time() - start\_time  
 self.unmute()  
 **except** Exception,data:  
 self.unmute()  
 self.\_agentCrash(agentIndex, quiet=**True**)  
 **return  
 else**:  
 observation = agent.observationFunction(self.state.deepCopy())  
 self.unmute()  
 **else**:  
 observation = self.state.deepCopy()  
  
 *# Bir eylem talep et* action = **None** self.mute()  
 **if** self.catchExceptions:  
 **try**:  
 timed\_func = TimeoutFunction(agent.getAction, int(self.rules.getMoveTimeout(agentIndex)) - int(move\_time))  
 **try**:  
 start\_time = time.time()  
 **if** skip\_action:  
 **raise** TimeoutFunctionException()  
 action = timed\_func( observation )  
 **except** TimeoutFunctionException:  
 print **"Ajan %d tek bir hareketle zaman aşımına uğradı!"** % agentIndex  
 self.agentTimeout = **True** self.unmute()  
 self.\_agentCrash(agentIndex, quiet=**True**)  
 **return** move\_time += time.time() - start\_time  
  
 **if** move\_time > self.rules.getMoveWarningTime(agentIndex):  
 self.totalAgentTimeWarnings[agentIndex] += 1  
 print **"Ajan %d harekete geçmek için çok uzun sürdü! Bu uyarı %d"** % (agentIndex, self.totalAgentTimeWarnings[agentIndex])  
 **if** self.totalAgentTimeWarnings[agentIndex] > self.rules.getMaxTimeWarnings(agentIndex):  
 print **"Ajan %d, maksimum uyarı sayısını aştı: %d"** % (agentIndex, self.totalAgentTimeWarnings[agentIndex])  
 self.agentTimeout = **True** self.unmute()  
 self.\_agentCrash(agentIndex, quiet=**True**)  
  
 self.totalAgentTimes[agentIndex] += move\_time  
 **if** self.totalAgentTimes[agentIndex] > self.rules.getMaxTotalTime(agentIndex):  
 print **"Ajan %d zaman tükendi! (time: %1.2f)"** % (agentIndex, self.totalAgentTimes[agentIndex])  
 self.agentTimeout = **True** self.unmute()  
 self.\_agentCrash(agentIndex, quiet=**True**)  
 **return** self.unmute()  
 **except** Exception,data:  
 self.unmute()  
 self.\_agentCrash(agentIndex)  
 **return  
 else**:  
 action = agent.getAction(observation)  
 self.unmute()  
 self.moveHistory.append( (agentIndex, action) )  
 **if** self.catchExceptions:  
 **try**:  
 self.state = self.state.generateSuccessor( agentIndex, action )  
 **except** Exception,data:  
 self.\_agentCrash(agentIndex)  
 **return  
 else**:  
 self.state = self.state.generateSuccessor( agentIndex, action )  
  
 *# Ekranı değiştir* self.display.update( self.state.data )  
 *# Oyuna özel koşullara izin ver (kazanma, kaybetme vb.)* self.rules.process(self.state, self)  
 *# Parça ilerlemesi* **if** agentIndex == numAgents + 1: self.numMoves += 1  
 *# Sonraki ajan* agentIndex = ( agentIndex + 1 ) % numAgents  
  
 **if** \_BOINC\_ENABLED:  
 boinc.set\_fraction\_done(self.getProgress())  
  
 *# Oyun sonucunu öğrenen bir öğrencisini bilgilendirmek* **for** agent **in** self.agents:  
 **if "final" in** dir( agent ) :  
 **try**:  
 self.mute()  
 agent.final( self.state )  
 self.unmute()  
  
 self.display.finish()

search.py:

**class** SearchProblem:  
 *"""  
 Bu sınıfta hiç bir şeyi değiştirmene gerek yok.  
 """* **def** getStartState(self):  
 *"""  
 Arama sorununun başlangıç ​​durumunu döndürür.  
 """* util.raiseNotDefined()  
  
 **def** isGoalState(self, state):  
 *"""  
 Yalnızca, durum geçerli bir hedef durum olduğunda True değerini döndürür.  
 """* util.raiseNotDefined()  
  
 **def** getSuccessors(self, state):  
 *"""  
 state: Arama durumu  
 """* util.raiseNotDefined()  
  
 **def** getCostOfActions(self, actions):  
 *"""  
 Bu yöntem, belirli bir eylem dizisinin toplam maliyetini döndürür.  
 actions=eylemler anlamında  
 """* util.raiseNotDefined()  
  
  
**def** tinyMazeSearch(problem):  
 *"""  
 TinyMaze'yi çözen bir hareket dizisi döndürür.(küçük labirent)  
 """* **from** game **import** Directions  
 s = Directions.SOUTH  
 w = Directions.WEST  
 **return** [s, s, w, s, w, w, s, w]  
  
  
**def** depthFirstSearch(problem):  
 *"""Önce arama ağacındaki en derin düğümleri arayın."""* loc\_stack = Stack()  
 visited\_node = {}  
 parent\_child\_map = {}  
 direction\_list = []  
  
 start\_node = problem.getStartState()  
 parent\_child\_map[start\_node] = []  
 loc\_stack.push(start\_node)  
  
 **def** traverse\_path(parent\_node):  
 **while True**:  
 map\_row = parent\_child\_map[parent\_node]  
 **if** (len(map\_row) == 2):  
 parent\_node = map\_row[0]  
 direction = map\_row[1]  
 direction\_list.append(direction)  
 **else**:  
 **break  
 return** direction\_list  
  
 **while** (loc\_stack.isEmpty() == **False**):  
  
 parent\_node = loc\_stack.pop()  
  
 **if** (problem.isGoalState(parent\_node)):  
 pathlist = traverse\_path(parent\_node)  
 pathlist.reverse()  
 **return** pathlist  
  
 **elif** (visited\_node.has\_key(parent\_node) == **False**):  
 visited\_node[parent\_node] = []  
 sucessor\_list = problem.getSuccessors(parent\_node)  
 no\_of\_child = len(sucessor\_list)  
 **if** (no\_of\_child > 0):  
 temp = 0  
 **while** (temp < no\_of\_child):  
 child\_nodes = sucessor\_list[temp]  
 child\_state = child\_nodes[0];  
 child\_action = child\_nodes[1];  
 **if** (visited\_node.has\_key(child\_state) == **False**):  
 loc\_stack.push(child\_state)  
 parent\_child\_map[child\_state] = [parent\_node, child\_action]  
 temp = temp + 1  
  
  
**def** breadthFirstSearch(problem):  
 *"""Önce arama ağacındaki en sığ düğümleri arayın."""* loc\_queue = Queue()  
 visited\_node = {}  
 parent\_child\_map = {}  
 direction\_list = []  
  
 start\_node = problem.getStartState()  
 parent\_child\_map[start\_node] = []  
 loc\_queue.push(start\_node)  
  
 **def** traverse\_path(parent\_node):  
 **while True**:  
 map\_row = parent\_child\_map[parent\_node]  
 **if** (len(map\_row) == 2):  
 parent\_node = map\_row[0]  
 direction = map\_row[1]  
 direction\_list.append(direction)  
 **else**:  
 **break  
 return** direction\_list  
  
 **while** (loc\_queue.isEmpty() == **False**):  
  
 parent\_node = loc\_queue.pop()  
  
 **if** (problem.isGoalState(parent\_node)):  
 pathlist = traverse\_path(parent\_node)  
 pathlist.reverse()  
 **return** pathlist  
  
 **elif** (visited\_node.has\_key(parent\_node) == **False**):  
 visited\_node[parent\_node] = []  
 sucessor\_list = problem.getSuccessors(parent\_node)  
 no\_of\_child = len(sucessor\_list)  
 **if** (no\_of\_child > 0):  
 temp = 0  
 **while** (temp < no\_of\_child):  
 child\_nodes = sucessor\_list[temp]  
 child\_state = child\_nodes[0];  
 child\_action = child\_nodes[1];  
 **if** (visited\_node.has\_key(child\_state) == **False**):  
 loc\_queue.push(child\_state)  
 **if** (parent\_child\_map.has\_key(child\_state) == **False**):  
 parent\_child\_map[child\_state] = [parent\_node, child\_action]  
 temp = temp + 1  
  
  
**def** uniformCostSearch(problem):  
 loc\_pqueue = PriorityQueue()  
 visited\_node = {}  
 parent\_child\_map = {}  
 direction\_list = []  
 path\_cost = 0  
  
 start\_node = problem.getStartState()  
 parent\_child\_map[start\_node] = []  
 loc\_pqueue.push(start\_node, path\_cost)  
  
 **def** traverse\_path(parent\_node):  
 **while True**:  
 map\_row = parent\_child\_map[parent\_node]  
 **if** (len(map\_row) == 3):  
 parent\_node = map\_row[0]  
 direction = map\_row[1]  
 direction\_list.append(direction)  
 **else**:  
 **break  
 return** direction\_list  
  
 **while** (loc\_pqueue.isEmpty() == **False**):  
  
 parent\_node = loc\_pqueue.pop()  
  
 **if** (parent\_node != problem.getStartState()):  
 path\_cost = parent\_child\_map[parent\_node][2]  
  
 **if** (problem.isGoalState(parent\_node)):  
 pathlist = traverse\_path(parent\_node)  
 pathlist.reverse()  
 **return** pathlist  
  
 **elif** (visited\_node.has\_key(parent\_node) == **False**):  
 visited\_node[parent\_node] = []  
 sucessor\_list = problem.getSuccessors(parent\_node)  
 no\_of\_child = len(sucessor\_list)  
 **if** (no\_of\_child > 0):  
 temp = 0  
 **while** (temp < no\_of\_child):  
 child\_nodes = sucessor\_list[temp]  
 child\_state = child\_nodes[0];  
 child\_action = child\_nodes[1];  
 child\_cost = child\_nodes[2];  
 gvalue = path\_cost + child\_cost  
 **if** (visited\_node.has\_key(child\_state) == **False**):  
 loc\_pqueue.push(child\_state, gvalue)  
 **if** (parent\_child\_map.has\_key(child\_state) == **False**):  
 parent\_child\_map[child\_state] = [parent\_node, child\_action, gvalue]  
 **else**:  
 **if** (child\_state != start\_node):  
 stored\_cost = parent\_child\_map[child\_state][2]  
 **if** (stored\_cost > gvalue):  
 parent\_child\_map[child\_state] = [parent\_node, child\_action, gvalue]  
 temp = temp + 1  
  
  
**def** nullHeuristic(state, problem=**None**):  
 *"""  
 Sezgisel bir fonksiyon, mevcut durumdan maliyeti en yakın olana kadar tahmin eder.  
 """* **return** 0  
  
  
**def** aStarSearch(problem, heuristic=nullHeuristic):  
 *"""İlk olarak en düşük maliyeti ve sezgisel olan düğümü arayalım."""* loc\_pqueue = PriorityQueue()  
 visited\_node = {}  
 parent\_child\_map = {}  
 direction\_list = []  
 path\_cost = 0  
 heuristic\_value = 0  
  
 start\_node = problem.getStartState()  
 parent\_child\_map[start\_node] = []  
 loc\_pqueue.push(start\_node, heuristic\_value)  
  
 **def** traverse\_path(parent\_node):  
 temp = 0  
 **while True**:  
 **'''print parent\_node'''** map\_row = parent\_child\_map[parent\_node]  
 **if** (len(map\_row) == 4):  
 parent\_node = map\_row[0]  
 direction = map\_row[1]  
 gvalue = map\_row[2]  
 fvalue = map\_row[3]  
 direction\_list.append(direction)  
 **'''print "Gvalue = %d" % gvalue  
 print fvalue'''  
 '''print "Hueristic = %d" % (fvalue-gvalue)'''  
 '''print "Admissible H = %d" % temp'''** temp = temp + 1  
 **else**:  
 **break  
 return** direction\_list  
  
 **while** (loc\_pqueue.isEmpty() == **False**):  
  
 parent\_node = loc\_pqueue.pop()  
  
 **if** (parent\_node != problem.getStartState()):  
 path\_cost = parent\_child\_map[parent\_node][2]  
  
 **if** (problem.isGoalState(parent\_node)):  
 pathlist = traverse\_path(parent\_node)  
 pathlist.reverse()  
 **return** pathlist  
  
 **elif** (visited\_node.has\_key(parent\_node) == **False**):  
 visited\_node[parent\_node] = []  
 sucessor\_list = problem.getSuccessors(parent\_node)  
 no\_of\_child = len(sucessor\_list)  
 **if** (no\_of\_child > 0):  
 temp = 0  
 **while** (temp < no\_of\_child):  
 child\_nodes = sucessor\_list[temp]  
 child\_state = child\_nodes[0];  
 child\_action = child\_nodes[1];  
 child\_cost = child\_nodes[2];  
  
 heuristic\_value = heuristic(child\_state, problem)  
 gvalue = path\_cost + child\_cost  
 fvalue = gvalue + heuristic\_value  
  
 **if** (visited\_node.has\_key(child\_state) == **False**):  
 loc\_pqueue.push(child\_state, fvalue)  
 **if** (parent\_child\_map.has\_key(child\_state) == **False**):  
 parent\_child\_map[child\_state] = [parent\_node, child\_action, gvalue, fvalue]  
 **else**:  
 **if** (child\_state != start\_node):  
 stored\_fvalue = parent\_child\_map[child\_state][3]  
 **if** (stored\_fvalue > fvalue):  
 parent\_child\_map[child\_state] = [parent\_node, child\_action, gvalue, fvalue]  
 temp = temp + 1  
  
  
*#Kısıtlamalar*bfs = breadthFirstSearch  
dfs = depthFirstSearch  
astar = aStarSearch  
ucs = uniformCostSearch

searcAgent.py:

**from** game **import** Directions  
**from** game **import** Agent  
**from** game **import** Actions  
**from** util **import** manhattanDistance  
**from** search **import** breadthFirstSearch  
**import** time  
**import** search  
  
**class** GoWestAgent(Agent):  
  
 **def** getAction(self, state):  
 *"Aracı bir GameState (pacman.py dosyasında tanımlı) alıyor."* **if** Directions.WEST **in** state.getLegalPacmanActions():  
 **return** Directions.WEST  
 **else**:  
 **return** Directions.STOP  
  
**class** SearchAgent(Agent):  
 *"""  
 Varsayılan olarak, bu aracı bulmak için DFS'yi bir PositionSearchProblem üzerinde çalıştırır.  
 Fn için seçenekler şunları içerir:  
 depthFirstSearch veya dfs  
 breadthFirstSearch veya bfs  
 Not: SearchAgent'ta hiçbir kodu DEĞİŞTİRMEMELİDİR  
 """* **def** \_\_init\_\_(self, fn=**'depthFirstSearch'**, prob=**'PositionSearchProblem'**, heuristic=**'nullHeuristic'**):  
  
 *# Uyarı: Doğru fonksiyonları ve problemleri bulmak için bazı gelişmiş Python sihirleri kullanılır.* **if** fn **not in** dir(search):  
 **raise** AttributeError, fn + **'search.pyde bir arama işlevi değil.'** func = getattr(search, fn)  
 **if 'heuristic' not in** func.func\_code.co\_varnames:  
 print(**'[SearchAgent] işlevi kullanmak'** + fn)  
 self.searchFunction = func  
 **else**:  
 **if** heuristic **in** globals().keys():  
 heur = globals()[heuristic]  
 **elif** heuristic **in** dir(search):  
 heur = getattr(search, heuristic)  
 **else**:  
 **raise** AttributeError, heuristic + **' searchAgents.py veya search.pyde bir işlev değil.'** print(**'[SearchAgent] using function %s and heuristic %s'** % (fn, heuristic))  
  
 *# Not: Bu Python hilesi biraz arama algoritması ve sezgisel birleştirir* self.searchFunction = **lambda** x: func(x, heuristic=heur)  
 **if** prob **not in** globals().keys() **or not** prob.endswith(**'Problem'**):  
 **raise** AttributeError, prob + **' SearchAgents.py de bir arama sorunu türü değil.'** self.searchType = globals()[prob]  
 print(**'[SearchAgent] problem türünü kullanma'** + prob)  
  
 **def** registerInitialState(self, state):  
 *"""  
 Burada hedefe giden yolu seçiyoruz. Bu aşamada, ajan  
 Hedefe giden yolu hesaplamalı ve yerel bir değişkende saklamalıdır.  
 """* **if** self.searchFunction == **None**: **raise** Exception, **'SearchAgent için hiçbir arama işlevi sağlanmadı'** starttime = time.time()  
 problem = self.searchType(state) *# Makes a new search problem* self.actions = self.searchFunction(problem) *# Find a path* totalCost = problem.getCostOfActions(self.actions)  
 print(**'Yol toplam maliyeti ile bulundu %d in %.1f seconds'** % (totalCost, time.time() - starttime))  
 **if '\_expanded' in** dir(problem): print(**'Arama düğümleri genişletildi: %d'** % problem.\_expanded)  
  
 **def** getAction(self, state):  
  
 **if 'actionIndex' not in** dir(self): self.actionIndex = 0  
 i = self.actionIndex  
 self.actionIndex += 1  
 **if** i < len(self.actions):  
 **return** self.actions[i]  
 **else**:  
 **return** Directions.STOP  
  
  
**class** PositionSearchProblem(search.SearchProblem):  
  
  
 **def** \_\_init\_\_(self, gameState, costFn=**lambda** x: 1, goal=(1, 1), start=**None**, warn=**True**, visualize=**True**):  
 *"""  
 Başlangıç ​​ve hedefi depolar.  
 gameState: Bir GameState nesnesi  
 costFn: Bir arama durumundan (tuple) negatif olmayan bir sayıya bir işlev  
 goal: gameState'deki bir konum  
 """* self.walls = gameState.getWalls()  
 self.startState = gameState.getPacmanPosition()  
 **if** start != **None**: self.startState = start  
 self.goal = goal  
 self.costFn = costFn  
 self.visualize = visualize  
 **if** warn **and** (gameState.getNumFood() != 1 **or not** gameState.hasFood(\*goal)):  
 print  
 **'Warning: this does not look like a regular search maze'** *# Teşhir amaçlı* self.\_visited, self.\_visitedlist, self.\_expanded = {}, [], 0 *# DO NOT CHANGE* **def** getStartState(self):  
 **return** self.startState  
  
 **def** isGoalState(self, state):  
 isGoal = state == self.goal  
  
 *# Sadece görüntüleme amaçlı* **if** isGoal **and** self.visualize:  
 self.\_visitedlist.append(state)  
 **import** \_\_main\_\_  
 **if '\_display' in** dir(\_\_main\_\_):  
 **if 'drawExpandedCells' in** dir(\_\_main\_\_.\_display): *# @UndefinedVariable* \_\_main\_\_.\_display.drawExpandedCells(self.\_visitedlist) *# @UndefinedVariable* **return** isGoal  
  
 **def** getSuccessors(self, state):  
  
  
 successors = []  
 **for** action **in** [Directions.NORTH, Directions.SOUTH, Directions.EAST, Directions.WEST]:  
 x, y = state  
 dx, dy = Actions.directionToVector(action)  
 nextx, nexty = int(x + dx), int(y + dy)  
 **if not** self.walls[nextx][nexty]:  
 nextState = (nextx, nexty)  
 cost = self.costFn(nextState)  
 successors.append((nextState, action, cost))  
  
 *# Görüntüleme amaçlı defter tutma* self.\_expanded += 1 *# DO NOT CHANGE* **if** state **not in** self.\_visited:  
 self.\_visited[state] = **True** self.\_visitedlist.append(state)  
  
 **return** successors  
  
 **def** getCostOfActions(self, actions):  
  
 **if** actions == **None**: **return** 999999  
 x, y = self.getStartState()  
 cost = 0  
 **for** action **in** actions:  
 *# Check figure out the next state and see whether its' legal* dx, dy = Actions.directionToVector(action)  
 x, y = int(x + dx), int(y + dy)  
 **if** self.walls[x][y]: **return** 999999  
 cost += self.costFn((x, y))  
 **return** cost  
  
  
**class** StayEastSearchAgent(SearchAgent):  
  
 **def** \_\_init\_\_(self):  
 self.searchFunction = search.uniformCostSearch  
 costFn = **lambda** pos: .5 \*\* pos[0]  
 self.searchType = **lambda** state: PositionSearchProblem(state, costFn, (1, 1), **None**, **False**)  
  
  
**class** StayWestSearchAgent(SearchAgent):  
  
  
 **def** \_\_init\_\_(self):  
 self.searchFunction = search.uniformCostSearch  
 costFn = **lambda** pos: 2 \*\* pos[0]  
 self.searchType = **lambda** state: PositionSearchProblem(state, costFn)  
  
  
**def** manhattanHeuristic(position, problem, info={}):  
 *"""manhattan uzaklığı"""* xy1 = position  
 xy2 = problem.goal  
 **return** abs(xy1[0] - xy2[0]) + abs(xy1[1] - xy2[1])  
  
  
**def** euclideanHeuristic(position, problem, info={}):  
  
 xy1 = position  
 xy2 = problem.goal  
 **return** ((xy1[0] - xy2[0]) \*\* 2 + (xy1[1] - xy2[1]) \*\* 2) \*\* 0.5  
  
**class** CornersProblem(search.SearchProblem):  
  
 **def** \_\_init\_\_(self, startingGameState):  
  
 self.walls = startingGameState.getWalls()  
 self.startingPosition = startingGameState.getPacmanPosition()  
 top, right = self.walls.height - 2, self.walls.width - 2  
 self.corners = ((1, 1), (1, top), (right, 1), (right, top))  
 **for** corner **in** self.corners:  
 **if not** startingGameState.hasFood(\*corner):  
 print  
 **'Uyarı: köşede yiyecek yok '** + str(corner)  
 self.\_expanded = 0 *# DEĞİŞTİRME; Genişletilmiş genişletilmiş düğüm sayısı* **"\*\*\* Kod burada \*\*\*"** self.Corner1 = **False** self.Corner2 = **False** self.Corner3 = **False** self.Corner4 = **False  
 '''self.corner\_visited = 0'''  
  
 def** getStartState(self):  
 **return** self.startingPosition  
  
 **def** isGoalState(self, state):  
 *"""Bu arama durumunun sorunun bir amaç durumu olup olmadığını döndürür. """* **if not** (state == self.startingPosition):  
 corner\_visited = state[1]  
 **if** (corner\_visited == (**True**, **True**, **True**, **True**)):  
 **return** state  
  
 **def** getSuccessors(self, state):  
  
 successors = []  
  
 **for** action **in** [Directions.NORTH, Directions.SOUTH, Directions.EAST, Directions.WEST]:  
  
 **if not** (state == self.startingPosition):  
 postition\_state = state[0]  
 self.Corner1 = state[1][0]  
 self.Corner2 = state[1][1]  
 self.Corner3 = state[1][2]  
 self.Corner4 = state[1][3]  
 **else**:  
 postition\_state = state  
 self.Corner1 = **False** self.Corner2 = **False** self.Corner3 = **False** self.Corner4 = **False** x, y = postition\_state  
 dx, dy = Actions.directionToVector(action)  
 nextx, nexty = int(x + dx), int(y + dy)  
 **if not** self.walls[nextx][nexty]:  
 cost = 1  
 new\_position = (nextx, nexty)  
 **if** (new\_position == self.corners[0]):  
 self.Corner1 = **True  
 if** (new\_position == self.corners[1]):  
 self.Corner2 = **True  
 if** (new\_position == self.corners[2]):  
 self.Corner3 = **True  
 if** (new\_position == self.corners[3]):  
 self.Corner4 = **True** nextState = (new\_position, (self.Corner1, self.Corner2, self.Corner3, self.Corner4))  
 successors.append((nextState, action, cost))  
  
 self.\_expanded += 1  
 **return** successors  
  
 **def** getCostOfActions(self, actions):  
 *"""maliyetini döndürür"""* **if** actions == **None**: **return** 999999  
 x, y = self.startingPosition  
 **for** action **in** actions:  
 dx, dy = Actions.directionToVector(action)  
 x, y = int(x + dx), int(y + dy)  
 **if** self.walls[x][y]: **return** 999999  
 **return** len(actions)  
  
  
**def** cornersHeuristic(state, problem):  
 *"""  
 Tanımladığınız CornersProblem için bir sezgisel.  
 Bu işlev her zaman alt sınırı olan bir sayıyı döndürmelidir. sorunun en kısa yolu olmalı(aynı zamanda tutarlı).  
 """* corners = problem.corners *# Bunlar köşe koordinatları* walls = problem.walls *# labirent duvarlarıdır* corner\_unvisited = []  
 total\_distance = 0  
  
 **if not** (state == problem.startingPosition):  
 postition\_state = state[0]  
 **if** (state[1][0] == **False**):  
 corner\_unvisited.append(problem.corners[0])  
 **if** (state[1][1] == **False**):  
 corner\_unvisited.append(problem.corners[1])  
 **if** (state[1][2] == **False**):  
 corner\_unvisited.append(problem.corners[2])  
 **if** (state[1][3] == **False**):  
 corner\_unvisited.append(problem.corners[3])  
 **else**:  
 postition\_state = state  
 corner\_unvisited = list(problem.corners)  
  
 **if** (len(corner\_unvisited) == 0):  
 **return** 0  
  
 **while** (len(corner\_unvisited) > 0):  
 distance\_to\_all\_corners = []  
 **for** selected\_corner **in** corner\_unvisited:  
 xy1 = postition\_state  
 xy2 = selected\_corner  
 distance\_to\_corner = abs(xy1[0] - xy2[0]) + abs(xy1[1] - xy2[1])  
 distance\_to\_all\_corners.append((distance\_to\_corner, selected\_corner))  
  
 closest\_corner = min(distance\_to\_all\_corners)  
 corner\_unvisited.remove(closest\_corner[1])  
 postition\_state = closest\_corner[1]  
 total\_distance = total\_distance + closest\_corner[0]  
  
 **return** total\_distance *# Default to trivial solution***class** AStarCornersAgent(SearchAgent):  
  
  
 **def** \_\_init\_\_(self):  
 self.searchFunction = **lambda** prob: search.aStarSearch(prob, cornersHeuristic)  
 self.searchType = CornersProblem  
  
  
**class** FoodSearchProblem:  
 *"""  
 foodGrid: Kalan yiyeceği belirten Doğru veya Yanlış olan bir Kılavuz  
 """* **def** \_\_init\_\_(self, startingGameState):  
 self.start = (startingGameState.getPacmanPosition(), startingGameState.getFood())  
 self.walls = startingGameState.getWalls()  
 self.startingGameState = startingGameState  
 self.\_expanded = 0 *# DO NOT CHANGE* self.heuristicInfo = {} *# A dictionary for the heuristic to store information* **def** getStartState(self):  
 **return** self.start  
  
 **def** isGoalState(self, state):  
 **return** state[1].count() == 0  
  
 **def** getSuccessors(self, state):  
 successors = []  
 self.\_expanded += 1 *# DO NOT CHANGE* **for** direction **in** [Directions.NORTH, Directions.SOUTH, Directions.EAST, Directions.WEST]:  
 x, y = state[0]  
 dx, dy = Actions.directionToVector(direction)  
 nextx, nexty = int(x + dx), int(y + dy)  
 **if not** self.walls[nextx][nexty]:  
 nextFood = state[1].copy()  
 nextFood[nextx][nexty] = **False** successors.append((((nextx, nexty), nextFood), direction, 1))  
 **return** successors  
  
 **def** getCostOfActions(self, actions):  
 *""" Belirli bir eylem dizisinin maliyetini döndürür."""* x, y = self.getStartState()[0]  
 cost = 0  
 **for** action **in** actions:  
 *# figure out the next state and see whether it's legal* dx, dy = Actions.directionToVector(action)  
 x, y = int(x + dx), int(y + dy)  
 **if** self.walls[x][y]:  
 **return** 999999  
 cost += 1  
 **return** cost  
  
  
**class** AStarFoodSearchAgent(SearchAgent):  
  
  
 **def** \_\_init\_\_(self):  
 self.searchFunction = **lambda** prob: search.aStarSearch(prob, foodHeuristic)  
 self.searchType = FoodSearchProblem  
  
  
**def** foodHeuristic(state, problem):  
 *"""  
 A \* kullanılıyorsa, daha kötü bir maliyet araştırması buluntusu olan bir çözüm bulur.  
 """* position, foodGrid = state  
 hvalue = 0  
 food\_available = []  
 total\_distance = 0  
 **for** i **in** range(0, foodGrid.width):  
 **for** j **in** range(0, foodGrid.height):  
 **if** (foodGrid[i][j] == **True**):  
 food\_location = (i, j)  
 food\_available.append(food\_location)  
  
 **if** (len(food\_available) == 0):  
 **return** 0  
  
 max\_distance = ((0, 0), (0, 0), 0)  
  
 **for** current\_food **in** food\_available:  
 **for** select\_food **in** food\_available:  
 **if** (current\_food == select\_food):  
 **pass  
 else**:  
 distance = manhattanDistance(current\_food, select\_food)  
 **if** (max\_distance[2] < distance):  
 max\_distance = (current\_food, select\_food, distance)  
  
 **if** (max\_distance[0] == (0, 0) **and** max\_distance[1] == (0, 0)):  
 hvalue = manhattanDistance(position, food\_available[0])  
 **else**:  
 d1 = manhattanDistance(position, max\_distance[0])  
 d2 = manhattanDistance(position, max\_distance[1])  
 hvalue = max\_distance[2] + min(d1, d2)  
  
 **return** hvalue  
  
  
**class** ClosestDotSearchAgent(SearchAgent):  
 *"""Bir dizi arama kullanarak tüm yiyecekleri ara"""* **def** registerInitialState(self, state):  
 self.actions = []  
 currentState = state  
 **while** (currentState.getFood().count() > 0):  
 nextPathSegment = self.findPathToClosestDot(currentState) *# The missing piece* self.actions += nextPathSegment  
 **for** action **in** nextPathSegment:  
 legal = currentState.getLegalActions()  
 **if** action **not in** legal:  
 t = (str(action), str(currentState))  
 **raise** Exception, **'findPathToClosestDot returned an illegal move: %s!\n%s'** % t  
 currentState = currentState.generateSuccessor(0, action)  
 self.actionIndex = 0  
 print  
 **'Path found with cost %d.'** % len(self.actions)  
  
 **def** findPathToClosestDot(self, gameState):  
 *"""  
 En yakın noktaya giden bir yolu döndürür.  
 """* startPosition = gameState.getPacmanPosition()  
 food = gameState.getFood()  
 walls = gameState.getWalls()  
 problem = AnyFoodSearchProblem(gameState)  
  
 action\_list = breadthFirstSearch(problem)  
  
 **return** action\_list  
  
  
**class** AnyFoodSearchProblem(PositionSearchProblem):  
 *"""Herhangi bir yemek için bir yol bulmak için bir arama problemi."""* **def** \_\_init\_\_(self, gameState):  
 *"Bilgileri gameState'den depolar. Bunu değiştirmenize gerek yoktur."  
 # Daha sonra başvurmak için yiyecekleri saklayın* self.food = gameState.getFood()  
 self.walls = gameState.getWalls()  
 self.startState = gameState.getPacmanPosition()  
 self.costFn = **lambda** x: 1  
 self.\_visited, self.\_visitedlist, self.\_expanded = {}, [], 0 *# DO NOT CHANGE* **def** isGoalState(self, state):  
  
 x, y = state  
 foodGrid = self.food  
 **if** (foodGrid[x][y] == **True**) **or** (foodGrid.count() == 0):  
 **return True  
  
  
def** mazeDistance(point1, point2, gameState):  
 *"""  
 Arama fonksiyonlarını kullanarak herhangi iki nokta arasındaki labirent mesafesini döndürür.  
 """* x1, y1 = point1  
 x2, y2 = point2  
 walls = gameState.getWalls()  
 **assert not** walls[x1][y1], **'point1 is a wall: '** + str(point1)  
 **assert not** walls[x2][y2], **'point2 is a wall: '** + str(point2)  
 prob = PositionSearchProblem(gameState, start=point1, goal=point2, warn=**False**, visualize=**False**)  
 **return** len(search.bfs(prob))

KeyboardAgent.py:

**from** game **import** Agent  
**from** game **import** Directions  
**import** random  
  
  
**class** KeyboardAgent(Agent):  
 WEST\_KEY = **'a'** EAST\_KEY = **'d'** NORTH\_KEY = **'w'** SOUTH\_KEY = **'s'** STOP\_KEY = **'q'  
  
 def** \_\_init\_\_(self, index=0):  
  
 self.lastMove = Directions.STOP  
 self.index = index  
 self.keys = []  
  
 **def** getAction(self, state):  
 **from** graphicsUtils **import** keys\_waiting  
 **from** graphicsUtils **import** keys\_pressed  
 keys = keys\_waiting() + keys\_pressed()  
 **if** keys != []:  
 self.keys = keys  
  
 legal = state.getLegalActions(self.index)  
 move = self.getMove(legal)  
  
 **if** move == Directions.STOP:  
 **if** self.lastMove **in** legal:  
 move = self.lastMove  
  
 **if** (self.STOP\_KEY **in** self.keys) **and** Directions.STOP **in** legal:  
 move = Directions.STOP  
  
 **if** move **not in** legal:  
 move = random.choice(legal)  
  
 self.lastMove = move  
 **return** move  
  
 **def** getMove(self, legal):  
 move = Directions.STOP  
 **if** (self.WEST\_KEY **in** self.keys **or 'Left' in** self.keys) **and** Directions.WEST **in** legal:  
 move = Directions.WEST  
 **if** (self.EAST\_KEY **in** self.keys **or 'Right' in** self.keys) **and** Directions.EAST **in** legal:  
 move = Directions.EAST  
 **if** (self.NORTH\_KEY **in** self.keys **or 'Up' in** self.keys) **and** Directions.NORTH **in** legal:  
 move = Directions.NORTH  
 **if** (self.SOUTH\_KEY **in** self.keys **or 'Down' in** self.keys) **and** Directions.SOUTH **in** legal:  
 move = Directions.SOUTH  
 **return** move  
  
  
**class** KeyboardAgent2(KeyboardAgent):  
  
 WEST\_KEY = **'j'** EAST\_KEY = **"l"** NORTH\_KEY = **'i'** SOUTH\_KEY = **'k'** STOP\_KEY = **'u'  
  
 def** getMove(self, legal):  
 move = Directions.STOP  
 **if** (self.WEST\_KEY **in** self.keys) **and** Directions.WEST **in** legal:  
 move = Directions.WEST  
 **if** (self.EAST\_KEY **in** self.keys) **and** Directions.EAST **in** legal:  
 move = Directions.EAST  
 **if** (self.NORTH\_KEY **in** self.keys) **and** Directions.NORTH **in** legal:  
 move = Directions.NORTH  
 **if** (self.SOUTH\_KEY **in** self.keys) **and** Directions.SOUTH **in** legal:  
 move = Directions.SOUTH  
 **return** move

graphicsDisplay.py;

**import** math  
**import** time  
  
  
  
DEFAULT\_GRID\_SIZE = 30.0  
INFO\_PANE\_HEIGHT = 35  
BACKGROUND\_COLOR = formatColor(0, 0, 0)  
WALL\_COLOR = formatColor(0.0 / 255.0, 51.0 / 255.0, 255.0 / 255.0)  
INFO\_PANE\_COLOR = formatColor(.4, .4, 0)  
SCORE\_COLOR = formatColor(.9, .9, .9)  
PACMAN\_OUTLINE\_WIDTH = 2  
PACMAN\_CAPTURE\_OUTLINE\_WIDTH = 4  
  
GHOST\_COLORS = []  
GHOST\_COLORS.append(formatColor(.9, 0, 0)) *# Red*GHOST\_COLORS.append(formatColor(0, .3, .9)) *# Blue*GHOST\_COLORS.append(formatColor(.98, .41, .07)) *# Orange*GHOST\_COLORS.append(formatColor(.1, .75, .7)) *# Green*GHOST\_COLORS.append(formatColor(1.0, 0.6, 0.0)) *# Yellow*GHOST\_COLORS.append(formatColor(.4, 0.13, 0.91)) *# Purple*TEAM\_COLORS = GHOST\_COLORS[:2]  
  
GHOST\_SHAPE = [  
 (0, 0.3),  
 (0.25, 0.75),  
 (0.5, 0.3),  
 (0.75, 0.75),  
 (0.75, -0.5),  
 (0.5, -0.75),  
 (-0.5, -0.75),  
 (-0.75, -0.5),  
 (-0.75, 0.75),  
 (-0.5, 0.3),  
 (-0.25, 0.75)  
]  
GHOST\_SIZE = 0.65  
SCARED\_COLOR = formatColor(1, 1, 1)  
  
GHOST\_VEC\_COLORS = list(map(colorToVector, GHOST\_COLORS))  
  
PACMAN\_COLOR = formatColor(255.0 / 255.0, 255.0 / 255.0, 61.0 / 255)  
PACMAN\_SCALE = 0.5  
  
FOOD\_COLOR = formatColor(1, 1, 1)  
FOOD\_SIZE = 0.1  
  
LASER\_COLOR = formatColor(1, 0, 0)  
LASER\_SIZE = 0.02  
  
CAPSULE\_COLOR = formatColor(1, 1, 1)  
CAPSULE\_SIZE = 0.25  
  
WALL\_RADIUS = 0.15  
  
  
**class** InfoPane:  
  
 **def** \_\_init\_\_(self, layout, gridSize):  
 self.gridSize = gridSize  
 self.width = (layout.width) \* gridSize  
 self.base = (layout.height + 1) \* gridSize  
 self.height = INFO\_PANE\_HEIGHT  
 self.fontSize = 24  
 self.textColor = PACMAN\_COLOR  
 self.drawPane()  
  
 **def** toScreen(self, pos, y=**None**):  
 **if** y == **None**:  
 x, y = pos  
 **else**:  
 x = pos  
  
 x = self.gridSize + x  
 y = self.base + y  
 **return** x, y  
  
 **def** drawPane(self):  
 self.scoreText = text(self.toScreen(  
 0, 0), self.textColor, **"SCORE: 0"**, **"Times"**, self.fontSize, **"bold"**)  
  
 **def** initializeGhostDistances(self, distances):  
 self.ghostDistanceText = []  
  
 size = 20  
 **if** self.width < 240:  
 size = 12  
 **if** self.width < 160:  
 size = 10  
  
 **for** i, d **in** enumerate(distances):  
 t = text(self.toScreen(self.width / 2 + self.width / 8 \* i,  
 0), GHOST\_COLORS[i + 1], d, **"Times"**, size, **"bold"**)  
 self.ghostDistanceText.append(t)  
  
 **def** updateScore(self, score):  
 changeText(self.scoreText, **"SCORE: % 4d"** % score)  
  
 **def** setTeam(self, isBlue):  
 text = **"RED TEAM"  
 if** isBlue:  
 text = **"BLUE TEAM"** self.teamText = text(self.toScreen(  
 300, 0), self.textColor, text, **"Times"**, self.fontSize, **"bold"**)  
  
 **def** updateGhostDistances(self, distances):  
 **if** len(distances) == 0:  
 **return  
 if 'ghostDistanceText' not in** dir(self):  
 self.initializeGhostDistances(distances)  
 **else**:  
 **for** i, d **in** enumerate(distances):  
 changeText(self.ghostDistanceText[i], d)  
  
 **def** drawGhost(self):  
 **pass  
  
 def** drawPacman(self):  
 **pass  
  
 def** drawWarning(self):  
 **pass  
  
 def** clearIcon(self):  
 **pass  
  
 def** updateMessage(self, message):  
 **pass  
  
 def** clearMessage(self):  
 **pass  
  
  
class** PacmanGraphics:  
  
 **def** \_\_init\_\_(self, zoom=1.0, frameTime=0.0, capture=**False**):  
 self.have\_window = 0  
 self.currentGhostImages = {}  
 self.pacmanImage = **None** self.zoom = zoom  
 self.gridSize = DEFAULT\_GRID\_SIZE \* zoom  
 self.capture = capture  
 self.frameTime = frameTime  
  
 **def** checkNullDisplay(self):  
 **return False  
  
 def** initialize(self, state, isBlue=**False**):  
 self.isBlue = isBlue  
 self.startGraphics(state)  
 self.distributionImages = **None** self.drawStaticObjects(state)  
 self.drawAgentObjects(state)  
  
 *# Information* self.previousState = state  
  
 **def** startGraphics(self, state):  
 self.layout = state.layout  
 layout = self.layout  
 self.width = layout.width  
 self.height = layout.height  
 self.make\_window(self.width, self.height)  
 self.infoPane = InfoPane(layout, self.gridSize)  
 self.currentState = layout  
  
 **def** drawDistributions(self, state):  
 walls = state.layout.walls  
 dist = []  
 **for** x **in** range(walls.width):  
 distx = []  
 dist.append(distx)  
 **for** y **in** range(walls.height):  
 (screen\_x, screen\_y) = self.to\_screen((x, y))  
 block = square((screen\_x, screen\_y),  
 0.5 \* self.gridSize,  
 color=BACKGROUND\_COLOR,  
 filled=1, behind=2)  
 distx.append(block)  
 self.distributionImages = dist  
  
 **def** drawStaticObjects(self, state):  
 layout = self.layout  
 self.drawWalls(layout.walls)  
 self.food = self.drawFood(layout.food)  
 self.capsules = self.drawCapsules(layout.capsules)  
 refresh()  
  
 **def** drawAgentObjects(self, state):  
 self.agentImages = [] *# (agentState, image)* **for** index, agent **in** enumerate(state.agentStates):  
 **if** agent.isPacman:  
 image = self.drawPacman(agent, index)  
 self.agentImages.append((agent, image))  
 **else**:  
 image = self.drawGhost(agent, index)  
 self.agentImages.append((agent, image))  
 refresh()  
  
 **def** swapImages(self, agentIndex, newState):  
 prevState, prevImage = self.agentImages[agentIndex]  
 **for** item **in** prevImage:  
 remove\_from\_screen(item)  
 **if** newState.isPacman:  
 image = self.drawPacman(newState, agentIndex)  
 self.agentImages[agentIndex] = (newState, image)  
 **else**:  
 image = self.drawGhost(newState, agentIndex)  
 self.agentImages[agentIndex] = (newState, image)  
 refresh()  
  
 **def** update(self, newState):  
 agentIndex = newState.\_agentMoved  
 agentState = newState.agentStates[agentIndex]  
  
 **if** self.agentImages[agentIndex][0].isPacman != agentState.isPacman:  
 self.swapImages(agentIndex, agentState)  
 prevState, prevImage = self.agentImages[agentIndex]  
 **if** agentState.isPacman:  
 self.animatePacman(agentState, prevState, prevImage)  
 **else**:  
 self.moveGhost(agentState, agentIndex, prevState, prevImage)  
 self.agentImages[agentIndex] = (agentState, prevImage)  
  
 **if** newState.\_foodEaten != **None**:  
 self.removeFood(newState.\_foodEaten, self.food)  
 **if** newState.\_capsuleEaten != **None**:  
 self.removeCapsule(newState.\_capsuleEaten, self.capsules)  
 self.infoPane.updateScore(newState.score)  
 **if 'ghostDistances' in** dir(newState):  
 self.infoPane.updateGhostDistances(newState.ghostDistances)  
  
 **def** make\_window(self, width, height):  
 grid\_width = (width - 1) \* self.gridSize  
 grid\_height = (height - 1) \* self.gridSize  
 screen\_width = 2 \* self.gridSize + grid\_width  
 screen\_height = 2 \* self.gridSize + grid\_height + INFO\_PANE\_HEIGHT  
  
 begin\_graphics(screen\_width,  
 screen\_height,  
 BACKGROUND\_COLOR,  
 **"CS188 Pacman"**)  
  
 **def** drawPacman(self, pacman, index):  
 position = self.getPosition(pacman)  
 screen\_point = self.to\_screen(position)  
 endpoints = self.getEndpoints(self.getDirection(pacman))  
  
 width = PACMAN\_OUTLINE\_WIDTH  
 outlineColor = PACMAN\_COLOR  
 fillColor = PACMAN\_COLOR  
  
 **if** self.capture:  
 outlineColor = TEAM\_COLORS[index % 2]  
 fillColor = GHOST\_COLORS[index]  
 width = PACMAN\_CAPTURE\_OUTLINE\_WIDTH  
  
 **return** [circle(screen\_point, PACMAN\_SCALE \* self.gridSize,  
 fillColor=fillColor, outlineColor=outlineColor,  
 endpoints=endpoints,  
 width=width)]  
  
 **def** getEndpoints(self, direction, position=(0, 0)):  
 x, y = position  
 pos = x - int(x) + y - int(y)  
 width = 30 + 80 \* math.sin(math.pi \* pos)  
  
 delta = width / 2  
 **if** (direction == **'West'**):  
 endpoints = (180 + delta, 180 - delta)  
 **elif** (direction == **'North'**):  
 endpoints = (90 + delta, 90 - delta)  
 **elif** (direction == **'South'**):  
 endpoints = (270 + delta, 270 - delta)  
 **else**:  
 endpoints = (0 + delta, 0 - delta)  
 **return** endpoints  
  
 **def** movePacman(self, position, direction, image):  
 screenPosition = self.to\_screen(position)  
 endpoints = self.getEndpoints(direction, position)  
 r = PACMAN\_SCALE \* self.gridSize  
 moveCircle(image[0], screenPosition, r, endpoints)  
 refresh()  
  
 **def** animatePacman(self, pacman, prevPacman, image):  
 **if** self.frameTime < 0:  
 print(**'Press any key to step forward, "q" to play'**)  
 keys = wait\_for\_keys()  
 **if 'q' in** keys:  
 self.frameTime = 0.1  
 **if** self.frameTime > 0.01 **or** self.frameTime < 0:  
 start = time.time()  
 fx, fy = self.getPosition(prevPacman)  
 px, py = self.getPosition(pacman)  
 frames = 4.0  
 **for** i **in** range(1, int(frames) + 1):  
 pos = px \* i / frames + fx \* \  
 (frames - i) / frames, py \* i / \  
 frames + fy \* (frames - i) / frames  
 self.movePacman(pos, self.getDirection(pacman), image)  
 refresh()  
 sleep(abs(self.frameTime) / frames)  
 **else**:  
 self.movePacman(self.getPosition(pacman),  
 self.getDirection(pacman), image)  
 refresh()  
  
 **def** getGhostColor(self, ghost, ghostIndex):  
 **if** ghost.scaredTimer > 0:  
 **return** SCARED\_COLOR  
 **else**:  
 **return** GHOST\_COLORS[ghostIndex]  
  
 **def** drawGhost(self, ghost, agentIndex):  
 pos = self.getPosition(ghost)  
 dir = self.getDirection(ghost)  
 (screen\_x, screen\_y) = (self.to\_screen(pos))  
 coords = []  
 **for** (x, y) **in** GHOST\_SHAPE:  
 coords.append((x \* self.gridSize \* GHOST\_SIZE + screen\_x,  
 y \* self.gridSize \* GHOST\_SIZE + screen\_y))  
  
 colour = self.getGhostColor(ghost, agentIndex)  
 body = polygon(coords, colour, filled=1)  
 WHITE = formatColor(1.0, 1.0, 1.0)  
 BLACK = formatColor(0.0, 0.0, 0.0)  
  
 dx = 0  
 dy = 0  
 **if** dir == **'North'**:  
 dy = -0.2  
 **if** dir == **'South'**:  
 dy = 0.2  
 **if** dir == **'East'**:  
 dx = 0.2  
 **if** dir == **'West'**:  
 dx = -0.2  
 leftEye = circle((screen\_x + self.gridSize \* GHOST\_SIZE \* (-0.3 + dx / 1.5), screen\_y -  
 self.gridSize \* GHOST\_SIZE \* (0.3 - dy / 1.5)), self.gridSize \* GHOST\_SIZE \* 0.2, WHITE, WHITE)  
 rightEye = circle((screen\_x + self.gridSize \* GHOST\_SIZE \* (0.3 + dx / 1.5), screen\_y -  
 self.gridSize \* GHOST\_SIZE \* (0.3 - dy / 1.5)), self.gridSize \* GHOST\_SIZE \* 0.2, WHITE, WHITE)  
 leftPupil = circle((screen\_x + self.gridSize \* GHOST\_SIZE \* (-0.3 + dx), screen\_y -  
 self.gridSize \* GHOST\_SIZE \* (0.3 - dy)), self.gridSize \* GHOST\_SIZE \* 0.08, BLACK, BLACK)  
 rightPupil = circle((screen\_x + self.gridSize \* GHOST\_SIZE \* (0.3 + dx), screen\_y -  
 self.gridSize \* GHOST\_SIZE \* (0.3 - dy)), self.gridSize \* GHOST\_SIZE \* 0.08, BLACK, BLACK)  
 ghostImageParts = []  
 ghostImageParts.append(body)  
 ghostImageParts.append(leftEye)  
 ghostImageParts.append(rightEye)  
 ghostImageParts.append(leftPupil)  
 ghostImageParts.append(rightPupil)  
  
 **return** ghostImageParts  
  
 **def** moveEyes(self, pos, dir, eyes):  
 (screen\_x, screen\_y) = (self.to\_screen(pos))  
 dx = 0  
 dy = 0  
 **if** dir == **'North'**:  
 dy = -0.2  
 **if** dir == **'South'**:  
 dy = 0.2  
 **if** dir == **'East'**:  
 dx = 0.2  
 **if** dir == **'West'**:  
 dx = -0.2  
 moveCircle(eyes[0], (screen\_x + self.gridSize \* GHOST\_SIZE \* (-0.3 + dx / 1.5), screen\_y -  
 self.gridSize \* GHOST\_SIZE \* (0.3 - dy / 1.5)), self.gridSize \* GHOST\_SIZE \* 0.2)  
 moveCircle(eyes[1], (screen\_x + self.gridSize \* GHOST\_SIZE \* (0.3 + dx / 1.5), screen\_y -  
 self.gridSize \* GHOST\_SIZE \* (0.3 - dy / 1.5)), self.gridSize \* GHOST\_SIZE \* 0.2)  
 moveCircle(eyes[2], (screen\_x + self.gridSize \* GHOST\_SIZE \* (-0.3 + dx), screen\_y -  
 self.gridSize \* GHOST\_SIZE \* (0.3 - dy)), self.gridSize \* GHOST\_SIZE \* 0.08)  
 moveCircle(eyes[3], (screen\_x + self.gridSize \* GHOST\_SIZE \* (0.3 + dx), screen\_y -  
 self.gridSize \* GHOST\_SIZE \* (0.3 - dy)), self.gridSize \* GHOST\_SIZE \* 0.08)  
  
 **def** moveGhost(self, ghost, ghostIndex, prevGhost, ghostImageParts):  
 old\_x, old\_y = self.to\_screen(self.getPosition(prevGhost))  
 new\_x, new\_y = self.to\_screen(self.getPosition(ghost))  
 delta = new\_x - old\_x, new\_y - old\_y  
  
 **for** ghostImagePart **in** ghostImageParts:  
 move\_by(ghostImagePart, delta)  
 refresh()  
  
 **if** ghost.scaredTimer > 0:  
 color = SCARED\_COLOR  
 **else**:  
 color = GHOST\_COLORS[ghostIndex]  
 edit(ghostImageParts[0], (**'fill'**, color), (**'outline'**, color))  
 self.moveEyes(self.getPosition(ghost),  
 self.getDirection(ghost), ghostImageParts[-4:])  
 refresh()  
  
 **def** getPosition(self, agentState):  
 **if** agentState.configuration == **None**:  
 **return** (-1000, -1000)  
 **return** agentState.getPosition()  
  
 **def** getDirection(self, agentState):  
 **if** agentState.configuration == **None**:  
 **return** Directions.STOP  
 **return** agentState.configuration.getDirection()  
  
 **def** finish(self):  
 end\_graphics()  
  
 **def** to\_screen(self, point):  
 (x, y) = point  
 x = (x + 1) \* self.gridSize  
 y = (self.height - y) \* self.gridSize  
 **return** (x, y)  
 **def** to\_screen2(self, point):  
 (x, y) = point  
 *#y = self.height - y* x = (x + 1) \* self.gridSize  
 y = (self.height - y) \* self.gridSize  
 **return** (x, y)  
  
 **def** isWall(self, x, y, walls):  
 **if** x < 0 **or** y < 0:  
 **return False  
 if** x >= walls.width **or** y >= walls.height:  
 **return False  
 return** walls[x][y]  
  
 **def** drawFood(self, foodMatrix):  
 foodImages = []  
 color = FOOD\_COLOR  
 **for** xNum, x **in** enumerate(foodMatrix):  
 **if** self.capture **and** (xNum \* 2) <= foodMatrix.width:  
 color = TEAM\_COLORS[0]  
 **if** self.capture **and** (xNum \* 2) > foodMatrix.width:  
 color = TEAM\_COLORS[1]  
 imageRow = []  
 foodImages.append(imageRow)  
 **for** yNum, cell **in** enumerate(x):  
 **if** cell:  
 screen = self.to\_screen((xNum, yNum))  
 dot = circle(screen,  
 FOOD\_SIZE \* self.gridSize,  
 outlineColor=color, fillColor=color,  
 width=1)  
 imageRow.append(dot)  
 **else**:  
 imageRow.append(**None**)  
 **return** foodImages  
  
 **def** drawCapsules(self, capsules):  
 capsuleImages = {}  
 **for** capsule **in** capsules:  
 (screen\_x, screen\_y) = self.to\_screen(capsule)  
 dot = circle((screen\_x, screen\_y),  
 CAPSULE\_SIZE \* self.gridSize,  
 outlineColor=CAPSULE\_COLOR,  
 fillColor=CAPSULE\_COLOR,  
 width=1)  
 capsuleImages[capsule] = dot  
 **return** capsuleImages  
  
 **def** removeFood(self, cell, foodImages):  
 x, y = cell  
 remove\_from\_screen(foodImages[x][y])  
  
 **def** removeCapsule(self, cell, capsuleImages):  
 x, y = cell  
 remove\_from\_screen(capsuleImages[(x, y)])  
  
 **def** drawExpandedCells(self, cells):  
 n = float(len(cells))  
 baseColor = [1.0, 0.0, 0.0]  
 self.clearExpandedCells()  
 self.expandedCells = []  
 **for** k, cell **in** enumerate(cells):  
 screenPos = self.to\_screen(cell)  
 cellColor = formatColor(  
 \*[(n - k) \* c \* .5 / n + .25 **for** c **in** baseColor])  
 block = square(screenPos,  
 0.5 \* self.gridSize,  
 color=cellColor,  
 filled=1, behind=2)  
 self.expandedCells.append(block)  
 **if** self.frameTime < 0:  
 refresh()  
  
 **def** clearExpandedCells(self):  
 **if 'expandedCells' in** dir(self) **and** len(self.expandedCells) > 0:  
 **for** cell **in** self.expandedCells:  
 remove\_from\_screen(cell)  
  
 **def** updateDistributions(self, distributions):  
 distributions = [x.copy() **for** x **in** distributions]  
 **if** self.distributionImages == **None**:  
 self.drawDistributions(self.previousState)  
 **for** x **in** range(len(self.distributionImages)):  
 **for** y **in** range(len(self.distributionImages[0])):  
 image = self.distributionImages[x][y]  
 weights = [dist[(x, y)] **for** dist **in** distributions]  
  
 **if** sum(weights) != 0:  
 **pass** color = [0.0, 0.0, 0.0]  
 colors = GHOST\_VEC\_COLORS[1:]  
 **if** self.capture:  
 colors = GHOST\_VEC\_COLORS  
 **for** weight, gcolor **in** zip(weights, colors):  
 color = [min(1.0, c + 0.95 \* g \* weight \*\* .3)  
 **for** c, g **in** zip(color, gcolor)]  
 changeColor(image, formatColor(\*color))  
 refresh()  
  
  
**class** FirstPersonPacmanGraphics(PacmanGraphics):  
  
 **def** \_\_init\_\_(self, zoom=1.0, showGhosts=**True**, capture=**False**, frameTime=0):  
 PacmanGraphics.\_\_init\_\_(self, zoom, frameTime=frameTime)  
 self.showGhosts = showGhosts  
 self.capture = capture  
  
 **def** initialize(self, state, isBlue=**False**):  
  
 self.isBlue = isBlue  
 PacmanGraphics.startGraphics(self, state)  
 walls = state.layout.walls  
 dist = []  
 self.layout = state.layout  
 self.distributionImages = **None** self.drawStaticObjects(state)  
 self.drawAgentObjects(state)  
 self.previousState = state  
  
 **def** lookAhead(self, config, state):  
 **if** config.getDirection() == **'Stop'**:  
 **return  
 else**:  
 **pass** allGhosts = state.getGhostStates()  
 visibleGhosts = state.getVisibleGhosts()  
 **for** i, ghost **in** enumerate(allGhosts):  
 **if** ghost **in** visibleGhosts:  
 self.drawGhost(ghost, i)  
 **else**:  
 self.currentGhostImages[i] = **None  
  
 def** getGhostColor(self, ghost, ghostIndex):  
 **return** GHOST\_COLORS[ghostIndex]  
  
 **def** getPosition(self, ghostState):  
 **if not** self.showGhosts **and not** ghostState.isPacman **and** ghostState.getPosition()[1] > 1:  
 **return** (-1000, -1000)  
 **else**:  
 **return** PacmanGraphics.getPosition(self, ghostState)  
  
  
**def** add(x, y):  
 **return** (x[0] + y[0], x[1] + y[1])  
SAVE\_POSTSCRIPT = **False**POSTSCRIPT\_OUTPUT\_DIR = **'frames'**FRAME\_NUMBER = 0  
**import** os  
  
  
**def** saveFrame():  
 **global** SAVE\_POSTSCRIPT, FRAME\_NUMBER, POSTSCRIPT\_OUTPUT\_DIR  
 **if not** SAVE\_POSTSCRIPT:  
 **return  
 if not** os.path.exists(POSTSCRIPT\_OUTPUT\_DIR):  
 os.mkdir(POSTSCRIPT\_OUTPUT\_DIR)  
 name = os.path.join(POSTSCRIPT\_OUTPUT\_DIR, **'frame\_%08d.ps'** % FRAME\_NUMBER)  
 FRAME\_NUMBER += 1  
 writePostscript(name)

Kodlar 2.Proje:

*# util.py***import** sys  
**import** inspect  
**import** heapq, random  
**import** cStringIO  
  
  
**class** FixedRandom:  
 **def** \_\_init\_\_(self):  
 fixedState = (3, (2147483648L, 507801126L, 683453281L, 310439348L, 2597246090L, \  
 2209084787L, 2267831527L, 979920060L, 3098657677L, 37650879L, 807947081L, 3974896263L, \  
 881243242L, 3100634921L, 1334775171L, 3965168385L, 746264660L, 4074750168L, 500078808L, \  
 776561771L, 702988163L, 1636311725L, 2559226045L, 157578202L, 2498342920L, 2794591496L, \  
 4130598723L, 496985844L, 2944563015L, 3731321600L, 3514814613L, 3362575829L, 3038768745L, \  
 2206497038L, 1108748846L, 1317460727L, 3134077628L, 988312410L, 1674063516L, 746456451L, \  
 3958482413L, 1857117812L, 708750586L, 1583423339L, 3466495450L, 1536929345L, 1137240525L, \  
 3875025632L, 2466137587L, 1235845595L, 4214575620L, 3792516855L, 657994358L, 1241843248L, \  
 1695651859L, 3678946666L, 1929922113L, 2351044952L, 2317810202L, 2039319015L, 460787996L, \  
 3654096216L, 4068721415L, 1814163703L, 2904112444L, 1386111013L, 574629867L, 2654529343L, \  
 3833135042L, 2725328455L, 552431551L, 4006991378L, 1331562057L, 3710134542L, 303171486L, \  
 1203231078L, 2670768975L, 54570816L, 2679609001L, 578983064L, 1271454725L, 3230871056L, \  
 2496832891L, 2944938195L, 1608828728L, 367886575L, 2544708204L, 103775539L, 1912402393L, \  
 1098482180L, 2738577070L, 3091646463L, 1505274463L, 2079416566L, 659100352L, 839995305L, \  
 1696257633L, 274389836L, 3973303017L, 671127655L, 1061109122L, 517486945L, 1379749962L, \  
 3421383928L, 3116950429L, 2165882425L, 2346928266L, 2892678711L, 2936066049L, 1316407868L, \  
 2873411858L, 4279682888L, 2744351923L, 3290373816L, 1014377279L, 955200944L, 4220990860L, \  
 2386098930L, 1772997650L, 3757346974L, 1621616438L, 2877097197L, 442116595L, 2010480266L, \  
 2867861469L, 2955352695L, 605335967L, 2222936009L, 2067554933L, 4129906358L, 1519608541L, \  
 1195006590L, 1942991038L, 2736562236L, 279162408L, 1415982909L, 4099901426L, 1732201505L, \  
 2934657937L, 860563237L, 2479235483L, 3081651097L, 2244720867L, 3112631622L, 1636991639L, \  
 3860393305L, 2312061927L, 48780114L, 1149090394L, 2643246550L, 1764050647L, 3836789087L, \  
 3474859076L, 4237194338L, 1735191073L, 2150369208L, 92164394L, 756974036L, 2314453957L, \  
 323969533L, 4267621035L, 283649842L, 810004843L, 727855536L, 1757827251L, 3334960421L, \  
 3261035106L, 38417393L, 2660980472L, 1256633965L, 2184045390L, 811213141L, 2857482069L, \  
 2237770878L, 3891003138L, 2787806886L, 2435192790L, 2249324662L, 3507764896L, 995388363L, \  
 856944153L, 619213904L, 3233967826L, 3703465555L, 3286531781L, 3863193356L, 2992340714L, \  
 413696855L, 3865185632L, 1704163171L, 3043634452L, 2225424707L, 2199018022L, 3506117517L, \  
 3311559776L, 3374443561L, 1207829628L, 668793165L, 1822020716L, 2082656160L, 1160606415L, \  
 3034757648L, 741703672L, 3094328738L, 459332691L, 2702383376L, 1610239915L, 4162939394L, \  
 557861574L, 3805706338L, 3832520705L, 1248934879L, 3250424034L, 892335058L, 74323433L, \  
 3209751608L, 3213220797L, 3444035873L, 3743886725L, 1783837251L, 610968664L, 580745246L, \  
 4041979504L, 201684874L, 2673219253L, 1377283008L, 3497299167L, 2344209394L, 2304982920L, \  
 3081403782L, 2599256854L, 3184475235L, 3373055826L, 695186388L, 2423332338L, 222864327L, \  
 1258227992L, 3627871647L, 3487724980L, 4027953808L, 3053320360L, 533627073L, 3026232514L, \  
 2340271949L, 867277230L, 868513116L, 2158535651L, 2487822909L, 3428235761L, 3067196046L, \  
 3435119657L, 1908441839L, 788668797L, 3367703138L, 3317763187L, 908264443L, 2252100381L, \  
 764223334L, 4127108988L, 384641349L, 3377374722L, 1263833251L, 1958694944L, 3847832657L, \  
 1253909612L, 1096494446L, 555725445L, 2277045895L, 3340096504L, 1383318686L, 4234428127L, \  
 1072582179L, 94169494L, 1064509968L, 2681151917L, 2681864920L, 734708852L, 1338914021L, \  
 1270409500L, 1789469116L, 4191988204L, 1716329784L, 2213764829L, 3712538840L, 919910444L, \  
 1318414447L, 3383806712L, 3054941722L, 3378649942L, 1205735655L, 1268136494L, 2214009444L, \  
 2532395133L, 3232230447L, 230294038L, 342599089L, 772808141L, 4096882234L, 3146662953L, \  
 2784264306L, 1860954704L, 2675279609L, 2984212876L, 2466966981L, 2627986059L, 2985545332L, \  
 2578042598L, 1458940786L, 2944243755L, 3959506256L, 1509151382L, 325761900L, 942251521L, \  
 4184289782L, 2756231555L, 3297811774L, 1169708099L, 3280524138L, 3805245319L, 3227360276L, \  
 3199632491L, 2235795585L, 2865407118L, 36763651L, 2441503575L, 3314890374L, 1755526087L, \  
 17915536L, 1196948233L, 949343045L, 3815841867L, 489007833L, 2654997597L, 2834744136L, \  
 417688687L, 2843220846L, 85621843L, 747339336L, 2043645709L, 3520444394L, 1825470818L, \  
 647778910L, 275904777L, 1249389189L, 3640887431L, 4200779599L, 323384601L, 3446088641L, \  
 4049835786L, 1718989062L, 3563787136L, 44099190L, 3281263107L, 22910812L, 1826109246L, \  
 745118154L, 3392171319L, 1571490704L, 354891067L, 815955642L, 1453450421L, 940015623L, \  
 796817754L, 1260148619L, 3898237757L, 176670141L, 1870249326L, 3317738680L, 448918002L, \  
 4059166594L, 2003827551L, 987091377L, 224855998L, 3520570137L, 789522610L, 2604445123L, \  
 454472869L, 475688926L, 2990723466L, 523362238L, 3897608102L, 806637149L, 2642229586L, \  
 2928614432L, 1564415411L, 1691381054L, 3816907227L, 4082581003L, 1895544448L, 3728217394L, \  
 3214813157L, 4054301607L, 1882632454L, 2873728645L, 3694943071L, 1297991732L, 2101682438L, \  
 3952579552L, 678650400L, 1391722293L, 478833748L, 2976468591L, 158586606L, 2576499787L, \  
 662690848L, 3799889765L, 3328894692L, 2474578497L, 2383901391L, 1718193504L, 3003184595L, \  
 3630561213L, 1929441113L, 3848238627L, 1594310094L, 3040359840L, 3051803867L, 2462788790L, \  
 954409915L, 802581771L, 681703307L, 545982392L, 2738993819L, 8025358L, 2827719383L, \  
 770471093L, 3484895980L, 3111306320L, 3900000891L, 2116916652L, 397746721L, 2087689510L, \  
 721433935L, 1396088885L, 2751612384L, 1998988613L, 2135074843L, 2521131298L, 707009172L, \  
 2398321482L, 688041159L, 2264560137L, 482388305L, 207864885L, 3735036991L, 3490348331L, \  
 1963642811L, 3260224305L, 3493564223L, 1939428454L, 1128799656L, 1366012432L, 2858822447L, \  
 1428147157L, 2261125391L, 1611208390L, 1134826333L, 2374102525L, 3833625209L, 2266397263L, \  
 3189115077L, 770080230L, 2674657172L, 4280146640L, 3604531615L, 4235071805L, 3436987249L, \  
 509704467L, 2582695198L, 4256268040L, 3391197562L, 1460642842L, 1617931012L, 457825497L, \  
 1031452907L, 1330422862L, 4125947620L, 2280712485L, 431892090L, 2387410588L, 2061126784L, \  
 896457479L, 3480499461L, 2488196663L, 4021103792L, 1877063114L, 2744470201L, 1046140599L, \  
 2129952955L, 3583049218L, 4217723693L, 2720341743L, 820661843L, 1079873609L, 3360954200L, \  
 3652304997L, 3335838575L, 2178810636L, 1908053374L, 4026721976L, 1793145418L, 476541615L, \  
 973420250L, 515553040L, 919292001L, 2601786155L, 1685119450L, 3030170809L, 1590676150L, \  
 1665099167L, 651151584L, 2077190587L, 957892642L, 646336572L, 2743719258L, 866169074L, \  
 851118829L, 4225766285L, 963748226L, 799549420L, 1955032629L, 799460000L, 2425744063L, \  
 2441291571L, 1928963772L, 528930629L, 2591962884L, 3495142819L, 1896021824L, 901320159L, \  
 3181820243L, 843061941L, 3338628510L, 3782438992L, 9515330L, 1705797226L, 953535929L, \  
 764833876L, 3202464965L, 2970244591L, 519154982L, 3390617541L, 566616744L, 3438031503L, \  
 1853838297L, 170608755L, 1393728434L, 676900116L, 3184965776L, 1843100290L, 78995357L, \  
 2227939888L, 3460264600L, 1745705055L, 1474086965L, 572796246L, 4081303004L, 882828851L, \  
 1295445825L, 137639900L, 3304579600L, 2722437017L, 4093422709L, 273203373L, 2666507854L, \  
 3998836510L, 493829981L, 1623949669L, 3482036755L, 3390023939L, 833233937L, 1639668730L, \  
 1499455075L, 249728260L, 1210694006L, 3836497489L, 1551488720L, 3253074267L, 3388238003L, \  
 2372035079L, 3945715164L, 2029501215L, 3362012634L, 2007375355L, 4074709820L, 631485888L, \  
 3135015769L, 4273087084L, 3648076204L, 2739943601L, 1374020358L, 1760722448L, 3773939706L, \  
 1313027823L, 1895251226L, 4224465911L, 421382535L, 1141067370L, 3660034846L, 3393185650L, \  
 1850995280L, 1451917312L, 3841455409L, 3926840308L, 1397397252L, 2572864479L, 2500171350L, \  
 3119920613L, 531400869L, 1626487579L, 1099320497L, 407414753L, 2438623324L, 99073255L, \  
 3175491512L, 656431560L, 1153671785L, 236307875L, 2824738046L, 2320621382L, 892174056L, \  
 230984053L, 719791226L, 2718891946L, 624L), **None**)  
 self.random = random.Random()  
 self.random.setstate(fixedState)  
  
  
**class** Stack:  
  
 **def** \_\_init\_\_(self):  
 self.list = []  
  
 **def** push(self,item):  
  
 self.list.append(item)  
  
 **def** pop(self):  
  
 **return** self.list.pop()  
  
 **def** isEmpty(self):  
  
 **return** len(self.list) == 0  
  
**class** Queue:  
  
 **def** \_\_init\_\_(self):  
 self.list = []  
  
 **def** push(self,item):  
  
 self.list.insert(0,item)  
  
 **def** pop(self):  
 *"""  
 eski olanı sil  
 """* **return** self.list.pop()  
  
 **def** isEmpty(self):  
  
 **return** len(self.list) == 0  
  
**class** PriorityQueue:  
  
  
 *"""  
 Bir öncelik sırası veri yapısını uygular. Her eklenen öğe  
 onunla ilişkili bir önceliği var ve müşteri genellikle ilgileniyor  
 Sıradaki en düşük öncelikli öğenin hızlı bir şekilde alınması. Bu  
 veri yapısı, O (1) en düşük öncelikli maddeye erişmesini sağlar.  
 """* **def** \_\_init\_\_(self):  
 self.heap = []  
 self.count = 0  
  
 **def** push(self, item, priority):  
 entry = (priority, self.count, item)  
 heapq.heappush(self.heap, entry)  
 self.count += 1  
  
 **def** pop(self):  
 (\_, \_, item) = heapq.heappop(self.heap)  
 **return** item  
  
 **def** isEmpty(self):  
 **return** len(self.heap) == 0  
  
 **def** update(self, item, priority):  
  
 **for** index, (p, c, i) **in** enumerate(self.heap):  
 **if** i == item:  
 **if** p <= priority:  
 **break  
 del** self.heap[index]  
 self.heap.append((priority, c, item))  
 heapq.heapify(self.heap)  
 **break  
 else**:  
 self.push(item, priority)  
  
**class** PriorityQueueWithFunction(PriorityQueue):  
 *"""  
      İçin aynı push / pop imzasıyla bir öncelik sırası uygular  
      Kuyruk ve Yığın sınıfları. Bu, giriş için değiştirme için tasarlanmıştır  
      bu iki sınıf. Arayanın öncelikli bir işlev sağlaması gerekir.  
      Her bir öğenin önceliğini çıkarır.  
      """* **def** \_\_init\_\_(self, priorityFunction):  
  
 self.priorityFunction = priorityFunction  
 PriorityQueue.\_\_init\_\_(self)  
 **def** push(self, item):  
 *"öncelik sırasına göre ekler"* PriorityQueue.push(self, item, self.priorityFunction(item))  
  
  
**def** manhattanDistance( xy1, xy2 ):  
 *" Manhattan algoritması xy1 ve xy2"* **return** abs( xy1[0] - xy2[0] ) + abs( xy1[1] - xy2[1] )  
  
**class** Counter(dict):  
 *"""  
Bir sayaç, bir anahtar seti için sayıları izler.  
  
 Sayaç sınıfı, standart python'un bir uzantısıdır.  
 sözlük türü Sayı değerlerine sahip olmak için uzmanlaşmıştır  
 (tamsayılar veya değişkenler) ve bir avuç ek içerir  
 veri sayma görevini kolaylaştırmak için işlevler. Özellikle,  
 tüm tuşların değeri 0 olur. Varsayılan olarak, bir sözlük kullanarak:  
  
 a = {}  
 ['test'] yazdır  
  
 Counter sınıfı analogu iken hata verir:  
 >>> a = Counter()  
 >>> print a['test']  
 0  
  
 varsayılan 0 değerini döndürür  
  
 >>> a = Counter()  
 >>> a['test'] = 2  
 >>> print a['test']  
 2  
  
  
 Sayaç ayrıca uygulamada kullanışlı ek işlevler içerir  
     Bu görev için sınıflandırıcılar. İki sayaç eklenebilir,  
     birlikte çıkarılmış veya çarpılmış. Detaylar için aşağıya bakınız. Yapabilirler  
     ayrıca normalleştirilebilir ve toplam sayım ve arg max çıkarılabilir.  
 """* **def** \_\_getitem\_\_(self, idx):  
 self.setdefault(idx, 0)  
 **return** dict.\_\_getitem\_\_(self, idx)  
  
 **def** incrementAll(self, keys, count):  
 *"""  
  
 >>> a = Counter()  
 >>> a.incrementAll(['bir','iki', 'üç'], 1)  
 >>> a['bir']  
 1  
 >>> a['iki']  
 1  
 """* **for** key **in** keys:  
 self[key] += count  
  
 **def** argMax(self):  
  
 **if** len(self.keys()) == 0: **return None** all = self.items()  
 values = [x[1] **for** x **in** all]  
 maxIndex = values.index(max(values))  
 **return** all[maxIndex][0]  
  
 **def** sortedKeys(self):  
  
 *"""  
 Tüm tuşlar için sayımların toplamını döndürür.  
 """* **return** sum(self.values())  
  
 **def** normalize(self):  
 *"""  
 Sayacı, hepsinin toplam sayımını sağlayacak şekilde düzenler.  
 tuşlar 1 olarak toplanır. Tüm tuşlar için sayıların oranı  
 aynı kalacak. Boş bir normalleştirmenin  
 Sayaç bir hataya neden olur.  
 """* total = float(self.totalCount())  
 **if** total == 0: **return  
 for** key **in** self.keys():  
 self[key] = self[key] / total  
  
 **def** divideAll(self, divisor):  
 *"""  
 tüm sayıları bölene böler  
 """* divisor = float(divisor)  
 **for** key **in** self:  
 self[key] /= divisor  
  
 **def** copy(self):  
 *"""  
 sayacın bir kopyasını döndürür  
 """* **return** Counter(dict.copy(self))  
  
 **def** \_\_mul\_\_(self, y ):  
 *"""  
 İki sayacı çarpmak, vektörlerinin nokta çarpımını verir.  
         Her benzersiz etiket bir vektör elemanıdır.  
  
 >>> a = Counter()  
 >>> b = Counter()  
 >>> a['birinci'] = -2  
 >>> a['ikinci'] = 4  
 >>> b['üçüncü'] = 3  
 >>> b['ikinci'] = 5  
 >>> a['üçüncü'] = 1.5  
 >>> a['dördüncü'] = 2.5  
 >>> a \* b  
 14  
 """* sum = 0  
 x = self  
 **if** len(x) > len(y):  
 x,y = y,x  
 **for** key **in** x:  
 **if** key **not in** y:  
 **continue** sum += x[key] \* y[key]  
 **return** sum  
  
 **def** \_\_radd\_\_(self, y):  
 *"""  
 Bir sayaca başka bir sayaç eklemek, geçerli sayacı artırır  
         İkinci sayaçta saklanan değerlerle ekler.  
  
 >>> a = Counter()  
 >>> b = Counter()  
 >>> a['birinci'] = -2  
 >>> a['ikinci'] = 4  
 >>> b['üçüncü'] = 3  
 >>> b['dördüncü'] = 1  
 >>> a += b  
 >>> a['birinci']  
 1  
 """* **for** key, value **in** y.items():  
 self[key] += value  
  
 **def** \_\_add\_\_( self, y ):  
 *"""  
 İki sayaç eklemek, tüm anahtarların birliği ile bir sayaç verir ve  
         Ikinci sayıları, ilk sayıları ekledi.  
  
 >>> a = Counter()  
 >>> b = Counter()  
 >>> a['birinci'] = -2  
 >>> a['ikinci'] = 4  
 >>> b['üçüncü'] = 3  
 >>> b['dördüncü'] = 1  
 >>> (a + b)['birinci']  
 1  
 """* addend = Counter()  
 **for** key **in** self:  
 **if** key **in** y:  
 addend[key] = self[key] + y[key]  
 **else**:  
 addend[key] = self[key]  
 **for** key **in** y:  
 **if** key **in** self:  
 **continue** addend[key] = y[key]  
 **return** addend  
  
 **def** \_\_sub\_\_( self, y ):  
 *"""  
 Bir sayacı diğerinden çıkarmak, tüm anahtarların birliği ile bir sayaç verir ve  
         ikincisinin sayıları ilk sayılardan çıkarılır.  
  
 >>> a = Counter()  
 >>> b = Counter()  
 >>> a['birinci'] = -2  
 >>> a['ikinci'] = 4  
 >>> b['üçüncü'] = 3  
 >>> b['dördüncü'] = 1  
 >>> (a - b)['birinci']  
 -  
 """* addend = Counter()  
 **for** key **in** self:  
 **if** key **in** y:  
 addend[key] = self[key] - y[key]  
 **else**:  
 addend[key] = self[key]  
 **for** key **in** y:  
 **if** key **in** self:  
 **continue** addend[key] = -1 \* y[key]  
 **return** addend  
  
**def** raiseNotDefined():  
 fileName = inspect.stack()[1][1]  
 line = inspect.stack()[1][2]  
 method = inspect.stack()[1][3]  
  
 print (**"\*\*\* Method not implemented: %s at line %s of %s"** % (method, line, fileName))  
 sys.exit(1)  
  
**def** normalize(vectorOrCounter):  
 *"""  
      her değeri tüm değerlerin toplamına bölerek bir vektör veya sayacı normalleştirmek.  
 """* normalizedCounter = Counter()  
 **if** type(vectorOrCounter) == type(normalizedCounter):  
 counter = vectorOrCounter  
 total = float(counter.totalCount())  
 **if** total == 0: **return** counter  
 **for** key **in** counter.keys():  
 value = counter[key]  
 normalizedCounter[key] = value / total  
 **return** normalizedCounter  
 **else**:  
 vector = vectorOrCounter  
 s = float(sum(vector))  
 **if** s == 0: **return** vector  
 **return** [el / s **for** el **in** vector]  
  
**def** nSample(distribution, values, n):  
 **if** sum(distribution) != 1:  
 distribution = normalize(distribution)  
 rand = [random.random() **for** i **in** range(n)]  
 rand.sort()  
 samples = []  
 samplePos, distPos, cdf = 0,0, distribution[0]  
 **while** samplePos < n:  
 **if** rand[samplePos] < cdf:  
 samplePos += 1  
 samples.append(values[distPos])  
 **else**:  
 distPos += 1  
 cdf += distribution[distPos]  
 **return** samples  
  
**def** sample(distribution, values = **None**):  
 **if** type(distribution) == Counter:  
 items = sorted(distribution.items())  
 distribution = [i[1] **for** i **in** items]  
 values = [i[0] **for** i **in** items]  
 **if** sum(distribution) != 1:  
 distribution = normalize(distribution)  
 choice = random.random()  
 i, total= 0, distribution[0]  
 **while** choice > total:  
 i += 1  
 total += distribution[i]  
 **return** values[i]  
  
**def** sampleFromCounter(ctr):  
 items = sorted(ctr.items())  
 **return** sample([v **for** k,v **in** items], [k **for** k,v **in** items])  
  
**def** getProbability(value, distribution, values):  
  
 total = 0.0  
 **for** prob, val **in** zip(distribution, values):  
 **if** val == value:  
 total += prob  
 **return** total  
  
**def** flipCoin( p ):  
 r = random.random()  
 **return** r < p  
  
**def** chooseFromDistribution( distribution ):  
 *"Bir sayaç alınır"* **if** type(distribution) == dict **or** type(distribution) == Counter:  
 **return** sample(distribution)  
 r = random.random()  
 base = 0.0  
 **for** prob, element **in** distribution:  
 base += prob  
 **if** r <= base: **return** element  
  
**def** nearestPoint( pos ):  
  
 ( current\_row, current\_col ) = pos  
  
 grid\_row = int( current\_row + 0.5 )  
 grid\_col = int( current\_col + 0.5 )  
 **return** ( grid\_row, grid\_col )  
  
**def** sign( x ):  
 *"""  
 X e göre 1 veya -1 değeri alır  
 """* **if**( x >= 0 ):  
 **return** 1  
 **else**:  
 **return** -1  
  
**def** arrayInvert(array):  
 *"""  
 matrisi ters çevir.  
 """* result = [[] **for** i **in** array]  
 **for** outer **in** array:  
 **for** inner **in** range(len(outer)):  
 result[inner].append(outer[inner])  
 **return** result  
  
**def** matrixAsList( matrix, value = **True** ):  
 *"""  
      Bir matrisi belirtilen değerle eşleşen bir koordinatlar listesine dönüştürür  
 """* rows, cols = len( matrix ), len( matrix[0] )  
 cells = []  
 **for** row **in** range( rows ):  
 **for** col **in** range( cols ):  
 **if** matrix[row][col] == value:  
 cells.append( ( row, col ) )  
 **return** cells  
  
**def** lookup(name, namespace):  
  
 dots = name.count(**'.'**)  
 **if** dots > 0:  
 moduleName, objName = **'.'**.join(name.split(**'.'**)[:-1]), name.split(**'.'**)[-1]  
 module = \_\_import\_\_(moduleName)  
 **return** getattr(module, objName)  
 **else**:  
 modules = [obj **for** obj **in** namespace.values() **if** str(type(obj)) == **"<type 'module'>"**]  
 options = [getattr(module, name) **for** module **in** modules **if** name **in** dir(module)]  
 options += [obj[1] **for** obj **in** namespace.items() **if** obj[0] == name ]  
 **if** len(options) == 1: **return** options[0]  
 **if** len(options) > 1: **raise** Exception, **'Name conflict for %s'  
 raise** Exception, **'%s not found as a method or class'** % name  
  
**def** pause():  
  
 print (**"<Press enter/return to continue>"**)  
 raw\_input()  
  
  
*#zaman aşımlarını çözmek için  
# NOT: TimeoutFuncton yanıt vermiyor. Daha sonra zaman aşımları  
# önceki zaman aşımlarını devre dışı bırak. Global bir liste tutularak çözülebilir  
#aktif zaman aşımı sayısı. Şu anda, test davaları olan sorular arıyor***import** signal  
**import** time  
**class** TimeoutFunctionException(Exception):  
 *"""Zaman aşımına uğraması"""* **pass  
  
  
class** TimeoutFunction:  
 **def** \_\_init\_\_(self, function, timeout):  
 self.timeout = timeout  
 self.function = function  
  
 **def** handle\_timeout(self, signum, frame):  
 **raise** TimeoutFunctionException()  
  
 **def** \_\_call\_\_(self, \*args, \*\*keyArgs):  
 *# SIGALRM sinyalimiz varsa, ve  
 # bu işlev çok uzun çalıştığında. Aksi takdirde, geçen süreyi kontrol edininiz.  
 # yöntem geri döndükten sonra ve sonra bir istisna atayın.* **if** hasattr(signal, **'SIGALRM'**):  
 old = signal.signal(signal.SIGALRM, self.handle\_timeout)  
 signal.alarm(self.timeout)  
 **try**:  
 result = self.function(\*args, \*\*keyArgs)  
 **finally**:  
 signal.signal(signal.SIGALRM, old)  
 signal.alarm(0)  
 **else**:  
 startTime = time.time()  
 result = self.function(\*args, \*\*keyArgs)  
 timeElapsed = time.time() - startTime  
 **if** timeElapsed >= self.timeout:  
 self.handle\_timeout(**None**, **None**)  
 **return** result  
  
  
  
\_ORIGINAL\_STDOUT = **None**\_ORIGINAL\_STDERR = **None**\_MUTED = **False  
  
class** WritableNull:  
 **def** write(self, string):  
 **pass  
  
def** mutePrint():  
 **global** \_ORIGINAL\_STDOUT, \_ORIGINAL\_STDERR, \_MUTED  
 **if** \_MUTED:  
 **return** \_MUTED = **True** \_ORIGINAL\_STDOUT = sys.stdout  
 sys.stdout = WritableNull()  
  
**def** unmutePrint():  
 **global** \_ORIGINAL\_STDOUT, \_ORIGINAL\_STDERR, \_MUTED  
 **if not** \_MUTED:  
 **return** \_MUTED = **False** sys.stdout = \_ORIGINAL\_STDOUT

*# multiAgents.py***from** util **import** manhattanDistance  
**from** game **import** Directions  
**import** random, util  
  
**from** game **import** Agent  
  
  
**class** ReflexAgent(Agent):  
   
  
 **def** getAction(self, gameState):  
   
   
 legalMoves = gameState.getLegalActions()  
  
 scores = [self.evaluationFunction(gameState, action) **for** action **in** legalMoves]  
 bestScore = max(scores)  
 bestIndices = [index **for** index **in** range(len(scores)) **if** scores[index] == bestScore]  
 chosenIndex = random.choice(bestIndices)  
  
 **return** legalMoves[chosenIndex]  
  
 **def** evaluationFunction(self, currentGameState, action):  
  
 successorGameState = currentGameState.generatePacmanSuccessor(action)  
 newPos = successorGameState.getPacmanPosition()  
 newFood = successorGameState.getFood()  
 newGhostStates = successorGameState.getGhostStates()  
 newScaredTimes = [ghostState.scaredTimer **for** ghostState **in** newGhostStates]  
  
  
 heuristic = 0  
 **for** states **in** newScaredTimes:  
 heuristic += states  
 **for** ghost **in** newGhostStates:  
 ghostDist += [manhattanDistance(ghost.getPosition(), newPos)]  
 **if** ghost.getDirection() == Directions.STOP:  
 heuristic -= 20  
 minGhost = min(ghostDist)  
 foodList = newFood.asList()  
 foodDist = []  
 **for** food **in** foodList:  
 foodDist += [manhattanDistance(food, newPos)]  
 **if** currentGameState.getNumFood() > successorGameState.getNumFood():  
 heuristic += 100  
 inverse = 0  
 **if** len(foodDist) > 0:  
 inverse = (1.0 / float(min(foodDist)))  
 **if** minGhost < 5:  
 minGhost = minGhost \* 0.5  
 **if** minGhost >= 5 **and** minGhost <= 20:  
 minGhost = minGhost \* 1.1  
 **if** minGhost > 20:  
 minGhost = minGhost \* 1.7  
 heuristic += minGhost + successorGameState.getScore() + inverse \* 100  
 **return** heuristic  
  
  
**def** scoreEvaluationFunction(currentGameState):  
  
 **return** currentGameState.getScore()  
  
  
**class** MultiAgentSearchAgent(Agent):  
 *"""  
 Bu sınıf, tüm üyelerinize ortak öğeler sağlar.  
       çok ajanlı arama yapanlar. Burada tanımlanan herhangi bir yöntem mevcut olacak  
       MinimaxPacmanAgent, AlphaBetaPacmanAgent ve ExpectimaxPacmanAgent için.  
  
       Burada \* herhangi bir değişiklik yapmanız gerekmez, ancak isterseniz  
       tüm rakip arama ajanlarınıza işlevsellik ekleyin. Lütfen yapma  
       ancak bir şeyleri kaldırın.  
  
       Not: Bu soyut bir sınıftır: somutlaştırılmaması gereken. Onun  
       sadece kısmen belirtilmiş ve genişletilmek üzere tasarlanmıştır. Ajan (game.py)  
       başka bir soyut sınıftır.  
 """* **def** \_\_init\_\_(self, evalFn=**'scoreEvaluationFunction'**, depth=**'2'**):  
 self.index = 0 *# her zaman index 0* self.evaluationFunction = util.lookup(evalFn, globals())  
 self.depth = int(depth)  
  
  
**class** MinimaxAgent(MultiAgentSearchAgent):  
  
  
  
 **def** getAction(self, gameState):  
 *"""  
 Self.depth komutunu kullanarak geçerli gameState öğesinden minimax işlemini döndürür  
           ve kendini değerlendirme işlevi.  
  
           İşte minimax uygulanırken faydalı olabilecek bazı yöntem çağrıları.  
  
           gameState.getLegalActions (agentIndex):  
             Bir aracı için yasal işlemlerin bir listesini döndürür  
             agentIndex = 0, Pacman anlamına gelir, hayaletler> = 1  
  
           gameState.generateSuccessor (agentIndex, action):  
             Bir aracı işlem yaptıktan sonra halefi oyun durumunu döndürür  
  
           gameState.getNumAgents ():  
             Oyundaki toplam oyuncu sayısını döndürür  
  
           gameState.isWin ():  
             Oyun durumunun kazanan bir durum olup olmadığını döndürür  
  
           gameState.isLose ():  
             Oyun durumunun kaybedilen bir durum olup olmadığını döndürür  
 """* result = self.value(gameState, 0)  
 **return** result[1]  
  
 **def** value(self, gameState, depth):  
 numAgents = gameState.getNumAgents()  
 **if** gameState.isWin() **or** gameState.isLose() **or** (depth == self.depth \* numAgents):  
 **return** (self.evaluationFunction(gameState), **None**)  
  
 **if** ((depth % numAgents) == 0): *# max* **return** self.minOrMax(gameState, depth, 1)  
 **else**: *# min* **return** self.minOrMax(gameState, depth, 0)  
  
 **def** minOrMax(self, gameState, depth, pacman):  
 agent = [depth % gameState.getNumAgents(), 0]  
 values = [float(**"inf"**), float(**"-inf"**)]  
 val = (values[pacman], **None**)  
 possibleActions = gameState.getLegalActions(agent[pacman])  
 **if** len(possibleActions) == 0:  
 **return** (self.evaluationFunction(gameState), **None**)  
 **for** action **in** possibleActions:  
 successor = gameState.generateSuccessor(agent[pacman], action)  
 result = self.value(successor, depth + 1)  
 **if** ((pacman == 1) **and** result[0] > val[0]) **or** ((pacman == 0) **and** (result[0] < val[0])):  
 val = (result[0], action)  
 **return** val  
  
  
**class** AlphaBetaAgent(MultiAgentSearchAgent):  
 *"""  
        Alfa beta budama özelliğine sahip minimax  
      """* **def** getAction(self, gameState):  
 *"""  
           Self.depth ve self.evaluationFunction işlevini kullanarak minimax eylemini döndürür  
         """  
  
 # alfa negatif olarak başlar (maksimize eder), beta pozitif başlar (minimize eder)* result = self.value(gameState, 0, -float(**"inf"**), float(**"inf"**))  
 **return** result[1]  
  
 **def** value(self, gameState, depth, alpha, beta):  
 numAgents = gameState.getNumAgents()  
 **if** gameState.isWin() **or** gameState.isLose() **or** (depth == self.depth \* numAgents):  
 **return** (self.evaluationFunction(gameState), **None**)  
  
 **if** ((depth % numAgents) == 0):  
 **return** self.minOrMax(gameState, depth, 1, alpha, beta)  
 **else**:  
 **return** self.minOrMax(gameState, depth, 0, alpha, beta)  
  
 **def** minOrMax(self, gameState, depth, pacman, alpha, beta):  
 agent = [depth % gameState.getNumAgents(), 0]  
 values = [float(**"inf"**), float(**"-inf"**)]  
 val = (values[pacman], **None**)  
 possibleActions = gameState.getLegalActions(agent[pacman])  
 **if** len(possibleActions) == 0:  
 **return** (self.evaluationFunction(gameState), **None**)  
 **for** action **in** possibleActions:  
 successor = gameState.generateSuccessor(agent[pacman], action)  
 result = self.value(successor, depth + 1, alpha, beta)  
 **if** pacman:  
 **if** result[0] > val[0]:  
 val = (result[0], action)  
 **if** val[0] > beta:  
 **return** val  
 alpha = max(alpha, val[0])  
 **else**:  
 **if** result[0] < val[0]:  
 val = (result[0], action)  
 **if** val[0] < alpha:  
 **return** val  
 beta = min(beta, val[0])  
 **return** val  
  
  
**class** ExpectimaxAgent(MultiAgentSearchAgent):  
  
  
 **def** getAction(self, gameState):  
 *"""  
 Self.depth ve self.evaluationFunction işlevini kullanarak expectimax eylemini döndürür  
           Tüm hayaletler rastgele seçtikleri seçimlerden modellenmelidir.  
           yasal hamle.  
 """* result = self.value(gameState, 0)  
 **return** result[1]  
  
 **def** value(self, gameState, depth):  
 numAgents = gameState.getNumAgents()  
 **if** gameState.isWin() **or** gameState.isLose() **or** (depth == self.depth \* numAgents):  
 **return** (self.evaluationFunction(gameState), **None**)  
  
 **if** ((depth % numAgents) == 0): *# sonraki hayalet max* **return** self.expectOrMax(gameState, depth, 1)  
 **else**: *# sonraki hayalet min* **return** self.expectOrMax(gameState, depth, 0)  
  
 **def** expectOrMax(self, gameState, depth, pacman):  
 agent = [depth % gameState.getNumAgents(), 0]  
 values = [0, float(**"-inf"**)]  
 val = (values[pacman], **None**)  
 possibleActions = gameState.getLegalActions(agent[pacman])  
 **if** len(possibleActions) == 0:  
 **return** (self.evaluationFunction(gameState), **None**)  
 **for** action **in** possibleActions:  
 successor = gameState.generateSuccessor(agent[pacman], action)  
 result = self.value(successor, depth + 1)  
 **if** pacman:  
 **if** result[0] > val[0]:  
 val = (result[0], action)  
 **else**:  
 p = 1.0 / float(len(possibleActions))  
 val = (val[0] + (result[0] \* p), action)  
 **return** val  
  
  
**def** betterEvaluationFunction(currentGameState):  
  
 successorGameState = currentGameState  
 newPos = successorGameState.getPacmanPosition()  
 newFood = successorGameState.getFood()  
 newGhostStates = successorGameState.getGhostStates()  
 newScaredTimes = [ghostState.scaredTimer **for** ghostState **in** newGhostStates]  
  
 heuristic = 0  
 **for** states **in** newScaredTimes:  
 heuristic += states  
 ghostDist = []  
 **for** ghost **in** newGhostStates:  
 ghostDist += [manhattanDistance(ghost.getPosition(), newPos)]  
 **if** ghost.getDirection() == Directions.STOP:  
 heuristic -= 20  
 minGhost = min(ghostDist)  
 foodList = newFood.asList()  
 foodDist = []  
 **for** food **in** foodList:  
 foodDist += [manhattanDistance(food, newPos)]  
 **if** currentGameState.getNumFood() > successorGameState.getNumFood():  
 heuristic += 100  
 inverse = 0  
 **if** len(foodDist) > 0:  
 inverse = (1.0 / float(min(foodDist)))  
 **if** minGhost < 5:  
 minGhost = minGhost \* 0.5  
 **if** minGhost >= 5 **and** minGhost <= 20:  
 minGhost = minGhost \* 1.1  
 **if** minGhost > 20:  
 minGhost = minGhost \* 1.7  
 heuristic += minGhost + successorGameState.getScore() + inverse \* 100  
 **return** heuristic  
  
better = betterEvaluationFunction

*# layout.py***from** util **import** manhattanDistance  
**from** game **import** Grid  
**import** os  
**import** random  
  
VISIBILITY\_MATRIX\_CACHE = {}  
  
**class** Layout:  
   
  
 **def** \_\_init\_\_(self, layoutText):  
 self.width = len(layoutText[0])  
 self.height= len(layoutText)  
 self.walls = Grid(self.width, self.height, **False**)  
 self.food = Grid(self.width, self.height, **False**)  
 self.capsules = []  
 self.agentPositions = []  
 self.numGhosts = 0  
 self.processLayoutText(layoutText)  
 self.layoutText = layoutText  
 self.totalFood = len(self.food.asList())  
  
 **def** getNumGhosts(self):  
 **return** self.numGhosts  
  
 **def** initializeVisibilityMatrix(self):  
 **global** VISIBILITY\_MATRIX\_CACHE  
 **if** reduce(str.\_\_add\_\_, self.layoutText) **not in** VISIBILITY\_MATRIX\_CACHE:  
 **from** game **import** Directions  
 vecs = [(-0.5,0), (0.5,0),(0,-0.5),(0,0.5)]  
 dirs = [Directions.NORTH, Directions.SOUTH, Directions.WEST, Directions.EAST]  
 vis = Grid(self.width, self.height, {Directions.NORTH:set(), Directions.SOUTH:set(), Directions.EAST:set(), Directions.WEST:set(), Directions.STOP:set()})  
 **for** x **in** range(self.width):  
 **for** y **in** range(self.height):  
 **if** self.walls[x][y] == **False**:  
 **for** vec, direction **in** zip(vecs, dirs):  
 dx, dy = vec  
 nextx, nexty = x + dx, y + dy  
 **while** (nextx + nexty) != int(nextx) + int(nexty) **or not** self.walls[int(nextx)][int(nexty)] :  
 vis[x][y][direction].add((nextx, nexty))  
 nextx, nexty = x + dx, y + dy  
 self.visibility = vis  
 VISIBILITY\_MATRIX\_CACHE[reduce(str.\_\_add\_\_, self.layoutText)] = vis  
 **else**:  
 self.visibility = VISIBILITY\_MATRIX\_CACHE[reduce(str.\_\_add\_\_, self.layoutText)]  
  
 **def** isWall(self, pos):  
 x, col = pos  
 **return** self.walls[x][col]  
  
 **def** getRandomLegalPosition(self):  
 x = random.choice(range(self.width))  
 y = random.choice(range(self.height))  
 **while** self.isWall( (x, y) ):  
 x = random.choice(range(self.width))  
 y = random.choice(range(self.height))  
 **return** (x,y)  
  
 **def** getRandomCorner(self):  
 poses = [(1,1), (1, self.height - 2), (self.width - 2, 1), (self.width - 2, self.height - 2)]  
 **return** random.choice(poses)  
  
 **def** getFurthestCorner(self, pacPos):  
 poses = [(1,1), (1, self.height - 2), (self.width - 2, 1), (self.width - 2, self.height - 2)]  
 dist, pos = max([(manhattanDistance(p, pacPos), p) **for** p **in** poses])  
 **return** pos  
  
 **def** isVisibleFrom(self, ghostPos, pacPos, pacDirection):  
 row, col = [int(x) **for** x **in** pacPos]  
 **return** ghostPos **in** self.visibility[row][col][pacDirection]  
  
 **def** \_\_str\_\_(self):  
 **return "\n"**.join(self.layoutText)  
  
 **def** deepCopy(self):  
 **return** Layout(self.layoutText[:])  
  
 **def** processLayoutText(self, layoutText):  
 *"""  
  
         Labirentin şekli. Her karakter  
         farklı bir nesne türünü temsil eder.  
          % - Duvar  
          . - Gıda  
          o - Kapsül  
          G - Hayalet  
 P - Pacman  
 Other characters are ignored.  
 """* maxY = self.height - 1  
 **for** y **in** range(self.height):  
 **for** x **in** range(self.width):  
 layoutChar = layoutText[maxY - y][x]  
 self.processLayoutChar(x, y, layoutChar)  
 self.agentPositions.sort()  
 self.agentPositions = [ ( i == 0, pos) **for** i, pos **in** self.agentPositions]  
  
 **def** processLayoutChar(self, x, y, layoutChar):  
 **if** layoutChar == **'%'**:  
 self.walls[x][y] = **True  
 elif** layoutChar == **'.'**:  
 self.food[x][y] = **True  
 elif** layoutChar == **'o'**:  
 self.capsules.append((x, y))  
 **elif** layoutChar == **'P'**:  
 self.agentPositions.append( (0, (x, y) ) )  
 **elif** layoutChar **in** [**'G'**]:  
 self.agentPositions.append( (1, (x, y) ) )  
 self.numGhosts += 1  
 **elif** layoutChar **in** [**'1'**, **'2'**, **'3'**, **'4'**]:  
 self.agentPositions.append( (int(layoutChar), (x,y)))  
 self.numGhosts += 1  
**def** getLayout(name, back = 2):  
 **if** name.endswith(**'.lay'**):  
 layout = tryToLoad(**'layouts/'** + name)  
 **if** layout == **None**: layout = tryToLoad(name)  
 **else**:  
 layout = tryToLoad(**'layouts/'** + name + **'.lay'**)  
 **if** layout == **None**: layout = tryToLoad(name + **'.lay'**)  
 **if** layout == **None and** back >= 0:  
 curdir = os.path.abspath(**'.'**)  
 os.chdir(**'..'**)  
 layout = getLayout(name, back -1)  
 os.chdir(curdir)  
 **return** layout  
  
**def** tryToLoad(fullname):  
 **if**(**not** os.path.exists(fullname)): **return None** f = open(fullname)  
 **try**: **return** Layout([line.strip() **for** line **in** f])  
 **finally**: f.close()

*# grading.py***import** cgi  
**import** time  
**import** sys  
**import** json  
**import** traceback  
**import** pdb  
**from** collections **import** defaultdict  
**import** util  
  
**class** Grades:  
   
 **def** \_\_init\_\_(self, projectName, questionsAndMaxesList,  
 gsOutput=**False**, edxOutput=**False**, muteOutput=**False**):  
 *"""  
 Bir projenin derecelendirme şemasını tanımlar  
       projectName: proje adı  
       questionsAndMaxesDict: (soru adı, soru başına maksimum puan) listesi  
 """* self.questions = [el[0] **for** el **in** questionsAndMaxesList]  
 self.maxes = dict(questionsAndMaxesList)  
 self.points = Counter()  
 self.messages = dict([(q, []) **for** q **in** self.questions])  
 self.project = projectName  
 self.start = time.localtime()[1:6]  
 self.sane = **True** self.currentQuestion = **None** self.edxOutput = edxOutput  
 self.gsOutput = gsOutput  
 self.mute = muteOutput  
 self.prereqs = defaultdict(set)  
  
 print **'başlangıç on %d-%d at %d:%02d:%02d'** % self.start  
  
 **def** addPrereq(self, question, prereq):  
 self.prereqs[question].add(prereq)  
  
 **def** grade(self, gradingModule, exceptionMap = {}, bonusPic = **False**):  
  
 completedQuestions = set([])  
 **for** q **in** self.questions:  
 print **'\nQuestion %s'** % q  
 print **'='** \* (9 + len(q))  
 print  
 self.currentQuestion = q  
  
 incompleted = self.prereqs[q].difference(completedQuestions)  
 **if** len(incompleted) > 0:  
 prereq = incompleted.pop()  
 print \  
% (prereq, q, q, prereq)  
 **continue  
  
 if** self.mute: util.mutePrint()  
 **try**:  
 util.TimeoutFunction(getattr(gradingModule, q),1800)(self)  
 **except** Exception, inst:  
 self.addExceptionMessage(q, inst, traceback)  
 self.addErrorHints(exceptionMap, inst, q[1])  
 **except**:  
 self.fail(**'FAIL: Bir dizgi istisnası ile sonlandırıldı.'**)  
 **finally**:  
 **if** self.mute: util.unmutePrint()  
  
 **if** self.points[q] >= self.maxes[q]:  
 completedQuestions.add(q)  
  
 print (**'\n### Question %s: %d/%d ###\n'** % (q, self.points[q], self.maxes[q]))  
  
  
 print (**'\nFinished at %d:%02d:%02d'** % time.localtime()[3:6])  
 print (**"\nProvisional grades\n=================="**)  
  
 **for** q **in** self.questions:  
 print (**'Question %s: %d/%d'** % (q, self.points[q], self.maxes[q]))  
 print (**'------------------'**)  
 print (**'Total: %d/%d'** % (self.points.totalCount(), sum(self.maxes.values())))  
 **if** bonusPic **and** self.points.totalCount() == 25:  
 **"""  
  
   
  
 --- ---- ---  
 | \ / + \ / |  
 | + \--/ \--/ + |  
 | + + |  
 | + + + |  
 @@@@@@@@@@@@@@@@@@@@@@@@@@  
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  
 \ @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  
 \ / @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  
 V \ @@@@@@@@@@@@@@@@@@@@@@@@@@@@  
 \ / @@@@@@@@@@@@@@@@@@@@@@@@@@  
 V @@@@@@@@@@@@@@@@@@@@@@@@  
 @@@@@@@@@@@@@@@@@@@@@@  
 /\ @@@@@@@@@@@@@@@@@@@@@@  
 / \ @@@@@@@@@@@@@@@@@@@@@@@@@  
 /\ / @@@@@@@@@@@@@@@@@@@@@@@@@@@  
 / \ @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  
 / @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  
 @@@@@@@@@@@@@@@@@@@@@@@@@@  
 @@@@@@@@@@@@@@@@@@  
  
"""  
  
 if** self.edxOutput:  
 self.produceOutput()  
 **if** self.gsOutput:  
 self.produceGradeScopeOutput()  
  
 **def** addExceptionMessage(self, q, inst, traceback):  
  
 self.fail(**'FAIL: Exception raised: %s'** % inst)  
 self.addMessage(**''**)  
 **for** line **in** traceback.format\_exc().split(**'\n'**):  
 self.addMessage(line)  
  
 **def** addErrorHints(self, exceptionMap, errorInstance, questionNum):  
 typeOf = str(type(errorInstance))  
 questionName = **'q'** + questionNum  
 errorHint = **''  
 if** exceptionMap.get(questionName):  
 questionMap = exceptionMap.get(questionName)  
 **if** (questionMap.get(typeOf)):  
 errorHint = questionMap.get(typeOf)  
 *# hata varsa başa dön* **if** (exceptionMap.get(typeOf)):  
 errorHint = exceptionMap.get(typeOf)  
  
 **if not** errorHint:  
 **return ''  
  
 for** line **in** errorHint.split(**'\n'**):  
 self.addMessage(line)  
  
 **def** produceGradeScopeOutput(self):  
 out\_dct = {}  
  
 *# gönderimin toplamı* total\_possible = sum(self.maxes.values())  
 total\_score = sum(self.points.values())  
 out\_dct[**'score'**] = total\_score  
 out\_dct[**'max\_score'**] = total\_possible  
 out\_dct[**'output'**] = **"Total score (%d / %d)"** % (total\_score, total\_possible)  
 tests\_out = []  
 **for** name **in** self.questions:  
 test\_out = {}  
 test\_out[**'name'**] = name  
 test\_out[**'score'**] = self.points[name]  
 test\_out[**'max\_score'**] = self.maxes[name]  
  
 is\_correct = self.points[name] >= self.maxes[name]  
 test\_out[**'output'**] = **" Question {num} ({points}/{max}) {correct}"**.format(  
 num=(name[1] **if** len(name) == 2 **else** name),  
 points=test\_out[**'score'**],  
 max=test\_out[**'max\_score'**],  
 correct=(**'X' if not** is\_correct **else ''**),  
 )  
 test\_out[**'tags'**] = []  
 tests\_out.append(test\_out)  
 out\_dct[**'tests'**] = tests\_out  
  
 **with** open(**'gradescope\_response.json'**, **'w'**) **as** outfile:  
 json.dump(out\_dct, outfile)  
 **return  
  
 def** produceOutput(self):  
 edxOutput = open(**'edx\_response.html'**, **'w'**)  
 edxOutput.write(**"<div>"**)  
  
 *# birinci* total\_possible = sum(self.maxes.values())  
 total\_score = sum(self.points.values())  
 checkOrX = **'<span class="incorrect"/>'  
 if** (total\_score >= total\_possible):  
 checkOrX = **'<span class="correct"/>'** header = **"""  
 <h3>  
 Total score ({total\_score} / {total\_possible})  
 </h3>  
 """**.format(total\_score = total\_score,  
 total\_possible = total\_possible,  
 checkOrX = checkOrX  
 )  
 edxOutput.write(header)  
  
 **for** q **in** self.questions:  
 **if** len(q) == 2:  
 name = q[1]  
 **else**:  
 name = q  
 checkOrX = **'<span class="incorrect"/>'  
 if** (self.points[q] >= self.maxes[q]):  
 checkOrX = **'<span class="correct"/>'** messages = **"<pre>%s</pre>"** % **'\n'**.join(self.messages[q])  
 output = **"""  
 <div class="test">  
 <section>  
 <div class="shortform">  
 Question {q} ({points}/{max}) {checkOrX}  
 </div>  
 <div class="longform">  
 {messages}  
 </div>  
 </section>  
 </div>  
 """**.format(q = name,  
 max = self.maxes[q],  
 messages = messages,  
 checkOrX = checkOrX,  
 points = self.points[q]  
 )  
 edxOutput.write(output)  
 edxOutput.write(**"</div>"**)  
 edxOutput.close()  
 edxOutput = open(**'edx\_grade'**, **'w'**)  
 edxOutput.write(str(self.points.totalCount()))  
 edxOutput.close()  
  
 **def** fail(self, message, raw=**False**):  
 self.sane = **False** self.assignZeroCredit()  
 self.addMessage(message, raw)  
  
 **def** assignZeroCredit(self):  
 self.points[self.currentQuestion] = 0  
  
 **def** addPoints(self, amt):  
 self.points[self.currentQuestion] += amt  
  
 **def** deductPoints(self, amt):  
 self.points[self.currentQuestion] -= amt  
  
 **def** assignFullCredit(self, message=**""**, raw=**False**):  
 self.points[self.currentQuestion] = self.maxes[self.currentQuestion]  
 **if** message != **""**:  
 self.addMessage(message, raw)  
  
 **def** addMessage(self, message, raw=**False**):  
 **if not** raw:  
 **if** self.mute: util.unmutePrint()  
 print (**'\*\*\* '** + message)  
 **if** self.mute: util.mutePrint()  
 message = cgi.escape(message)  
 self.messages[self.currentQuestion].append(message)  
  
 **def** addMessageToEmail(self, message):  
 print (**"WARNING\*\*\*\* addMessageToEmail is deprecated %s"** % message)  
 **for** line **in** message.split(**'\n'**):  
 **pass  
  
  
class** Counter(dict):  
  
 **def** \_\_getitem\_\_(self, idx):  
 **try**:  
 **return** dict.\_\_getitem\_\_(self, idx)  
 **except** KeyError:  
 **return** 0  
  
 **def** totalCount(self):  
 *"""  
 toplamını döndürür  
 """* **return** sum(self.values())

# **P**ROJE GERÇEKLEŞTİRİMİNDE YARARLANILAN KAYNAKLAR

* <http://aytugonan.cbu.edu.tr/>
* GitHub
* Şadi Evren Şeker(Bilgisayar Kavramları )