Командный тур

5.1. Задачи

Краткая справка

Финальный этап профиля проводился на специально разработанном стенде-тренажере «ИЭС» (Подробное устройство стенда представлено в Приложении 1). Участникам предстояло работать с моделью энергосистемы, в которой присутствует большое количество альтернативных источников энергии, накопителей энергии и широкие возможности для экономических действий. В результате участники изучали: физические и экономические параметры энергосетей; сильные и слабые стороны альтернативной энергетики, взаимосвязь инженерных и экономические решений и проблему качества и надежности электросетей. В финальной задаче участникам требовались навыки и знания из школьной программы, а также самостоятельно приобретенные на хакатонах и из требуемых для решения задач второго этапа:

- Умение программировать на языке Python.
- Умение работать со случайными величинами в программных вычислениях.
- Умение решать базовые оптимизационные задачи.
- Навыки реализации решений математических задач в виде программ.
- Понимание правил и решения закрытого аукциона первой цены.
- Навыки командной работы над программным кодом.
- Навыки работы с большими рядами данных в математических задачах.
- Навыки работы с рядами данных в алгоритмах.
- Понимание принципов работы и характеристик ветрогенераторов.
- Теория вероятностей.

Успешное решение финальной задачи предполагает освоение и развитие следующих знаний и навыков:

- Умение работы в команде.
- Умение самостоятельно выделять и формулировать подзадачи.
- Понимание принципов оценки риска.
- Навыки программирования, в том числе на языке Python.
- Навыков работы с рядами данных, как аналитической, так и алгоритмической.
- Навыки работы с системами с инерцией.

- Практического использования теории вероятностей, в том числе в программных вычислениях.

Финальная командная задача: Моделирование энергосистемы и разработка алгоритмов управления энергообеспечения

На стенде моделируется два небольших города, энергосистемам которого предстоит быть полностью перестроенной. В том числе, в энергосистемы будет добавлено большое количество ветровой и солнечной генерации.

Энергосистемы объединены в одну и подсоединены к внешней энергосистеме по схеме «микрогрид»: каждая из энергосистем сначала балансируется собственными ресурсами, а затем пытается получить недостающую или передать избыточную энергию энергосистемам оппонентов. Только после этого все энергосистемы полностью балансируются через внешнюю энергосистему.

Каждая будущая энергосистема будет «разделена» между четырьмя конкурирующими компаниями. Каждая команда станет одной из восьми конкурирующих энергокомпаний и будет строить свою собственную энергосистему и управлять ей.

Разделение будет происходить через цепочку аукционов, в которых участники изначально будут в принципиально одинаковых условиях. Часть объектов могла быть установлена на любом стенде (они имели две идентичные копии на обеих площадках проведения), часть была «привязана» к стенду. «Привязанные» объекты имели строго одинаковые характеристики.

После этого команды проектируют собственные энергосистемы (из одних и тех же объектов можно составить очень разные по эффективности энергосистемы) и готовят скрипты для управления ими. Далее происходит натурное моделирование нескольких дней работы энергосистемы, в ходе которого участники управляют своими энергосистемами посредством скриптов; и происходит экспериментальное измерение эффективности построенных энергосистем. Очки, набранные командами во время моделирования, пересчитываются в баллы, которые их участники получат за командную часть олимпиады.

Проведение финала происходило в виде командного турнира. Цель команд участников — набрать наибольшее число баллов в турнире.

Финальный командный тур был распределенный, проходил на двух площадках — в Москве и Иркутске. На каждой площадке игры проводились на идентичных стендах. Связь между стендами и главным сервером игры осуществлялась через сеть Интернет. Возможности стенда позволяли одновременно играть 4-м командам, итого в одной игре одновременно могло принимать участие до 8 команд.

Регламент турнира

1. Подготовка.
    (a) Проводился уже сформированными командами. Это продолжительный по времени этап (3 дня), в течение которого участники знакомились с правилами, тренировались работать на стенде, изучали предоставленную систему и готовили заготовки (управляющие скрипты, стратегии и вспомогательные программы).

2. Два полуфинала турнира
    (a) Команды делились на две группы по 7 команд (по 4 –Москва, по 3 - Иркутск) случайным образом. С каждой группой проводился отдельный полуфинал.
    (b) В полуфинале проводилась одна игра.
3. Финал турнира
    (a) В финал проходили по две команды с каждого стенда с каждого полуфинала (8 команд), набравшие наибольшее число очков (у.е.).
    (b) В финале проходила одна игра.
    (c) Победитель определялся по сумме очков (у.е.) за игру.

Этапы одной игры

1. Анализ прогнозов погоды. Минимум за 20 минут до игры участникам выдавались прогнозы погоды и потребления для каждого потребителя в предстоящей игре. За это время участники должны были спроектировать энергосистему, наиболее отвечающую предстоящим условиям.
2. Основной аукцион. На этом этапе определялось, какой объект к чьей энергосистеме будет подключён. Аукцион закрытого типа, с продолжением в случае одинаковых ставок. Этот этап практически невозможно успешно пройти без глубокого предварительного анализа и заготовленных программ для поддержки принятия решений.
3. Дополнительный аукцион. Проводился через 1 минуту после основного. Каждая команда имела право повторно «выставить на торги» любой из приобретённых ранее объектов. Этот этап давал командам шанс на исправление одной ошибки, если она не слишком велика.
4. Монтаж энергосистемы и адаптация стратегии. Участникам было предоставлено на сборку сети 20 минут. За это время они находили оптимальную конфигурацию своей энергосистемы, монтировали её на стенде, включая оптимальную установку электростанций. В это же время команда адаптировала к получившейся энергосистеме составленные заранее управляющие скрипты.
5. Моделирование. Команды в полном составе находились возле собственных терминалов управления; к стенду запрещено было приближаться ближе, чем на 2 метра всем, включая обслуживающий персонал, во избежание влияния на физические измерения. Участники наблюдали за работой своих скриптов и могли его заменить, например, в случае обнаружения ошибки. Число загружаемых скриптов правилами не было ограничено. Всё управление на этом этапе осуществлялось только скриптами.

Аукцион

Аукцион проводился по закрытой схеме. Выигрывал предложивший наибольшую цену в аукционе на электростанции, и наименьшую — в аукционе на потребителей. Стартовая цена для электростанций составляла 1, для потребителей — 10.

На ставку отводилось 30 секунд. В случае близости наилучших ставок на 0,5 или

менее, проводился дополнительный тур аукциона, в котором участвовали только те, чьи ставки попали в диапазон 0,5 от ставки лидера.

В случае, если один из участников достигает предельной цены, аукцион заканчивается. Если два и более участника достигли предельной цены, для них начинался аукцион по системе «all-pay» («платят все»). В этом аукционе свои ставки выплачивают все: и победители, и проигравшие. Итоговые ставки участников этого аукциона вычитались из их результата.

Порядок выставления лотов фиксирован и известен участникам заранее.

В течение минуты после окончания аукциона каждая команда имела право выставить на торги один из своих объектов. Команда ничего с этого не приобретает, но получает возможность избавиться от лишнего объекта, нарушающего баланс энергосистемы. Команда не имела права делать ставки на объект, который выставила на торги. У каждой команды в наличии по умолчанию имелось один дом и одна солнечная электростанция.

Пример результатов аукциона

Подзадачи финальной задачи

В ходе выполнения финальной задачи оценивалось комплексное решение финальной задачи профиля. Выделение и формулирование подзадачи является частью интегральной (главной) задачи профиля. Проверкой решения подзадачи являлся вклад её в результат игры. Описанные ниже задачи выделены разработчиками и были известны участником из описания. Распределение усилий между подзадачами принимали сами команды.

Физика

Задачи из физики присутствовали, однако их физическая составляющая была сведена к минимуму, и их физические модели были тривиальными.

Оптимальное расположение ветряка

Задача возникает на 4-м этапе игры, при монтаже энергосистемы. Кроме того, команда должна знать, насколько эффективно она может решить эту задачу для эффективной работы на этапах 1–3 — при анализе прогнозов погоды и при участии в аукционе.

Первая гипотеза «установить ветряк как можно ближе к анемометру» полностью разрушается двумя факторами:

1. При приближении к вентилятору создаваемое им ветровое поле становится всё более неравномерным. Например, на направлении его оси скорость ветра снижается, поскольку проталкивание воздуха осуществляется не всей плоскостью вентилятора, а только его лопастями. На следующем порядке малости на скорость ветра влияет расположение вентилятора относительно стенда и стен помещения, расположения других объектов на стенде (в особенности других ветряков).
2. Устройство анемометра, установленного на модели ветряка,- вертикально-осевое. На скорость его вращения, в большей степени, чем сама скорость ветра, влияет разность ветрового давления, создаваемого на правую и левую его часть. Получается, что анемометр нужно устанавливать не в зоне большего ветра, а в зоне, в которой силы, вращающие его в «правильную» сторону (против часовой стрелки), будут максимально превосходить силы, вращающие его в противоположную сторону.

Эта задача оптимального расположения весьма эффективно и просто решается при помощи простого наблюдения за скоростью вращения анемометра.

Оптимальное расположение солнечной батареи

Эта задача возникает на 4-м этапе игры, при монтаже энергосистемы. Кроме того, команда должна знать, насколько эффективно она может решить эту задачу для эффективной работы на этапах 1–3 — при анализе прогнозов погоды и при участии в аукционе.

Задача оптимального расположения также достаточно эффективно решается при помощи простой серии экспериментов и наблюдения.

Определение взаимосвязи между яркостью освещения и генерацией солнечных батарей

Эта задача возникает на 5-м этапе игры, при моделировании энергосистемы. Вырабатываемая мощность солнечных батарей зависит от напряжения на солнечных панелях модели солнечной электростанции на стенде. Напряжение на солнечных панелях по отношению к яркости светильников, строго говоря, нелинейно. Однако

характеристики солнечных панелей, измеряющих цепей и светильников подобраны так, что отклонение реальных значений от линейной их аппроксимации составляет не более 2

Время релаксации измерительной системы солнечных батарей в 2,5 раза меньше минимального интервала между изменением яркости и измерением вырабатываемой мощности, поэтому генерация солнечных батарей зависит только от погоды на текущем такте игры.

Определение взаимосвязи между скоростью ветра и генерацией ветряков

Эта задача возникает на 5-м этапе игры, при моделировании энергосистемы. Кроме того, команда должна иметь решение этой задачи для эффективной работы на этапах 1–3 — при проектировании энергосистемы и при участии в аукционе.

Задача вычисления генерации ветровых электростанций по данным погоды похоже на такую же задачу для солнечных батарей, но является более сложной.

Для получения величины генерации используется скорость вращения анемометра, который установлен на модели ветровой электростанции. Устройство анемометра — вертикально-осевой; его лопасти устроены так, что скорость его вращения линейно пропорциональна скорости ветра (если считать поток ветра гомогенным). В случае постоянной негомогенности ветрового потока (когда анемометр не перемещается) скорость вращения анемометра также линейна по отношению к максимальной скорости ветра в потоке.

Измерительная система — инерциальная система вращения с трением. Её параметры нужно вычислять по данным прошедших игр. Время полной остановки анемометра с максимальной скорости вращения составляет около 3 тактов игры (оно зависит от максимальной скорости ветра в месте расположения анемометра). Время полного разгона аналогично составляет 1 такт игры.

Генерируемая мощность пропорциональная кубу скорости вращения анемометра. При достижении максимального уровня генерации (15 МВт) мощность далее не растёт.

Зависимость генерации от скорости ветра можно оценить либо эвристически, либо как линейную комбинацию от погоды за последние 3 такта игры на основании данных прошедших игр.

Ветровую электростанцию возможно установить так, что максимальная мощность будет достигаться даже при сравнительно небольших скоростях ветра (по данным погоды). Вычисление коэффициента в зависимости генерации от скорости ветра необходимо делать экспериментально. Это можно оценить предварительными экспериментами, либо заложить процедуру оценки в управляющий скрипт, чтобы он делал её на каждом такте.

Например, вычисленные «задним числом» коэффициенты линейной комбинации от прогнозов погоды за финальную игру составляют для наилучшего ветрогенератора:

1. 0,69 от погоды на такт, для которого вычисляем прогноз.
2. 0,22 от погоды за предыдущий такт

3. 0,09 от погоды за пред-предыдущий такт

При вычислениях необходимо учитывать тот факт, что генерация ветровых электростанций ограничена правилами на уровне 15 МВт. Поэтому если прогнозируемая генерация превышает этот уровень, надо считать, что спрогнозировано именно 15. Средняя величина ошибки такого набора коэффициентов между прогнозируемой и реальной генерацией на играх финала составила 4,2%, что, с одной стороны, связано с большой инерциальностью физического анемометра. С другой стороны, из-за кубической зависимости генерации от силы ветра, любые погрешности «в середине» диапазона скоростей ветра очень сильно влияют на прогнозируемую мощность. Если скорость ветра мала или велика, погрешности влияют очень слабо.

Решение

```haskell
module Main where

import System.Environment
import System.IO
import Data.Ord
import Data.List

testFile = "/home/user/ips2019/sampleData.txt"

main :: IO ()
main = do
  h <- openFile testFile ReadMode
  hSetEncoding h utf8
  contents <- hGetContents h
  let loglines = tail . lines $ contents
  let grs = map readOneLine loglines
  print grs
  print $ findBest grs
  --print $ searchBest 10 ( metrics grs ) [] $ allCombs 5 10

data Gr = Gr
  { sun
  , sunGen :: Float
  } deriving ( Show )

readOneLine :: String -> Gr
readOneLine str = Gr w wg
  where
  (_:_:w:wg:_) = map read $ words str

data LC = LC [ Float ]
  deriving ( Show )

rlc ( LC x ) = LC ( reverse x )

theoreticDiff :: [ Gr ] -> LC -> Maybe Float
theoreticDiff grs (LC lcs)
  | length grs < length lcs = Nothing
  | otherwise = Just $ (^2) $ sum sunny - sunGen ( last grs )
  where
  pairs = zip grs $ reverse lcs
  ony ( gr, coeff ) = sun gr * coeff
  sunny = map ony pairs

```

```haskell
metrics :: [ Gr ] -> LC -> Float
metrics [] _ = 0
metrics grs lc = case theoreticDiff grs lc of
  Nothing -> 0
  Just val -> val + metrics ( tail grs ) lc

allCombs :: Int -> Int -> [ LC ]
allCombs len steps = map LC $ map normalize $ sequence $ replicate len list
  where
  normalize :: [ Float ] -> [ Float ]
  normalize fs | sum fs == 0 = fs
               | otherwise = map (/ sum fs ) fs
  list = map fromIntegral [0..steps-1]

findBest :: [ Gr ] -> LC
findBest grs = minimumBy ( comparing ( metrics grs )) $ allCombs 10 3

searchBest :: Int -> ( a -> Float ) -> [ a ] -> [ a ] -> [ a ]
searchBest _ _ accum [] = accum
searchBest limit f accum (lc:lcs)
  | length accum < limit = searchBest limit f (lc:accum) lcs
  | otherwise = searchBest limit f ( tail $ sortBy ( comparing f ) (lc:accum)) lcs
```

Математика

Вычисление полного энергетического баланса на основании данных прогнозов

Эта задача возникает на 1-м и 5-м этапах игры, при анализе прогнозов и моделировании энергосистемы.

Всё время игры командам доступны все данные о прогнозах погоды и действующих контрактах. Из них можно вычислить прогноз дефицита или профицита мощности на каждый такт. Для этого нужно на основании составленных заранее игроками моделей вычислить из прогнозов погоды прогнозы генерации. Из результирующего множества случайных величин (вероятная генерация для каждой электростанции и вероятное потребления для каждого потребителя) нужно вычислить их сумму. Она будет представлять собой распределение вероятностей профицита/дефицита мощности в системе. Важно учитывать, что максимальный дефицит или профицит мощности ограничен главной подстанцией и установленными на ней объектами.

Решение

Приведённый ниже модуль на языке Python3 называется powerbalance и будет использоваться почти во всех остальных решениях

```python
import operator as o

#Этот модуль будет использоваться почти во всех остальных решениях

#Операция на «нечётких множествах»
def fuzzyop(fuz1, fuz2, op):
    result = []
    for (val1,prob1) in fuz1:
        for (val2,prob2) in fuz2:
```

```
10              result.append((op(val1,val2),prob1*prob2))
11      return result
12
13  #Оптимизировать представление случайной величины
14  def squash(d):
15      d.sort(key=lambda x: x[0])
16      l=len(d)
17      newD = []
18      acc = 0
19      for i in range(len(d)-1):
20          if d[i][0] != d[i+1][0]:
21              x = (d[i][0], acc+d[i][1] )
22              newD.append(x)
23              acc = 0
24          else:
25              acc += d[i][1]
26      x = (d[len(d)-1][0], acc+d[len(d)-1][1])
27      newD.append(x)
28      return newD
29
30  #Округление по произвольной базе
31  def myround(value,step):
32      mod = value % step
33      div = value // step
34      if mod > step / 2:
35          return step * ( div + 1 )
36      else:
37          return step * div
38
39  #Огрубить случайную величину. чтобы ускорить вычисления
40  #Желательно также убирать вероятности ниже порогового значения
41  roughstep = 0.2
42  def rough(fuz):
43      result = [(myround(val,roughstep),prob) for (val,prob) in fuz]
44      return squash(result)
45
46  #Получить численную случайную величину из прогноза
47  stepsize = 0.1
48  def fromForecast(forecast):
49      lower = forecast / 1.2
50      upper = forecast / 0.8
51      stepsLower = []
52      x = forecast
53      while x >= lower:
54          stepsLower.append(x)
55          x -= stepsize
56      lowers = []
57      for l in stepsLower:
58          lowers.append((l,1/len(stepsLower)))
59      stepsUpper = []
60      x = forecast + stepsize
61      while x <= upper:
62          stepsUpper.append(x)
63          x += stepsize
64      uppers = []
65      for l in stepsUpper:
66          uppers.append((l,1/len(stepsUpper)))
67      result = lowers + uppers
68      return result
69
```

```
70  #Прочитать прогнозы ветра
71  with open('wind.txt') as f:
72      tmp = f.read().splitlines()
73      wind = [ eval(x) for x in tmp ]
74
75  #Прочитать прогнозы солнца
76  with open('sun.txt') as f:
77      tmp = f.read().splitlines()
78      sun = [ eval(x) for x in tmp ]
79
80  #Прочитать прогнозы потребления домов
81  with open('houses.txt') as f:
82      tmp = f.read().splitlines()
83      houses = [ eval(x) for x in tmp ]
84
85  #Прочитать прогнозы потребления заводов
86  with open('factories.txt') as f:
87      tmp = f.read().splitlines()
88      factories = [ eval(x) for x in tmp ]
89
90  #Прочитать прогнозы потребления больниц
91  with open('hospitals.txt') as f:
92      tmp = f.read().splitlines()
93      hospitals = [ eval(x) for x in tmp ]
94
95  #Линейные коэффициенты генерации для ветра и солнца
96  coefSun = [0.12, 0.94, 0.7]
97  coefWind = [0.09, 0.32, 0.49, 0.41, 0.27, 0.16]
98
99  # Вычисление прогноза генерации
100 def powerForecast(coefficients,values,tick):
101     power = [(0,1)]
102     for i in range(0,len(coefficients)):
103         if tick-i < 0:
104             val=0
105         else:
106             val=values[tick-i]
107         part = fuzzyop(fromForecast(val),[(coefficients[i],1)],o.mul)
108         power = fuzzyop(power,part,o.add)
109     return rough(power)
110
111 # Вычисление прогноза генерации
112 def powerSun(tick):
113     return powerForecast(coefSun,sun,tick)
114
115 # Вычисление прогноза генерации
116 linear = 0.44 #коэффициент при x^3
117 def powerWind(tick):
118     tmp = powerForecast(coefWind,wind,tick)
119     return [(max(15,linear*(x**3)),p) for (x,p) in tmp ]
120
121 class Network:
122     houses = 0
123     hospitals = 0
124     factories = 0
125     wind = 0
126     solar = 0
127     def __init__(self,h,f,b,w,s):
128         self.houses = h
129         self.hospitals = b
```

```
130         self.factories = f
131         self.wind = w
132         self.solar = s
133 
134 # Задаём состав сети
135 network = Network(2,2,1,1,2)
136 
137 def powerBalance(net):
138     power = [(0,1)]
139     for tick in range(0,3):
140         for _ in range(0,net.houses):
141             power=fuzzyop(power,fromForecast(houses[tick]),o.sub)
142         power=rough(power)
143         for _ in range(0,net.factories):
144             power=fuzzyop(power,fromForecast(factories[tick]),o.sub)
145         power=rough(power)
146         for _ in range(0,net.hospitals):
147             power=fuzzyop(power,fromForecast(hospitals[tick]),o.sub)
148         power=rough(power)
149         for _ in range(0,net.solar):
150             power=fuzzyop(power,powerSun(tick),o.add)
151         power=rough(power)
152         for _ in range(0,net.wind):
153             power=fuzzyop(power,powerWind(tick),o.add)
154     return rough(power)
155 
156 def expect(fuz):
157     result = 0
158     for (v,p) in fuz:
159         result += v*p
160     return result
161 
162 #Напечатать математическое ожидание энергетического баланса
163 print(expect(powerBalance(network)))
164 
```

Вычисление баланса энергорайонов энергосистемы

Каждая энергосистема состоит из набора подстанций и подключённых к ним объектов. Эффективно энергосистема представляет собой дерево, в узлах которого находятся подстанции, а рёбра представляют собой энергорайоны с множеством подключённых объектов и, что очень важно, двумя узлами подстанций, к которым подключен энергорайон (причём если энергорайон физически подключён только к одной подстанции, то второй узел виртуален). По правилам мощность, протекающая через один узел, не может быть больше 15 МВт, иначе узел аварийно отключится. Это приводит к тому, что необходимо прогнозировать энергобаланс в каждом энергорайоне по-отдельности (полностью аналогично нахождению общего баланса), и учитывать складывание токов в энергосистеме при протекании их по дереву. Эта задача похожа на задачу «Раздеревенение» второго заочного тура олимпиады прошлого года, и задачу «Посади электродерево» очного тура по информатике этого года.

Вычисление экономического баланса энергосистемы на основании данных прогнозов

Эта задача возникает на 1-м и 5-м этапах игры, при анализе прогнозов и моделировании энергосистемы. Кроме того, команда должна иметь решение этой задачи для эффективной работы на этапах 2–3 — участии в аукционе.

Далее нужно решить задачу нахождения вероятностного распределения экономических потерь. Это делается почти тривиально при принятии консервативной оценки прибылей и убытков от взаимодействия с внешней энергосистемой: каждый мегаватт непредсказанной избыточной мощности эффективно несёт убыток в 1 у.е., а каждый мегаватт недостаточной — 2 у.е. К этим значениям нужно добавить стоимость электроэнергии, закупленной/проданной во внешней энергосистеме по цене за 1 такт вперёд. Далее математическое ожидание получившейся случайной величины нужно минимизировать, используя параметры направляемой/извлекаемой мощности из аккумуляторов и закупаемой/продаваемой мощности во внешней энергосистеме. Выгоды от торговли с другими командами и ранних закупок во внешней энергосистеме можно учитывать независимо.

Эта задача немного проще, чем кажется из-за того, что использование аккумуляторов экономически намного выгоднее взаимодействия с внешней энергосистемой, имеет смысл закупать/продавать электроэнергию в ней только в случае невозможности сделать это через аккумулятор. Эти два параметра оказываются связаны в один — «балансирующее воздействие на энергосистему», которое можно разбить на аккумуляторы и внешнюю энергосистему «задним числом» после решения этой задачи.

Эту задачу (нахождение балансирующего воздействия, минимизирующего математическое ожидание экономических потерь) недостаточно решить аналитически, алгоритм решения нужно реализовать также в управляющем скрипте (и других вспомогательных программах), что для большинства школьников является нетривиальной и неустойчивой к ошибкам задачей. Немного проще её решить «перебором» — перебрав все значения балансирующего воздействия в размахе случайного распределения дефицита/профицита мощности в системе с фиксированным шагом, например, 0,1. Такой точности будет вполне достаточно, такое решение можно реализовать быстрее, чем точное, и впоследствии при наличии времени его можно точным заменить.

В дальнейшем на основании этих данных можно вычислить распределение вероятностей прибыли за всю игру.

Решение

Программа представляет собой модуль costbalance, который будет использоваться в примере решения задачи нахождения предельной стоимости объекта на аукционе.

```
1  import powerbalance as p
2  # Импорт нашего же модуля
3
4  # Константы их правил
5  buyCost = 5
6  buyCostFast = 10
```

```
sellCost = 2
sellCostFast = 1

#Вычисление стоимости одного исхода энергобаланса
def makeCost(value,adjust):
    result = 0
    if adjust > 0:
        result -= adjust * buyCost
    else:
        result += adjust * sellCost
    diff = value + adjust
    if diff < 0:
        result += diff * sellCostFast
    else:
        result -= diff * buyCostFast
    return result

#Вычисление распределения вероятностей расходов/прибылей
def costBalance(power,adjust):
    return [ (makeCost(v,adjust),p) for (v,p) in power]

#Простой и глупый алгоритм жадного спуска
def greed(a,b,c,x):
    if b < a and b < c:
        return 0
    if b > a:
        return -x
    if b < c:
        return x
    else:
        return -x

#Находим какой-то из минимумов коррекции баланса энергосистемы
def greedilyFindAdjust(net):
    adjStep = 0.1
    adj = 0
    power = p.powerBalance(p.network)
    print('Preparations are complete')
    while True:
        g = greed(costBalance(power,adj-adjStep),costBalance(power,adj),
                  costBalance(power,adj+adjStep),adjStep)
        if g == 0:
            return adj
        else:
            print(adj)
            adj += g

#Напечатать результат
print(greedilyFindAdjust(p.network))
```

Расчёт предельной цены объекта

Эта задача возникает на 2-м и 3-м этапах игры, при участи в аукционе.

Это задача нахождения предельной цены контракта объекта на аукционе — такой цены, приобретение контракта по которой ожидаемо не принесёт ни прибылей, ни убытков. Это расширение над задачей нахождения полного экономического баланса энергосистемы — достаточно вычислить суммарную прогнозируемую прибыль энергосистемы с этим объектом и без него. Для электростанций разницу между этими

величинами нужно разделить на число тактов в игре. Для потребителей — разделить на суммарное потребление его за всю игру (это случайная величина; для примерной оценки её можно заменить на математическое ожидание, но на этом этапе у команд уже должно быть в избытке опыта вычислений со случайными величинами).

Получившееся число: для электростанций — максимальная осмысленная цена, для потребителей — минимальная.

Решение

```
#Импортируем собственный модуль расчёта энергетического баланса
from powerbalance import *

#Импортируем собственный модуль расчёта экономического баланса
from costbalance import *
import operator as o

networkBefore = Network(2,2,1,1,2)
networkAfter = Network(2,2,1,1,3)

def findCumulativeCost(net1,net2):
    before = costBalance(networkBefore)
    after = costBalance(networkAfter)
    powerBefore = powerBalance(networkBefore)
    powerAfter = powerBalance(networkAfter)
    if net1.solar + net1.wind != net2.solar + net2.wind:
        return fuzzyop((fuzzyop(after,before,o.sub)),
                       (fuzzyop(powerBefore,powerAfter,o.sub)),o.div)
    else:
        return fuzzyop(fuzzyop(after,before,o.sub),[(168,1)],o.div)

print(findCumulativeCost(networkBefore,networkAfter))
```

Составление и адаптация стратегии для аукционов

Эта задача возникает на 2-м и 3-м этапах игры, при участи в аукционе.

Хотя команды могут найти оптимальную энергосистему для любого набора прогнозов погоды, эту энергосистему им ещё нужно «отвоевать» у конкурентов. Эта игра в целом относится к рефлексивным, игры этого класса не имеют устойчивого решения, а использование смешанных стратегий едва ли оправдано на одном прогоне игры. Участникам нужно анализировать поведение оппонентов, предугадывать их цели, и искать способы помешать целям оппонентов и защитить свои. Для этого можно создать несколько вспомогательных инструментов:

1. Нахождение эффективных конфигураций энергосистем. Не только оптимальной, но всех, которые достаточно хороши. Все команды будут стремиться к какой-то из них, и для любой промежуточной ситуации на аукционе можно предположить, к какой из них будут стремиться оппоненты.
2. Нахождение оптимальной ставки на аукционе, исходя из ценности лота для себя и оппонентов. Ценности лота для оппонентов можно оценить из предположения об энергосистеме, к которой они стремятся, с использованием решения задачи расчёта прогноза рентабельности. В этих условиях оптимальной ставкой будет максимальная ставка всех оппонентов, при условии, что она не

превышает собственной максимальной ставки. Далее участникам будет интересно от этой ставки отклониться, чтобы рискнуть и увеличить выгоду, либо нарушить стратегию оппонентов (от одного приобретённого по некорректной цене объекта, согласно правилам аукциона, можно избавиться в конце аукциона).

Также в течение всего времени аукциона командам нужно оценивать риск того, что целевую энергосистему составить не удастся, и при необходимости переходить к альтернативным вариантам. Такую задачу решить в общем случае в ограниченное время не представляется возможным, поэтому в ней на первое место выходит смекалка, скорость мышления и скорость принятия решений в команде.

Работа с биржей электроэнергии

Эта задача, тривиальная сама по себе, значительно углубляет задачи вычисления экономического баланса энергосистемы и составления стратегии для аукциона.

Эта задача возникает на 5-м этапе игры, при моделировании энергосистемы.

Таблица цен составлена таким образом, что любое взаимодействие с другим игроком всегда выгоднее его отсутствия (и вынужденного взаимодействия с внешней энергосистемой). Причём более раннее выставление заявки на биржу всегда выгоднее выставление такой же заявки позже.

Таким образом, задача сводится к тому, чтобы делать как можно более ранние закупки на бирже (которые можно распланировать на основании прогнозов даже до начала игры) и их коррекции во время игры на основании действительного состояния энергосистемы (здесь в основном речь идёт о коррекции моделей солнечных батарей и ветряков «на лету»).

Информатика

Система поддержки принятия решений на аукционе

Это первая точка сборки решений предметных задач, возникающих в командном туре.

Основная задача таких систем — вычисление ценности лота при заданных параметрах энергосистемы.

Самый примитивный вариант такой системы может быть устроен следующим образом:

1. Имеются прогнозы погоды и потребления на следующую игру. Мы будем считать, что реальные значения будут точно соответствовать прогнозам.
2. Для каждого такта игры вычисляем генерацию. Например, солнечные батареи вычисляем из яркости солнца с найденным ранее коэффициентом конверсии солнечной энергии в мощность, например, 1 МВт / 651 лк.
3. Вычисляем энергобаланс в каждый такт игры.
4. Вычисляем изменение счёта в каждый такт игры.
5. Добавляем в энергосистему интересующий нас объект.

6. Повторяем шаги 1–4.
7. Сравниваем результаты для энергосистемы при наличии и без наличия интересующего объекта.
   (a) Для электростанций оптимальная цена есть их цена плюс разница результатов энергосистем, делённая на число тактов игры.
   (b) Для потребителей оптимальная цена есть их цена плюс разница результатов игры, делённая на потреблённую потребителем энергию.

Хорошую систему от примитивной могут отличать следующие характеристики:

- Учёт погрешностей прогнозов. Расчёт наихудшего варианта, наилучшего, наиболее вероятного и других статистических характеристик.
- Использование более точных оценок генерации.
- Расчёт стоимости лота для энергосистем оппонентов — чтобы выиграть лот, нужно ставить не собственную цену, а цену, большую, чем у оппонентов. Для этого необходимо оценивать ценность лота для оппонентов.
- Прогноз эффективности задуманной энергосистемы.
- Расчёт величины риска в зависимости от размера ставки — для этого нужно использовать данные об энергосистемах оппонентов и опыт взаимодействия с ними.

Важно, что такая система лишь помогает принимать решения, и не гарантирует того, что команда с наилучшей реализацией этой задачи наиболее эффективно проведёт аукцион.

Разработанные командами программы варьировались по сложности от простых таблиц Excel, до веб-сервисов с элементами ИИ.

Управляющие скрипты

Это вторая точка сборки предметных задач, возникающих в командном туре.

Задача управляющего скрипта — используя возможность управления продажей электроэнергии во внешнюю энергосистему, управления аккумуляторами, прогнозирования генерации, максимизировать число очков, которое получит команда за командный тур.

Если эта задача не решена, то результаты команды будут плохими независимо от результатов аукциона и глубины решения предметных задач командного тура.

Самый простой вариант скрипта может действовать, например, по следующему алгоритму:

1. Включить все отключённые линии
2. Оценить генерацию на следующем такте. Вычислить энергобаланс следующего такта.
3. ЕСЛИ энергобаланс положителен, ПЕРЕЙТИ к п. 6
4. Попытаться ликвидировать дефицит из аккумуляторов.
5. Ликвидировать дефицит из внешней энергосистемы.
6. ЗАВЕРШИТЬ РАБОТУ

7. Попытаться ликвидировать профицит, перенаправив мощность в аккумуляторы.
8. Ликвидировать профицит, продав мощность во внешнюю энергосистему.
9. ЗАВЕРШИТЬ РАБОТУ

Хороший скрипт может обладать следующими характеристиками:

- Оценки энергобаланса на всю игру вперёд и ранняя закупка электроэнергии.
- Коррекция прогнозов генерации на основании реальных данных от электростанций.
- Использование распределения вероятных значений энергобаланса, чтобы вычислять средневзвешенную величину его коррекции: дефицитный энергобаланс обходится дороже профицитного, соответственно нужно минимизировать не модуль энергобаланса, а математическое ожидание экономических потерь в результате ошибок прогнозов.
- Такое управление аккумуляторами, которое полностью разряжает их к последнему такту игры.
- Прогнозирование возможностей аварийных отключений в энергосистеме и использование возможности ограниченного регулирования мощности потребителей и электростанций их избегания.
- Использование возможности ограниченного регулирования мощности потребителей и электростанций для увеличения экономической эффективности энергосистемы (в некоторых ситуациях плата за такое регулирование может быть меньше убытков на полноценное снабжение потребителя или плата за повышение генерации — меньше затрат на закупку мощности).
- Моделирование состояния энергосистем оппонентов для предсказания будущих состояний биржи.
- Уточнение моделей энергосистем оппонентов на основании реального состояния биржи.
- Управление риском: в ряде случаев осмысленно использование неоптимальных характеристик управления, которые несмотря на то, что они снижают математическое ожидание счёта в командном этапе, увеличивают вероятность обойти другую команду.

Решение

```
import ips
psm = ips.init()

def average_weather(type, turn):
    if type == 'sun':
        q = psm.sun[turn].forecast
        return (0.25 * (q.lower0 * 0.5 + q.lower50 + q.median + q.upper50
            + q.upper0 * 0.5))
    else:
        q = psm.wind[turn].forecast
        return (0.25 * (q.lower0 * 0.5 + q.lower50 + q.median + q.upper50
            + q.upper0 * 0.5))

main_station = list(psm.stations.keys())
```

```
15  buffer = ips.buffer()
16  turn = psm.get_move()
17  clients = psm.powersystem.get_all_clients()
18  first_turn = 1
19  last_turn = 50
20  next_turn = turn + 1
21
22  def wind_koeff(turn):
23      return 0.9
24
25  sun_koeff = 1  # Зависимость мощности панелей от солнца
26  current_consumption = 0  # Текущее производство
27  current_production = 0  # Текущее потребление
28  panels = 1  # Кол-во панелей
29  turbines = 0  # Кол-во турбин
30  trade10_koeff = 1  # Какую часть средней разницы энергии
31  #                    отправлять на биржу через 10 ходов
32  comments = True  # Включены ли комментарии выполненных скриптом действий
33  ips.set_order_trace(True)
34  average_estimated_production = [0] * 16
35  average_estimated_consumption = [0] * 16
36
37  def est_cons(turn):
38      cons = 0
39      clients = psm.powersystem.get_all_clients()
40      for client in clients:
41          if client.is_consumer():
42              cons += ((client.preset[turn][0] + client.preset[turn][1]) / 2)
43      for offer in psm.exchange:
44          if offer.owner == psm.you:
45              if offer.issued + offer.exchange == turn:
46                  if offer.amount > 0:
47                      cons += offer.amount
48      return cons
49
50  def est_prod(turn):
51      prod = 0
52      prod += (wind_koeff(turn) * average_weather('wind', turn) * turbines)
53      prod += (sun_koeff * average_weather('sun', turn) * panels)
54      for offer in psm.exchange:
55          if offer.owner == psm.you:
56              if offer.issued + offer.exchange == turn:
57                  if offer.amount < 0:
58                      prod -= offer.amount
59      return prod
60
61  for client in clients:
62      if client.is_consumer():
63          current_consumption += abs(client.power[-1])
64      else:
65          current_production += abs(client.power[-1])
66
67  for offer in psm.exchange:
68      if offer.owner == psm.you:
69          if offer.issued + offer.exchange == turn:
70              if offer.amount > 0:
71                  current_consumption += offer.amount
72              else:
73                  current_production -= offer.amount
74
```

```
75  average_estimated_consumption[0] = current_consumption
76  average_estimated_production[0] = current_production
77  for i in range(1, 16):
78      average_estimated_production[i] = 1
79      average_estimated_consumption[i] = 1
80
81  if turn < last_turn - 10:
82      shortage10 = average_estimated_consumption[10] - average_estimated_production[10]
83      if shortage10 > 0:
84          psm.orders.trade10.buy(shortage10)
85          print('buying in 10 turns', shortage10)
86      elif shortage10 < 0:
87          psm.orders.trade10.sell(-shortage10)
88          print('selling in 10 turns', -shortage10)
89
90  for client in clients:
91      if client.is_generator():
92          if 'solar' in str(client):
93              solar_out = client.power[-1]
94          # print(client.addr, 'generating', client.power[-1])
95      elif client.is_consumer():
96          pass
97          # print(client.addr, 'consuming', client.power[-1])
98      else:
99          pass
100         # print('unknown', client.addr)
101 if solar_out > 0:
102     real_sun_k = psm.sun[turn].value / solar_out
103     print('sun koeff =', real_sun_k)
104
105 if last_turn - turn > 17:
106     z = 16
107 else:
108     z = last_turn - turn - 2
109 for i in range(1, z):
110     shortage = average_estimated_consumption[i] - average_estimated_production[i]
111     if shortage > 25:
112         print('critical: in', i, 'turns shortage', shortage, 'MBт',
113               'order already placed for')
114     elif shortage < -25:
115         print('critical: in', i, 'turns overgeneration', -shortage, 'MBт')
116     elif shortage > 15:
117         print('warning: in', i, 'turns shortage', shortage, 'MBт')
118     elif shortage < -15:
119         print('warning: in', i, 'turns overgeneration', -shortage, 'MBт')
120 if last_turn - turn > 5:
121     z = 4
122 else:
123     z = 0
124 for i in range(4):
125     shortage = average_estimated_consumption[i] - average_estimated_production[i]
126     if shortage > 0:
127         print('In', i, 'turns shortage', shortage, 'MBт')
128     else:
129         print('In', i, 'turns overgeneration', -shortage, 'MBт')
130
131 for offer in psm.exchange:
132     if offer.owner == psm.you:
133         if offer.amount > 0:
134             print(offer.amount, 'will be bought in',
```

```
                offer.issued - offer.exchange - turn, 'turn(s)')
        else:
            print(offer.amount, 'will be sold in',
                offer.issued - offer.exchange - turn, 'turn(s)')
```

Система оценки

Игровые очки, автоматически начисляемые в экономической модели игры, являются интегральной оценкой всех логистических, энергетических, физических, математических решений принятых командой. В финальном подсчете для команд-победителей полуфиналов турнира игровые очки переводятся в баллы от 0 до 100. Команда, набравшая максимальное число очков, получает 100 баллов (и становится командой-победителем), команда, набравшая минимальное число очков — 0 баллов. Остальные команды, получают баллы, нормированные на эти два результата по формуле

$$S = 100 \times \frac{x - MIN}{MAX - MIN},$$

где x — число внутриигровых очков, набранных командой, MAX — число очков, набранное в финале командой-победителем, MIN — число очков, набранное командой, занявшее в финале последнее место. Таким образом, команда-победитель финала получает всегда 100 баллов, команда, занявшая в финале последнее место — всегда 0 баллов, остальные — от 0 до 100, пропорционально показанным ими результатам.

5.2. Приложения к задачам

Приложение 1. Детали правил

Прогнозы

Все прогнозы характеризуются пятью числами, которые задают квартили распределения.

У каждого объекта прогноз свой.

Все прогнозы, кроме ветра, строятся как отклонения от паттерна. Прогнозы всегда умещаются в коридор от 80 до 120

Паттерн солнца (повторяется периодически с начала игры):

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1.6, 2.5, 3, 3.7, 4.2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 14.5, 14.8, 15, 15, 15, 15, 14.1, 13.0, 11, 10.5, 9.8, 9, 8, 7, 6, 5, 4.3, 2, 1.2, 0.4

Паттерн больниц:

0.6, 0.6, 0.5, 0.4 , 0.3, 0.3, 0.4, 0.5 , 0.5, 0.7, 0.8, 1.0 , 1.1, 1.2, 1.4, 1.5 , 1.6, 1.7, 1.6, 1.6 , 1.5, 1.5, 1.4, 1.4 , 1.3, 1.3, 1.2, 1.2 , 1.2, 1.1, 1.1, 1.1 , 1.2, 1.2, 1.1, 1.0 , 1.0, 0.9, 0.8, 0.7 , 0.7, 0.7, 0.6, 0.6 , 0.5, 0.4, 0.4, 0.5

Паттерн заводов:

4.2, 4.3, 4.4, 4.5 , 4.6, 4.6, 4.7, 4.8 , 4.9, 4.9, 5.0, 5.1 , 5.3, 5.6, 5.9, 6.2 , 6.3, 6.3, 6.4, 6.3 , 6.5, 6.8, 6.9, 7.0 , 7.4, 8.2, 8.9, 8.8 , 8.7, 8.6, 8.8, 9.0 , 9.1, 9.0, 8.9, 8.8 , 8.0, 7.4, 7.0, 6.6 , 6.3, 6.0, 5.8, 5.2 , 5.0, 4.4, 4.1, 4.0

Паттерн домов:

1.5, 1.1, 0.8, 0.8 , 0.7, 0.7, 0.7, 0.8 , 0.9, 0.9, 1.0, 1.1 , 1.1, 1.2, 1.3, 1.4 , 1.5, 1.6, 1.8, 2.0 , 2.1, 2.1, 2.0, 1.7 , 1.4, 1.3, 1.3, 1.2 , 1.1, 1.2, 1.3, 1.5 , 1.6, 1.9, 2.1, 2.3 , 2.7, 2.9, 3.1, 3.7 , 4.2, 4.4, 4.3, 4.0 , 3.3, 2.8, 2.4, 2.1

Отключения

Предельная мощность подключения к внешней сети ограничена и составляет 30 МВт

При перегрузке подключения к внешней сети отключаются ветки главной подстанции в порядке 3, 2, 1, пока сеть не сможет автоматически сбалансироваться.

Предельная мощность, проходящая через узел подстанции, ограничена и составляет 20 МВт

Установка трансформатора увеличивает мощность внешнего подключения подстанции на 15 МВт

Потребителю можно отдать приказ на сокращение потребление, до 25

Электростанции можно отдать приказ на сокращение или увеличение генерации. За сокращение генерации игрок дополнительно 1

За отключение объекта также платится пропорциональный штраф: пятикратную стоимость МВт*такт для потребителей 1-й категогии, двукратную для 2-й и однократную для 3-й

Обслуживание отключённой электростанции обходится в 2,5 раза дороже работающей.

Максимальная мощность солнечной батареи составляет 15 МВт.

Максимальная мощность ветряка составляет 15 МВт.

Скрипты

Суммарное время работы скриптов ограничено тремя секундами. Обратите внимание, что получение данных стенда скриптом занимает 0.25 секунды.

Аккумуляторы

Ёмкость аккумулятора составляет 25 МВт, максимальная скорость заряда — 5 МВт*такт, максимальная скорость разряда — 5 МВт*такт

Биржа

Взаимодействие со внешней энергосистемой осуществляется через биржу. Энергия покупается/продаётся из внешней энергосистемы только при невозможности сде-

лать это через других игроков.

Поставки энергии между игроками гарантируются внешней энергосистемой.

Существует три типа биржи, на 0, на 1, на 3 и на 10 ходов вперёд.

Цена, по которой игроки торгуются энергией, зависит от того, на какой бирже купил её покупатель, и на какой продал продавец. Цена будет в пользу того, кто выставил заявку на более ранней бирже. Таблица цен:

Покупатель →	0	1	3	10	Внешняя ЭС
0	3.5	3	2.5	2	1
1	4	3.5	3	2.5	2
3	4.5	4	3.5	3	2.5
10	5	4.5	4	3.5	3
Внешняя ЭС	7	5	4.5	4	—

При наличии одной заявки на покупку и многих на продажу, от каждой заявки на продажу берётся одинаковый процент, и наоборот.

Работа на бирже проходит в автоматическом режиме. Игроки могут только составлять

Приложение 2. Справка по системе управления скриптами

Краткий принцип работы

1. Есть система управления скриптами (далее СУС).
2. СУС имеет именованные слоты для хранения управляющих скриптов.
3. Скрипт пишется на языке Python версии 3.6 и, помимо стандартных, использует специальную библиотеку для энергостенда.
4. Есть назначенная очередь выполнения, указывающая, какие скрипты и в каком порядке должны выполняться в ближайший ход.
5. В начале каждого хода помеченные на запуск скрипты перемещаются в текущую очередь и выполняются по порядку до тех пор, как очередь не закончится.
6. Скрипт может получить данные со стенда на текущий ход и отправлять на него управляющие приказы. Это основная задача управляющего скрипта и СУС в целом.
7. Скрипт может ставить в очередь другой скрипт (по его имени). Он выполнится после всех остальных.
8. Скрипт может запускать другой как дочерний, передавая ему на вход произвольные данные в JSON. При этом родительский скрипт блокируется до завершения всех дочерних. Дочерний скрипт может вернуть данные в JSON, они передаются в родительский после разблокировки.
9. Количество выполнений для каждого скрипта ограничено в течение хода и равно 100. Запуск скрипта с синтаксической ошибкой также уменьшает счётчик.

10. Общее время выполнения всех скриптов ограничено глобальным таймаутом. При его достижении все запущенные скрипты «убиваются», а дальнейший запуск запрещается до начала нового хода.
11. Количество вложенных дочерних скриптов тоже ограничено и равно 100. Но из-за пункта 9 это не будет проблемой.
12. Скрипт может узнать текущую очередь, количество оставшихся выполнений для любого скрипта и момент глобального таймаута (штамп времени).

Базовый пример

```
import ips # 1
psm = ips.init() # 2
clients = psm.powersystem.get_all_clients() # 3
consumption = 0 # прогноз суммарного потребления
generation = 0 # прогноз суммарной генерации
for client in clients:
if client.is_generator():
generation += client.power[-1]
else:
consumption += client.power[-1]
shortage = consumption - generation
if shortage > 0: psm.orders.trade0.buy(shortage) # 4
else: psm.orders.trade0.sell(-shortage) # 4
```

1. Импорт библиотеки API стенда.
2. Запрос на сервер и создание объекта с данными системы на текущий ход.
3. Данные извлекаются путём вызова методов и функций, см. далее.
4. Вызываемые приказы сразу же отправляются на сервер и отражаются в интерфейсе.

Получение данных

Прежде всего:

```
import ips # импортируем библиотеку
# при работе с тестовой версией нужно задать данные вручную
# например, из файла (другие способы будут ниже)
ips.debug_psm_file("state.json") # убедитесь, что файл лежит рядом со скриптом
psm = ips.init() # читаем состояние энергостенда в объект
# и продолжаем работать
```

Погода

```
# их структура одинакова и представляет собой список,
# каждый элемент которого соответствует своему ходу
print(psm.sun) # значения яркости солнца
print(psm.wind) # значения скорости ветра
# элемент состоит из двух полей:
# - реальное значение, которое может быть пусто (ход не наступил)
# - прогноз в виде процентилей
print(psm.wind[0]) # объект-пара для ветра за первый ход
print(psm.wind[0].value) # либо реальное значение, либо None
```

```
    print(psm.wind[0].forecast) # прогноз в виде набора процентилей
    # актуальное значение погоды берётся по номеру хода
    # для извлечения номера хода можно использовать это:
    tick = psm.get_move() # рассмотрим поля для отдельного взятого прогноза
    q = psm.wind[tick].forecast print(q.lower0) # 0%
    print(q.lower50) # 25%
    print(q.median) # 50%
    print(q.upper50) # 75%
    print(q.upper0) # 100%
    # вспомогательные кортежи с парами значений
    print(q.quart1) # Q1 (0%, 25%)
    print(q.quart2) # Q2 (25%, 50%)
    print(q.quart3) # Q3 (50%, 75%)
    print(q.quart4) # Q4 (75%, 100%)
    print(q.spread) # (0%, 100%)
    print(q.spread50) # (25%, 75%)
```

Биржа

```
    print(psm.exchange) # все предложения на всех биржах
    print(psm.exchange[0]) # конкретное предложение

    # для отдельно взятого предложения
    print(psm.exchange[0].exchange) # тип биржи (через сколько ходов)
    print(psm.exchange[0].amount) # объём заявки
    print(psm.exchange[0].issued) # номер хода, на котором она создано
    print(psm.exchange[0].owner) # контрагент
```

Подстанции

```
    print(psm.stations) # все подстанции в виде dict
    print(list(psm.stations.keys())) # адреса подстанций

    # возьмём главную подстанцию M0 (у вас будет другой адрес)
    print(psm.stations["M0"]) # объект главной подстанции
    print(psm.stations["M0"].addr) # адрес
    print(psm.stations["M0"].cells) # кол-во аккумуляторов
    print(psm.stations["M0"].transformers) # кол-во трансформаторов
    print(psm.stations["M0"].charge) # общий заряд аккумуляторов
    print(psm.stations["M0"].modules) # список модулей как объектов
    print(psm.stations["M0"].modules[0].is_cell) # проверка типа модуля

    # для аккумулятора можно получить
    # динамический список со значениями заряда за этот и предыдущие ходы
    print(psm.stations["M0"].modules[0].charge) # аналогичный формат значений
    #используется и ниже

    # ВНИМАНИЕ! значение за последний ход - конец списка (charge[-1])
    # в начале игры некоторые дин. списки пусты, это эквивалент нуля

    # возьмём произвольную миниподстанцию m5 (у вас их может и не быть)
    # print(psm.stations["m5"]) # миниподстанция
    print(psm.stations["m5"].addr) # адрес миниподстанции
```

Энергосеть

```
print(psm.powersystem) # объект энергосети

print(psm.powersystem.get_all_lines()) # извлечь все имеющиеся линии
print(psm.powersystem.get_all_clients()) # всех имеющихся клиентов
print(psm.powersystem.get_all_regions()) # все имеющиеся энергорайоны
print(psm.powersystem.line("MO", 1)) # энергорайон на первой линии M0

# в системе линия представлена парой (адрес_подстанции, номер)
print(psm.powersystem.lines()) # получить все энергорайоны
key_list = [("MO", 1), ("MO", 2)]
print(psm.powersystem.lines(key_list)) # энергорайоны по ключам в списке
# эти две функции возвращают массив пар линия-район
# возьмём отдельный энергорайон
region = psm.powersystem.line("MO", 1)
print(region.get_all_lines()) # аналогичны функциям выше, но выдаются
print(region.get_all_clients()) # элементы из себя и соседних районов
print(region.get_all_regions()) #
print(region.online) #
print(region.lines) # линии к соседним энергорайонам (список)
print(region.clients) # клиенты в данном энергорайоне (список)

# возьмём отдельного клиента
client = region.clients[0]
print(client.is_consumer()) # ответ на вопрос "это потребитель?"
print(client.is_generator()) # или "это генератор?"
print(client.addr) # адрес объекта
print(client.contract) # тариф по контракту
print(client.profits) # доход за каждый ход (дин. список)
print(client.losses) # расход за каждый ход (дин. список)
print(client.power) # текущая мощность (дин. список)
# она же реальное потребление-генерация
print(client.influence) # влияние на объект (дин. список)
# если клиент - потребитель
print(client.preset) # список значений и прогнозов аналогично погоде
# это необходимо в основном для работы с прогнозами,
# для чтения текущих значений удобнее применять power
```

Прочая информация

```
print(psm.you) # ваш индекс игрока, пара чисел стенд-терминал
# не путать с адресом своей подстанции!
print(psm.score) # счёт на текущий момент (динамческий список)
print(psm.get_move()) # определить номер текущего хода (первый ход = 1)
print(psm.humanize()) # полное описание состояния стенда (отладка)
```

Приказы

Существуют в двух форматах:

- строгие приказы, отправляются сразу

  `psm.orders`

- ленивые приказы, сохраняются в виде просматриваемого объекта

  `psm.orders_lazy`

Есть два способа отправки ленивых приказов:

- вызов метода от объекта приказа

```
lazy_order.send()
```

- пакетная отправка нескольких приказов

```
ips.send_orders([order1, order2])
```

Приказы обрабатываются в порядке их отправки в систему!

Все приказы на биржу учитываются и применяются отдельно!

Для остальных приказов приоритет — у последнего вызванного!

```
### Линии (выполняется последний отправленный)
psm.orders.line_on("M0", 1) # включение линии 1 подстанции M0
psm.orders.line_off("M0", 1) # выключение этой же линии
print(psm.orders_lazy.line_on("M0", 1).addr) # адрес подстанции
print(psm.orders_lazy.line_on("M0", 1).line) # линия
print(psm.orders_lazy.line_on("M0", 1).value) # устанавливаемое значение

#### Биржа (накапливается)
psm.orders.trade1.buy(5) # купить 5 МВт-ч на следующий ход
psm.orders.trade1.sell(5) # продать 5 МВт-ч на следующий ход
psm.orders_lazy.trade3.buy(5) # ... через 3 хода
psm.orders_lazy.trade10.buy(5) # ... через 10 ходов
print(psm.orders_lazy.trade0.buy(5).exchange) # тип биржи (число ходов)
print(psm.orders_lazy.trade0.buy(5).value) # объём купли-продажи

### Аккумуляторы (выполняется последний отправленный)
# В качестве адреса указывается адрес подстанции!
psm.orders.cell_charge("M0", 1) # зарядить на 1 МВт-ч
psm.orders.cell_discharge("M0", 1) # разрядить на 1 МВт-ч
print(psm.orders_lazy.cell_charge("M0", 1).addr) # адрес
print(psm.orders_lazy.cell_charge("M0", 1).power) # мощность

### Влияние на объекты (выполняется последний отправленный)
# В качестве адреса указывается адрес объекта!
psm.orders.influence("h0", 1) # установить влияние на h0 в значение 1
print(psm.orders_lazy.influence("h0", 1).addr) # адрес объекта
print(psm.orders_lazy.influence("h0", 1).value) # значение
```

Запуск других скриптов

Есть два способа вызвать другой скрипт (или самого себя). Команда enqueue добавляет скрипт в конец очереди, т.е. он будет выполнен после завершения остальных скриптов. ips.enqueue("eta") Также можно выполнять скрипты как функции с входными и выходными данными:

```
####### РОДИТЕЛЬСКИЙ СКРИПТ #######
# Пусть у нас есть набор данных для передачи.
data = ["horse", "chair", 28]
# В дочерний скрипт передаётся data,
# скрипт блокируется до завершения дочернего.
# Вывод дочернего скрипта запишется в output
output = ips.call("beta", data)

```

```
    ####### ДОЧЕРНИЙ СКРИПТ #######
    # Чтобы записать выходные данные, в дочернем скрипте пишется команда:
    ips.exit_with(["horse", "chair", 28])
    # После её выполнения скрипт завершится, вернув управление родителю.
```

Вспомогательные команды

```
    ips.send_orders(lazy_order_list) # массовая отправка ленивых приказов
    ips.get_launches("alpha") # получить число оставшихся запусков для скрипта
    ips.get_timeout() # получить UNIX-timestamp полного завершения выполнения
    ips.get_queue() # получить текущую очередь выполнения
    ips.set_order_trace(True) # включить/выключить вывод приказов в консоль
```

Отладка

В целях упрощения отладки представлены функции для установки заглушек. Они работают исключительно в тестовой версии фреймворка, при выполнении в СУС данные вызовы лишь пишут предупреждение в поток ошибок и пропускаются.

```
    # установка данных энергостенда, перед ips.init()
    ips.debug_psm_file("state.json") # из файла
    ips.debug_psm_file("<...>") # из строкового объекта
    # установка входных данных, перед ips.buffer()
    ips.debug_buffer_file("buffer.json") # аналогично
    ips.debug_buffer_json("<...>") #
    ips.debug_launches(99) # число запусков для get_launches
    ips.debug_timeout(time.time() + 10) # таймстамп для get_timeout
    # О, привет. Раз уж ты здесь, рекомендую почитать исходники библиотеки.
    # В них есть полезные вещи для боевого применения и, возможно, ошибки.
    # Найдёшь последние - дай знать преподавателям, они передадут.
```

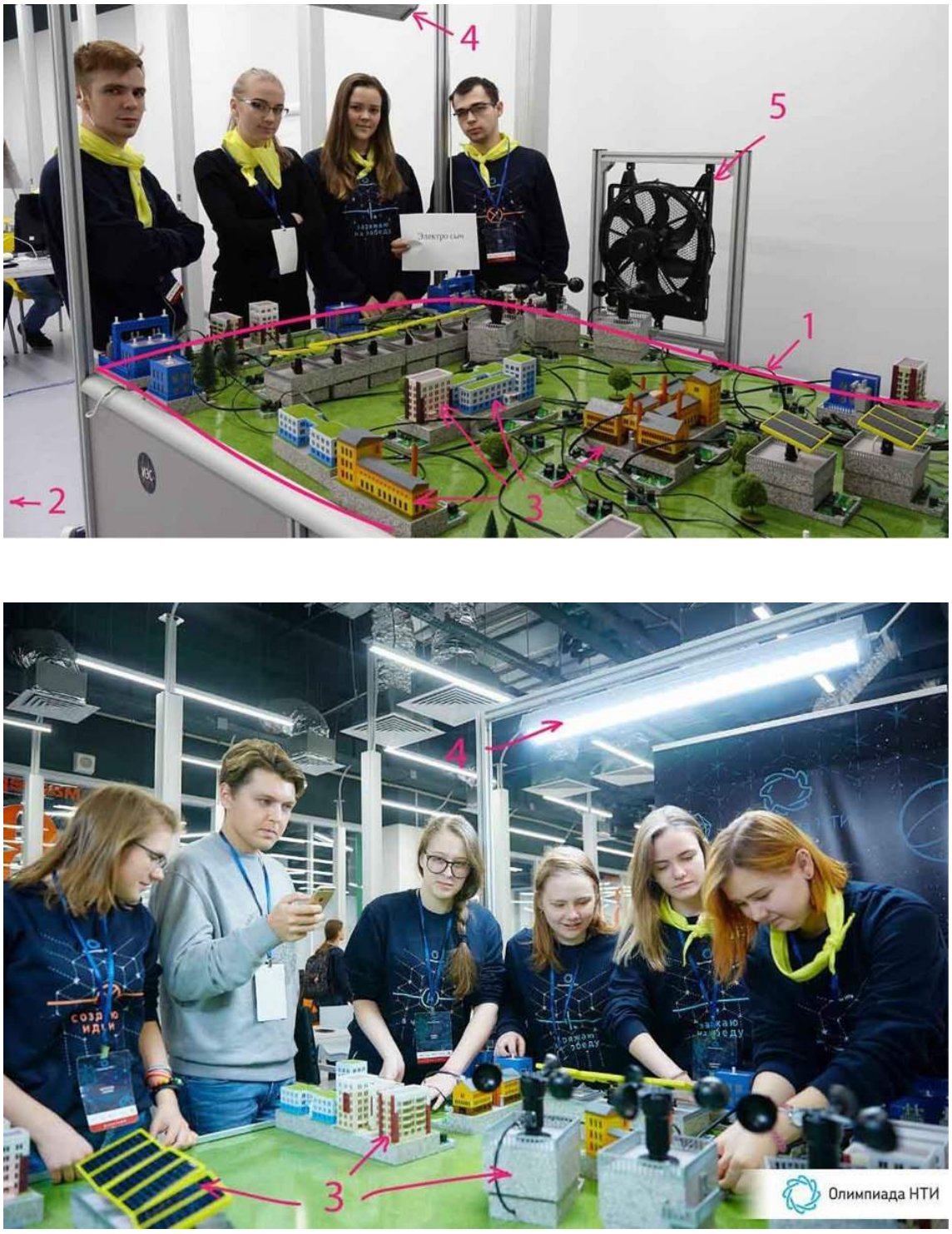
Приложение 3. Устройство Стенд-тренажер “ИЭС” 2018-2019 года

Стенд-тренажер “ИЭС” 2018-2019 года состоит из следующих основных частей:

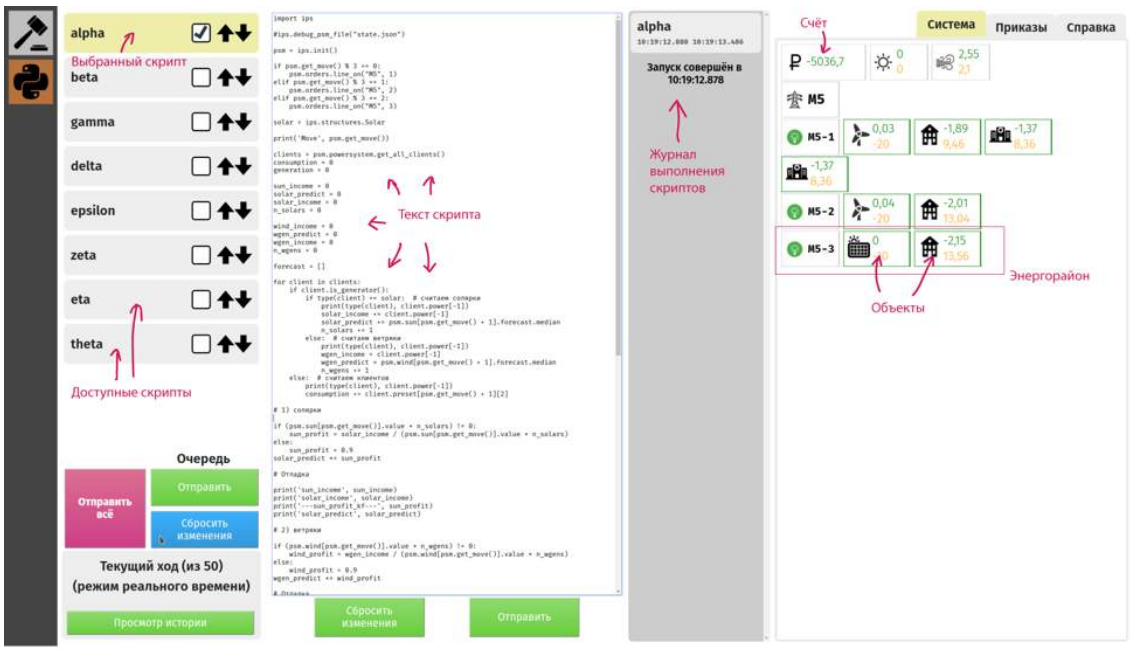
1. Модельная поверхность, на которой располагаются модели объектов энергосистемы: электростанций, потребителей и объектов энергетической инфраструктуры.
2. Терминалы управления энергосистемой. Это персональные компьютеры, объединённые со стендом в единую информационную сеть. Каждая команда работает со своим терминалом.
3. Модели объектов энергосистемы. Это небольшие стереотипные архитектурные модели, содержащие в себе необходимую управляющую электронику и измерительные системы.
4. Светильники, моделирующие солнечное освещение. Они способны создавать освещённость в центре стола до 5 клк.
5. Мощный вентилятор, моделирующий ветровые условия. Он способен создавать ветер со скоростью до 5 м/с на расстоянии не менее 1 метра от плоскости

вращения.

6. Модули расширения главной подстанции — аккумуляторы и трансформаторы.



Фотографии стенда



Скриншот пользовательского ПО стенда (с комментариями)