电气工程导论 C 大作业实验报告

莫名其妙小组

January 13, 2018

1 项目介绍

我们组的项目是Momie ，一个 steam 游戏搜索引擎。

1.1 项目目的

确定这个项目是因为现有的游戏搜索引擎有两个巨大的缺陷：

(1) 不支持中文名和别名搜索，仅支持英文官方名称搜索。

实际上很多时候我们并不能完全记下来如 Getting Over It with Bennett Foddy，或是《和班尼特福迪一起攻克难关》这样拗口的名字，我们称呼一个游戏更多的还是用它的民间通称，如《抡大锤》，或《掘地求升》等。这给国人的游戏的搜索带来了很大的不便。

(2) 无法进行模糊查找，无法对游戏内容进行搜索。

另外一个问题是，很多时候我们并不确切地知道自己想要找的游戏。比如，有时候人们突然想找一个设计机械的游戏来游玩，这时候模糊搜索就能为他们提供很多选择。此外，有时候还会发生这样的事情，人们在看别人游戏的时候发现了一款很对自己胃口的游戏，但是并不知道那是哪款游戏，这个时候如果能通过对游戏剧情、内容的搜索，就可以找到自己想要的游戏。

因此，我们想要建立一个功能强大的搜索引擎，方便玩家们进行游戏搜索。

1.2 项目功能

基础功能：

- 游戏的文本搜索和图片搜索
- 游戏信息整合及呈现（游戏名，发行商，标签，评论等）

拓展功能：

- 按游戏公司搜索游戏
- 输入自动补全及纠错
- 游戏中英文名称互换

- 游戏别名识别
- 游戏相似度（相关游戏）
- 游戏评分（画面，剧情等多方面评分）

1.3 项目特色

(1) 我们的文字搜索可以选择三个方面（company content 和 title）利用一个文字搜索框，不需要用户使用 bool 运算等搜索，自动利用一定按钮进行在多个域中找出最好的结果，界面易懂友好。

(2) 我们整合了各种评论信息并对其进行 sentiment analysis，可以根据用户的喜好改变排序。

(3) 我们进行了全新的游戏相关度计算，同时考虑了名称、标签、评论、发行商四方面因素，更加全面而符合用户的需求。

(4) 我们由于整合互联网上了大量信息，所以搜索时可以更好地确定用户描述的是哪一款游戏，比如可以输入网友对游戏的俗称而搜索游戏，这一点现在是其它游戏搜索网站所做不到的。

(5) 拥有一个具有一定容错功能的搜索补全，可以方便用户使用。搜索补全的容错功能是对中文的同字不同音容错，对英文的单个单词拼错几个字母容错。拥有一个拼写纠错，可以在用户输入的 query 与现成的有很略微的差别时提醒“你要找的是不是. . . . ”

(6) 我们可以实现游戏中英文名称的自动转换以及游戏别名的识别

(7) 我们的界面简洁而不失特色，方便而不失丰富，能给予用户良好的使用体验

(8) 组组员合作高度分工，合作多使用 json 和 pickle 文件在小组组员之间批量传输数据，进行像流水线一样的加工处理。小组组员处理大批量的数据时使用了记录 log 文件的方式，记录个别出错的文件特殊处理，使得大家的工作更加的鲁棒，不至于牵一发而动全身。

2 团队分工

石哲宇，516030910532：

- 爬虫
- 图片搜索
- 中文名获取

杨晨宇，516030910537：

- 索引
- NLP(情感分析，别名识别)
- 输入辅助功能

姚青山，516030910539：

- 游戏相似度
- 猜你喜欢

赵星，516030910540：

- Web 前后端
- 整合

温涵博，516030910535：

- parse

3 各模块具体功能及实现

1. 爬虫部分

1. 爬取的内容

我们爬取了以下内容：

1. steam 商店页面
2. steam 社区及用户评论
3. 百度搜索内容
4. 新浪新闻

2. 遇到的问题

由于这些网站比较友善，对访问频率和访问内容都没有特别严格的限制，爬取过程中没有遇到太多阻碍。只有两点小的方面需要提到：

1. steam 商店有一些游戏存在验证，如年龄验证，“不适合工作时浏览的内容”验证。用 chrome 浏览器访问，然后用开发者工具抓包，可以得到请求内容，找到与年龄和内容限制相关的字段，将其加入爬虫的 head 或 cookies 中，便可以绕过验证。
2. 爬取 steam 评论的过程时，steam 社区提供了访问用户评论的api。然而它的文档页面提供的参数列表是有错误的，其中的 language 字段实际上是无效的，不论怎么调整，都无法返回中文评论内容。为此查询了国内外各大论坛，甚至都没有找到相关问题。我们甚至给 steam 客服发了邮件，希望他们更正这个错误，然而并没有得到回复，因此最后只好独立解决。我们猜测整个 steam 社区的通信格式是一样的，通过观察社区其他页面的请求中关于语言方面的字段，并将其逐个加入爬虫的请求中进行尝试，最后成功找到了正确的请求字段及内容，得到了中文评论内容。

3. 数据持久化，间断爬取

另外，由于我们的数据比较大的单元数量也在 3 万以上，比较小的单位数量超过三十万，对这样的内容进行爬取，必须要考虑间断爬取。为了解决这个问题，我们使用 pickle 将对象持久化，每爬取到一定数量后便将数据 dump 到硬盘中一次，提高了鲁棒性和安全性，同时也避免了重复爬取，提高了爬取的效率。

2. 图像搜索部分

我们的搜索引擎支持通过游戏截图搜索游戏。

1. 基本思路

采用 LSH 的方法，对从各大网站爬取到的截图建立哈希，对初步筛选出的相似图片再对比特征计算距离，排序后便得到最相似的一些结果。

2. 实现

由于一共爬取到 30 多万张图片，对这样规模的数据建立哈希，如果单纯用实验课上的方法是无法完成的。原因如下:

1. 为这么多图片建立 hash，时间上无法承受。

2. 二是 hash 值的空间太小，将会有非常多的图片在同一个 basket 里，查询速度也会很慢。

为了解决以上问题，我们对算法进行了两点改进

1. 将统计颜色直方图，改为用缩略图替代统计信息

在统计颜色直方图信息时我们发现，尽管正确使用 Numpy，也无法为统计过程加速。由于数据量巨大，这样的时间代价我们时无法接受的。另外，在统计颜色信息时，我们实际上并不十分关心图片的细节，因此需要一种更快速的方法计算统计信息。能想到朴素方法的是进行抽样和插值，但是由于 python 本身执行效率并不高，因此手写之后发现并无太大效果，很难保证效果和时间都可以接受。然而在 SIFT 算法的实验中，用到了 OpenCV 自带的 resize 方法。经查阅资料发现，resize 方法使用的是比较优秀的插值方法，而且默认用 C++ Concurrency 进行优化加速，因此采用该方法来提升速度。将图片 resize 为合适的大小，用 resize 后的结果代替对每个块的统计结果。

2. 加入梯度直方图信息

由于我们应对的所有图片都是游戏截图，它们作为生成图片，与真实照片在梯度上有很大区别，相比照片大量为正态分布，生成图片在梯度分布上有更大的多样性。因此考虑将梯度直方图信息加入 hash 的特征。

为提升速度，首先用上一步的方法将图片缩小为合适的大小，然后用 sobel 算子统计梯度直方图。再将梯度直方图的横坐标合并，然后用类似颜色直方图的方法建立特征向量，得到梯度直方图的 LSH 特征。

最后，将颜色直方图得到的特征向量和梯度直方图得到的特征向量拼接，得到最终的特征向量。

进行改进后，对所有 30 多万张图片建立 LSH 集只需要三个小时以内，同时也方便对参数进行调试。

3. Index and Search 部分

Index 部分

- **gameindexer.py**-index 部分的核心。写了 Indexfiles 类，支持了 index 信息的批量索引，查重，批量修改，删除，增加信息等等功能，方便后面使用。
- **gameindexpusher.py**-index 部分的操作脚本。用于把各种批量信息解码处理并且添加到 index 中。本次实验加到 index 中的信息有：通过规则解析的中文名字、通过马尔科夫挖掘猜出的中文别称，通过 sentiment analysis 算出的每个游戏的分值，计算的游戏相关度的信息，所有的 steam 上的用户评论
- **txt2json.py**-用于把 parse 出来的游戏信息转化为 json 这种有结构的文件格式，方便 gameindexer 批量索引
- **gameindexpusherwin.py**-是因为最后期末复习时间，时间紧迫，本人多台电脑必须同时运行，为了节约时间，写了适合在 Windows python3 平台下的 indexpusher

search 部分

- **gamesearcher.py**-search 的核心，里面的内容被前端调用，支持通过 appid 获得游戏所有的信息，通过关键词搜索。其中 keywordsearch 函数可以选择不同的模式，根据前端的参数调整不同的排序搜索方式。
- **querycorrection.py**-为前端提供“你要找的是不是?”功能，算法使用 lsh，在不同的哈希方式下得到可能相似的搜索，然后逐一进行编辑距离的计算。目前只适合对英文输入检查拼写。
- **queryguesser.py**-为前端提供搜索补全功能，算法使用排序二分查找。具有对中文的同字不同音容错，对英文的单个单词拼错几个字母容错的特色。

  利用 python 模块：pypinyin

sentiment analysis 部分

利用现有成果：

Stanford nlp parser

一个 java 程序，可以分析输入的文件的句法结构，得到输出文件（只找到了单独运行的方法，不能被 python 调用）

gensim （word2vec）词向量可以比较两个不同词汇之间的相似度

下载了网友分享的自己训练的词向量数据https://www.cnblogs.com/Darwin2000/p/5786984.html

sentimentscore.txt

下载的情感词典，储存着各种词汇表达的褒义贬义情感值

BosonNLP 情感词典https://bosonnlp.com/dev/resource

- **nlpenfile.py**-因为尝试使用 python 或者 java 调用 Stanford nlp parser 的方法失败了，所以只能把需要处理的评论信息变成文件交给 parser 处理。

  又因为 parser 每一次处理一个文件必须重新加载一次，需要 15 秒左右，而远超过处理一个游戏的 review 文字所需时间。所以只好有这个无奈之举，把所有的游戏的 review

文字放到一个文件中，让 parser 整体处理。同时加入一些特殊的文字用来区分不同游戏的 reviews。

- **nlpdefile.py**-和 nlpenfile 同属于伺候 parser 的脚本程序。在 parser 解析出所有的句子结构之后，得到 txt 文件，处理这个 txt 文件，提取我们需要的 nsubj、amod、dobj 词汇对。同时区分哪一些是属于哪一个游戏的，输出 json 这种具有结构的文件。
- **nlpvector.py**-分析 nlpdefile.py 得到的 json 文件，利用 gensim 和 sentimentscore 对每一个游戏进行 sentiment analysis。
- **nlpvectorwindows.py**-同样是因为考试周时间安排紧张，本人大电脑性能较好（windows python3）因此在大电脑上也分配了计算的任务。

索引信息架构说明

- **name**

  用于存放爬取到的英文名和通过规则识别从百度搜索结果中找到的中文名

  英文名在首次构建时从基础的 steam 网页信息批量索引进入

  中文名在分析百度结果之后使用 gameindexpusher.py 添加进 name 域

- **names**

  存储通过马尔科夫模型猜到的别名，用于搜索。

  在挖掘出别名之后通过 gameindexerpusher.py 添加进 names 域

- **id**

  是一个 lucene int 型的域，是每一个游戏的唯一标识，对索引的操作使用这个 id 来确定目标

  在首次构建索引时存入

- **tags**

  爬取的游戏的标签，用于搜索和相似比较

  在首次构建时从基础的 steam 网页信息批量索引进入

- **producer**

  爬取的游戏的开发商，用于搜索和相似比较

  在首次构建时从基础的 steam 网页信息批量索引进入

- **description**

  爬取的游戏的内容简介，用于搜索和相似比较

  在首次构建时从基础的 steam 网页信息批量索引进入

- **related**

  我们小组自己计算得到的相似游戏，存储形式为几个 appid

  在计算完毕之后通过 gameindexerpusher.py 添加

- **vector**

  sentiment analysis 之后的结果，每个方面的分值组成的六维向量

  在计算完毕之后通过 gameindexerpusher.py 添加

- **review**

  steam 网页上的用户评论，用于搜索（用于呈现的信息更完全的 review 我们单独保存了起来）

  最后通过 gameindexerpusher.py 添加

- **price urls cover**

  用于呈现的游戏价格游戏图片游戏封面

  在首次构建时从基础的 steam 网页信息批量索引进入

4. 游戏相似度部分

1. 想法来源

Steam 平台本身会推荐一些相关游戏，而各大游戏榜单中也会给出一部分游戏的相关游戏，但是这些相关游戏首先不一定满足当前游戏玩家的主流审美要求，其次并没有很强的逻辑性和规律性，那么如何进行全新的游戏相似度计算就成了一个很重要的问题。

2. 基本思路

分析了 Steam 各个游戏之间的共性和差异性，发现 Steam 的相关游戏基本上是基于游戏标签进行的。而在分析的过程中，找到了其他的可利用的信息。第一，游戏名，CS 和 CS:GO 这类同系列的游戏名字就十分相似，所以这肯定是一个很重要的评判标准；第二，标签，同样是动作游戏、沙盒游戏的两个游戏一定也是有相似性的；第三，发行商，同一个游戏公司发行的游戏在画面、类型等方面都有很大程度上的相似性；第四，评论，用户对不同游戏的评论可能会用到一些同样的词，例如，打击感强烈，剧情代入感强，而这种主观上的印象是人们在评判相关游戏时很重要的一部分。

对于上述四个方面的内容，分别求其相似度，在通过实验数据的测试和对比，进行加权处理，就可以得到一套全新的相似度计算体系。

3. 实现过程

1. 标签相似度

对于标签的相似度计算，采用了 Jaccard 距离，即交集与并集长度之比：

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

在这里曾经考虑过是否需要根据不同游戏标签的数量进行调整，但是测试了几组数据发现，标签较多的游戏是由于其游戏人次较高，标签中也会包含很多细节性的小标签，这样就使得它与其他游戏的交集长度较长，对其进行处理反倒会使得结果不准确。

2．文本相似度（游戏名及评论）

这项工作在初期利用了 Google 的 word2vec，通过维基百科和新浪新闻中的共计 15G 的语料进行了模型的训练，经过了几个小时的训练之后发现训练结果的词典和本项目（游戏）所需要的词典有着较大的差距，较多词没有收录在词典中，于是这个模型最终用于进行一些词语的相似度计算和其他人工作中用到的 word2vec 的模型。

文本相似度计算的主体利用的是**基于 VSM 的余弦相似度计算**：

第一步，向量空间模型 VSM

向量空间模型（Vector Space Model，简称 VSM）表示通过向量的方式来表征文本。一个文档（Document）被描述为一系列关键词（Term）的向量。

简言之，判断一篇文章是否是你喜欢的文章，即将文章抽象成一个向量，该向量由 n 个词 Term 组成，每个词都有一个权重（Term Weight），不同的词根据自己在文档中的权重来影响文档相关性的重要程度。

Document = term1, term2, ……, termN Document Vector = weight1, weight2, ……, weightN

$$V(d) = (t_1 w_1(d); ...; t_n w_n(d))$$

其中 ti(i=1,2,...n) 是一列相互之间不同的词，wi(d) 是 ti 在 d 中对应的权值。

选取特征词时，需要降维处理选出有代表性的特征词，包括人工选择或自动选择。

第二步，TF-IDF

特征抽取完后，因为每个词语对实体的贡献度不同，所以需要对这些词语赋予不同的权重。计算词项在向量中的权重方法——TF-IDF。

它表示 TF（词频）和 IDF（倒文档频率）的乘积：

$$\text{TF} - \text{IDF} = \text{词频(TF)} \times \text{逆文档频率（IDF）}$$

词频（TermFrequency，简称 TF）表示特征词出现的次数除以文章总词数：

$$\text{词频(TF)} = \frac{\text{某个词在文章中的出现次数}}{\text{文章的总词数}}$$

其中 TF 表示某个关键词出现的频率，IDF 为所有文档的数目除以包含该词语的文档数目的对数值。

$$IDF = \log_2 \frac{|D|}{|w \in d|}$$

|D| 表示所有文档的数目，|wd| 表示包含词语 w 的文档数目。

由于“是”“的”“这”等词经常会出现，故需要 IDF 值来降低其权值。所谓降维，就是降低维度。具体到文档相似度计算，就是减少词语的数量。常见的可用于降维的词以功能词和停用词为主（如：" 的"，" 这" 等），事实上，采取降维的策略在很多情况下不仅可以提高效率，还可以提高精度。

最后 TF-IDF 计算权重越大表示该词条对这个文本的重要性越大。

第三步，余弦相似度计算

这样,就需要一群你喜欢的文章,才可以计算 IDF 值。依次计算得到你喜欢的文章 D=(w1, w2, ..., wn) 共 n 个关键词的权重。当你给出一篇文章 E 时，采用相同的方法计算出 E=(q1, q2, ..., qn)，然后计算 D 和 E 的相似度。

计算两篇文章间的相似度就通过两个向量的余弦夹角 cos 来描述。文本 D1 和 D2 的相似性公式如下：

$$sim(D_1, D_2) = \cos\theta = \frac{\sum_{k=1}^{n} w_k(D_1) \times w_k(D_2)}{\sqrt{(\sum_{k=1}^{n} w_k^2(D_1)) \times (\sum_{k=1}^{n} w_k^2(D_2))}}$$

其中分子表示两个向量的点乘积，分母表示两个向量的模的积。

计算过后，就可以得到相似度了。我们也可以人工的选择两个相似度高的文档，计算其相似度，然后定义其阈值。同样，一篇文章和你喜欢的一类文章，可以取平均值或寻找一类文章向量的中心来计算。主要是将语言问题转换为数学问题进行解决。

这个算法的缺点：计算量太大、添加新文本需要重新训练词的权值、词之间的关联性没考虑等。

但是在本算法的基础上利用上述的 word2vec 模型做词语相似度上的补充就可以很大程度上提高其可行性。

3．加权处理

通过大量的数据测试和结果比对，得出了如下的加权方式：

为了提高同系列游戏的相关性，对游戏设置了一个大于 1 的权值 1.3，而对于另外两个重要判据：标签和评论则赋予权值 1，对于发行商这个判据，由于其与游戏相关度并没有绝对的联系，选择了 0.3 的权值。

按上述方式进行运算，就可以计算出每个游戏与其他游戏之间的相关度，最后取前五个作为该游戏的相关游戏。

4．猜你喜欢

这部分内容的关键在于两方面，第一，“你”，也就是说如何根据户自己的个性来决定推荐给他的游戏；第二，“猜”，这个字点出了这个算法需要一定程度上的猜测，用户虽然很喜欢 A 类游戏，但是也许 B 类游戏也会给他新的体验。

基于上述叙述，最终这部分算法选择了建立在游戏相关度基础上的同元素合并和其他元素补充。对于用户搜索记录中的所有游戏，寻找其相似游戏，随后逐一取交集，这部分游戏就是根据用户的个人喜好选择出来的，而对于这其中没有出现的游戏，则从整个游戏库中选择其他的游戏随机进行补充，实现 “猜” 的部分，这样一来就可以得到一个既有确定性又有随机性的充满惊喜的算法了。

5．网站前端及后端

次的 UI 采用上课所学的 web.py 的框架。在服务器端利用了上课所学的如 GET，POST 方法接收网页数据，加上记录访问记录的 cookies，返回网页模板，展示主页以及搜索界面，或者返回 json 数据供 jQuery ajax 操作处理，进行 queryguesser 和搜索结果排序等功能的实现；在前端的 html 采用 bootstrap 框架本着简介明了的原则设计了一共 5 个界面。总的来说页面比较简介明了，各页面之间相互链接的关系也较为合理，实现了一个搜索引擎 UI 的基本要求。

在服务器端引用了必要的 Python 库以及需要整合的算法代码文件，包括 7 个 url 及对应的处理类，8 个数据处理函数，下面将依据 html 模板一一介绍服务器端以及对应的前端 html 实现。

网站结构

1. 主页

该页面为搜索引擎的主页，除包含每个页面均有的 appbar 以外，其余部分依次为搜索输入组件，其中左侧的下拉按钮组件可以选择搜索的模式，中间的搜索功能支持自动补全，右侧的图片按钮可以进行图片搜索；下方的猜你喜欢模块，根据用户的搜索记录进行相应的游戏推荐。

界面依搜索框左侧的下拉菜单选项分为三种界面，momie@game，momie@company，momie@content，分别为通过标题，开发商，游戏内容搜索游戏，其中，由于 momie@content 是后加内容，所以功能展现及各方面优化有一些简陋。

特点

1. 其中左侧列表的 dropdown 空间采用 js 原生代码编写
2. 中央的 input 输入框，实现自动补全功能使用 jQuery 的 ajax 操作实现，对输入框的 onchange 事件响应以下的 txtchange 函数，由于 web.py 模板限制，jQuery 中的 $ 符号用 jQuery 代替

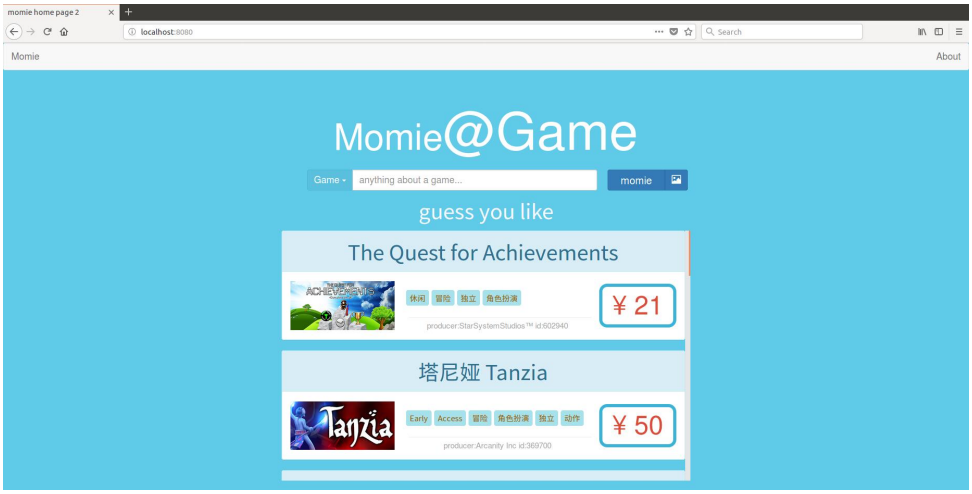
```
<script type="text/ecmascript">
function txtchange() {
    var word = jQuery("#searchinput").val();
    jQuery.ajax({
        type: "GET",
        url: "/guesser",
        data:{'word':word},
        dataType: "json",
        async: true,
        success: function(data) {
            jQuery("#guessresults").html(""); //删除原有数据
            if (data != "null") {
                for (var i = 0; i < jQuery(data).length; i++) {
                    jQuery("#guessresults").append('<li class="list-group-item guesseritem" onclick="mousedown(this)">' + data[i] + '</li>');
                }
                jQuery('#guessresults').width(jQuery('#searchinput').width()+26);
                jQuery("#guessresults").slideDown();
            }
        }
    });
}
//选择其中的提示内容
function mousedown(object) {
    jQuery("#searchinput").val(jQuery(object).text());
    jQuery("#guessresults").fadeOut();
}
//文档框失去焦点，隐藏提示内容
function lost() {
    jQuery("#guessresults").fadeOut();
}
</script>
```

3. 右侧的图像搜索按钮，采用 post 的方式提交图片信息，将原文件提交按钮隐藏，自制的按钮采用绝对定位加 jQuery 动态调整的方式，定位在文本搜索提交按钮的右侧
4. 下方的猜你喜欢采用模块内滚动条的方式，以将搜索框保持在界面当中

关键模版参数

- switched：在自动纠错模块中，用于保存用户原有的搜索关键词，便于用户搜索纠错前的关键词；同时做为是否显示纠错提示的判定条件
- ifcontent：用于排序模块调用’/rank’ 时的参数，决定排序时按照 momie@game 的搜索方式还是按照 momie@content 的搜索方式

对应模版： home2.html



对应后端：

- **home**-获取 ifC 参数以及 cookie 'visit'，调用 guess 函数获取猜你喜欢模块推荐的游戏 ID，再调用 minigamegetter 函数获取需要在主页上的信息，存入一个 list 当中，最后将 ifC 参数与该 list 传给 home2.html 模板。
- **guesser**-获取 word 后，调用 Guesser 类的 guess 方法获取补全结果，使用 json 库的 dumps 方法转为 json 模式后返回，便于 html 的调用

2. 游戏（内容）搜索结果页面

该界面显示 momie@game 与 momie@content 的搜索结果，页面基调为天蓝色，主要由 appbar，纠错提示，排序按钮，以及下方的搜索结果组成。

特点

1. appbar 中加入橙色按钮，可以按照 momie@company 模式搜索输入框中内容
2. 排序功能按钮使用 bootstrap 中的 pagination 组件，功能的实现上采用 ajax 方式调用函数得到排序结果后再填充到页面中
3. 下方的搜索结果采用 flex 布局，标题为游戏名称，下方为封面，标签，开发商，id，以及游戏价格
4. 自动纠错模块为一黄条，提供用户搜索纠错前词汇的链接，以及关闭该提示的按钮

对应模版： resultGame.html

对应后端：

- **gamesearch**-获取以上三个参数后，根据是否纠错以及搜索模式返回不同的模板参数，其中，由于自动纠错函数 (correcter 类的 correct 方法）对于某些正确结果会返回相同值，所以进行了判重，而对于某些自动纠错的结果会没有搜索结果，所以也进行了筛选。
- **gamerank**-调用 gamesearcher 函数（主要因素为 rankmod 参数）获取返回结果后，转换为 json 格式传递给 html 模板文件显示。

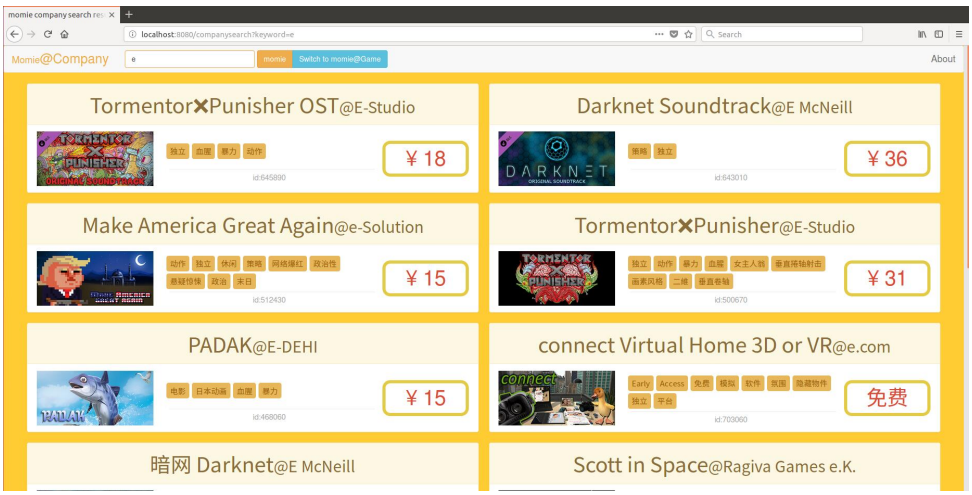
3. 按公司搜索结果页面

该界面显示 momie@company 的搜索结果，页面基调为橙黄色，主要由 appbar，纠错提示和下方的搜索结果组成。页面构成与 resultGame.html 基本相同，只是在搜索结果的现实上略作调整，以突出开发商的显示。

特点

1. 与 resultGame.html 基本一致，无排序功能，更换了主题颜色，切换模式按钮，在搜索结果的显示中将开发商使用 @ 连接在游戏名称之后突出显示

对应模版： resultCompany.html



对应后端：

- **companysearch**-机制与 gamesearch 中基本相同，调用 companysearcher 函数返回搜索结果，传递参数给 resultCompany.html 模板。

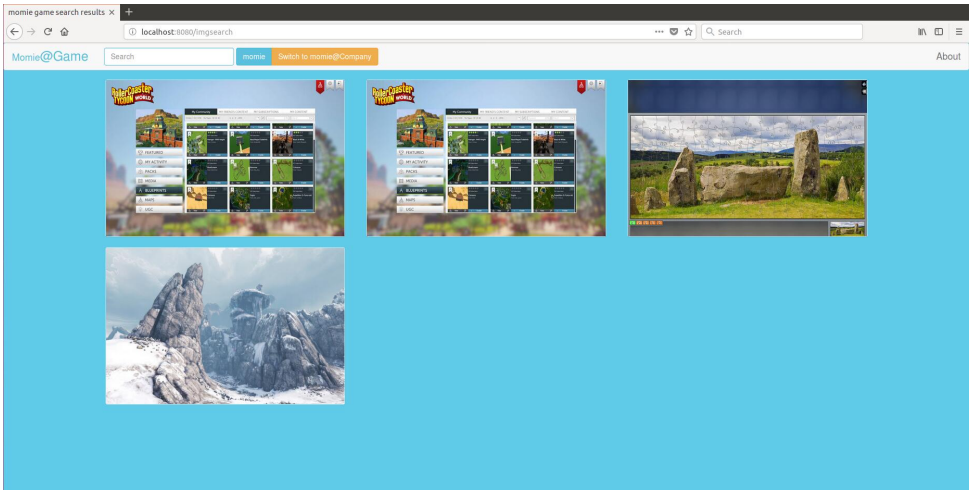
4. 图片搜索结果页面

该界面为图像搜索结果的显示界面，但依旧保留了上方 appbar 中的文字搜索框，美中不足的是，没有在 appbar 中加入图像搜索按钮。

特点

1. 搜索结果采用 flex 布局，保证了每一行的宽度相同，但是没有使得图片间的间距相同
2. 对于图片，对 onmouseenter 事件响应 jQuery 中的 slideDown 事件，onmouseleave 事件响应 jQuery 中的 slideUp 事件，以实现游戏名称的浮动显示，增强了页面对于图片的可读性

对应模版： resultImg.html



对应后端：

- **imgsearch**-获取 POST 方法上传的文件后，调用 filename 参数，对文件类型进行检查，确认为图片后，调用 file 参数获取图像文件内容，读取后保存至服务器端。之后调用 imgsearcher 函数获取搜索结果，删除服务器端文件，将搜索结果返回值 resultImg.html 模板。

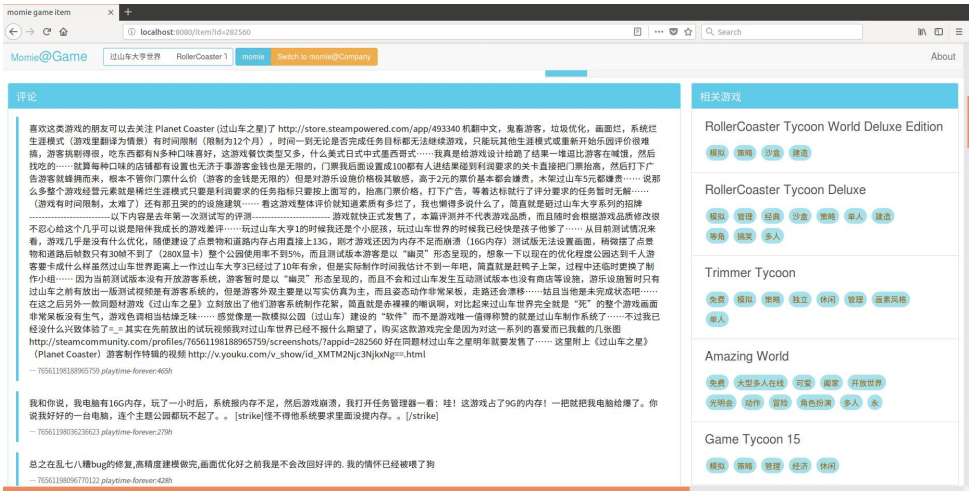
5. 游戏信息展示页面

该页面为游戏的介绍页面，除上方的 appbar 外，包括三层 div，第一层存储游戏名称，游戏描述，以及游戏评分；第二层存储游戏图片，封面，开发商，steam 链接，价格以及标签第三层存储用户评论以及相关游戏链接。

特点

1. 大量使用 div 嵌套配合 flex 布局实现页面的稳定显示
2. 游戏图片板块使用了 bootstrap 的 carousel 控件实现了幻灯片效果
3. 评论引用了 bootstrap 的引用控件，更改了 border 的颜色
4. 相关游戏引用了 bootstrap 的 list-group 控件，提供相关游戏的名称，标签以及链接

对应模版：itemGame.html



对应后端：

- **item**-保存游戏 id 至 visit cookie，调用 gamegetter 函数获取游戏相关信息，传递给 itemGame.html 模板

调用函数：

gamegetter(id)

对通过 idget 得到的游戏信息进行预处理，以便于在模板中显示。其中游戏的描述由于在 index 前采用结巴分词进行处理，所以如果是中文的话，进行了去空格的处理；而游戏的评分进行了可视化的换算；游戏的评论提取了评论内容，评论玩家 id，以及评论玩家已玩时长；对于相关游戏获取其游戏名称，游戏标签，连同游戏 id 一并返回。

6. 中文名获取及别名获取

1. 游戏别名的获取

利用百度搜索的结果词条猜游戏的别名，百度的结果搜索词条意思是在百度上搜索已经爬取到的游戏的英文官方名，假设得出的结果中拥有游戏的其他别名。

提取原理是利用马尔科夫转移模型，因为判定名字实际上算是一种分词的过程，所以统计每个字转移到另外的的字的概率。在多次出现游戏中文名的时候，特定的字之间的转移概率高于平均水平，因此可以以此作为判断标准来生成出名字。

做法是先统计了整个文本库平均的马尔科夫转移概率，但是类似于 rf-idf，对于同一个游戏的搜索结果有多次出现的前后字的关系，统计时只算做一个，之后对每一个游戏的搜索结果统计马尔科夫转移概率，用单个游戏的值除以整个的值，得到这两个字相连可能是游戏名一部分的概率。

因为调整参数的时间不多，所以效果比较一般，伴随着出现的总有游戏主播的名字，一些游戏相关的词汇（硬盘，存档，三维，修改. . .）

到这一步，生成的名字中还总是有重复出现的错误词条，所以使用本文件把每一个游戏猜得的与其他游戏相重复的词条删除，最后达到比较可以使用的效果。

2. 中文名获取

以上的别名挖掘内容可以获取很多可能的别名，但通过该方法猜测出的名字是具有一定概率的组合，无法作为正确的中文名进行使用。为了准确地获得最常用的中文名，还需要考虑

别的方法。

跟以上方法一样，利用百度的搜索结果。观察各文章内容的格式，可以总结出一个规律：中文名总以形如《难死塔 (tricky towers)》的形式出现，因为在中文规范中，游戏的标题都采用书名号。

因此，在百度搜索《tricky towers》，然后将前五页页面内容上的所有书名号中的中文内容提取出来。统计各内容出现的次数，从大到小进行排序，出现数量较多的就极有可能是该游戏的正式中文名。最后，对得到的结果进行去重筛选，比对 dlc 信息，便可以得到最终的正式中文名的结果。我们抽样了 100 个游戏进行测试，该方法完全正确的概率在 0.9 以上。

4 总结

Momie 的强大性在于它不只是一个简单的对 Steam 的辅助，它对 Steam 中的大量信息进行了整合，并从当代青年的审美观出发，对游戏做了全新的评价机制和关联机制，完善了大量的工作，不仅可以搜索游戏，更是能够帮你找到想玩的游戏，推荐有趣的游戏。

Momie 的工作仍然有许多待完善之处，数据库的更新和完善，功能的健全和丰富，算法中对 word2vec 等工具更好的利用，我们尚未结束，我们仍在路上。

Momie 团队相信这不仅是一次实验，不仅是一次作业，是我们对搜索引擎全新探索的开始，我们十分感谢老师和助教的悉心指导和帮助，也为自己付出的努力感到自豪，更将坚定不移地继续前行。。