Universidad Nacional del Centro de la Provincia de Buenos Aires
Facultad de Ciencias Exactas

Ingeniería de Sistemas

Taller de Python y aplicaciones

Generación de juegos: parsing de historias generadas automáticamente utilizando técnicas de NLP

Autor

Luciano Tangorra

1. Resumen

En este trabajo se presenta un parser de diferentes atributos sobre un texto generado automáticamente. El objetivo es encontrar unidades mínimas que ayuden a recrear dicho texto de forma visual en formato de videojuego, analizando cada oración y su composición y buscando relaciones entre las palabras y las distintas sentencias. Se plantean cuatro valores de interés y se aplican técnicas de NLP, junto con diversos modelos pre entrenados para cumplir el objetivo.

El análisis del texto y el desarrollo de los algoritmos permitió comprender más en detalle los alcances y límites que tiene este proyecto, concluyendo en que al parsear el texto se pierde información que quizás es importante. A su vez, se comprendió que es necesario reevaluar las libertades brindadas en el juego y plantear nuevas limitantes que simplifiquen la cantidad de atributos a parsear del texto.

2. Introducción

Aunque en el sector de los videojuegos se suelen nombrar conceptos como la inteligencia artificial, sólo se utiliza una parte pequeña de la misma. Uno de los métodos más aplicados es la generación procedural [1], tanto de mapas como de objetos que puede poseer el personaje. Esta técnica ayuda a generar sensación de libertad y de “mundo infinito”. La realidad es que todo lo generado se encuentra restringido por los mismos desarrolladores. Uno de los juegos más conocidos que utiliza este concepto es el Minecraft [2]. Cada vez que el jugador elige iniciar una nueva partida, un nuevo mundo (normalmente único) es generado siguiendo ciertas reglas, como por ejemplo, las zonas de hielo no tienen árboles de selva. Otro juego muy conocido es el No Man 's Sky [3], en donde todos los mundos se generan de forma procedural y las probabilidades de que dos mundos sean iguales son ínfimas.

A partir del auge de los modelos de deep learning para la generación de lenguaje natural, surgieron diferentes propuestas sobre qué usos podrían darse en el mundo de los videojuegos. AIDungeon [4] es uno de los casos de estudio de mayor interés ya que utilizando GPT-2 [5] (y luego GPT-3 [6]) es capaz de generar una historia interactiva infinita. Esto quiere decir que, basado en un input del usuario (texto en lenguaje natural sobre qué se desea hacer) se generará una historia en donde se mantendrá (o intentará) el contexto del mundo generado.

La idea de esta propuesta es asombrosa por sí sola pero le falta algo esencial que hace que tantos juegos triunfen: la parte visual. Desarrollar un juego capaz de generarse automáticamente y que cada partida sea totalmente diferente de la otra (distinta historia, mundo y personajes) es algo nunca antes visto.

En este trabajo se propone buscar una forma de desarrollar la base para un juego que se genere a sí mismo, en donde cada partida sea única y cada experiencia distinta. Se hará énfasis en la utilización de herramientas de análisis y generación de texto para facilitar la creación de la parte visual del juego.

3. Objetivos

Como hipótesis del trabajo se plantea la generación de historias utilizando deep learning para, a partir de esto, desarrollar un videojuego basado completamente en esta historia obtenida. Por lo tanto, el objetivo principal es buscar formas de procesar los textos para diferenciar las relaciones entre las palabras y, de este modo, poder obtener valores de interés: Entidades, Descripción, Posesiones y Acciones. Esto es sólo una de las etapas necesarias para llevar a cabo el proyecto de generar mundos visuales mediante deep learning.

4. Desarrollo

Para el desarrollo del proyecto se utilizó el lenguaje de programación Python junto con diferentes librerías, tales como: Spacy [7], Neuralcoref [10], Contractions [12], AllenNLP Predictor.

- **SpaCy** [7]: una de las librerías más utilizadas para NLP (natural language processing). Se pueden encontrar varios modelos pre entrenados, así como utilizar el pipeline por defecto, al cual se le ingresa una oración y la procesa de diferentes formas, permitiendo obtener la etiqueta POS [8], las entidades encontradas, las dependencias entre las palabras, entre otras cosas. Se eligió esta librería (en lugar de otras, como como NLTK [9]) por su facilidad de uso y por las ventajas que brinda el pipeline por defecto.
- **Neuralcoref** [10]: es una librería basada en deep learning creada para completar las correferencias en el texto [11]. Es una extensión de SpaCy, la cual se agrega al pipeline por defecto y se aumentan las funcionalidades y la cantidad de resultados obtenidos. La performance del modelo de correferencias se basa en el rendimiento que tengan los modelos de etiqueta POS, el parser y NER (name-entity recognition) implementados en SpaCy. Esta librería, a pesar de no haberse agregado en un principio, fue de utilidad ya que permitió mantener las diferentes entidades en distintas partes del texto (reemplaza he, him, his con el nombre de la entidad vista que crea conveniente).
- **Contractions** [12]: permite eliminar la contracción en el texto, por ejemplo “I’ve” se reemplaza con “I have”.
- **AllenNLP Predictor**: es una librería que posee un modelo pre entrenado de semantic role labeling, la cual sirvió para encontrar las acciones de las entidades. Este modelo busca: entidad, qué acción realizó dicha entidad y sobre qué se realizó esa acción. Para probar este modelo, ingresa al siguiente link: AllenNLP - Demo.

Se comenzó buscando modelos pre entrenados para generar historias de estilo medieval, ya que el objetivo es desarrollar un RPG (role-playing game) ambientado en esa época. Una vez encontrado y elegido uno de los modelos se probaron posibles salidas del mismo. Este modelo generador proporcionó las siguientes oraciones:

- You are pedro aznar, a knight living in the kingdom of Larion.
- You have a steel longsword and a wooden shield.
- You are on a quest to defeat the evil dragon of Larion.
- You've heard he lives up at the north of the kingdom.

- You set on the path to defeat him and walk into a dark forest.
- As you enter the forest you see a homunculus wearing a black cloak and a red crown on his head.
- You draw your steel longsword, preparing for a fight.

Estas oraciones funcionarán como el “conjunto de entrenamiento”. Esto es así porque se utilizarán para desarrollar los algoritmos que facilitarán la creación de la parte visual, los cuales luego son probados en diferentes oraciones para evaluar su correctitud.

Para comprender el “conjunto de entrenamiento” se realizó un análisis general del texto obtenido, observando las similitudes y diferencias entre las oraciones, la relación entre las palabras y la etiqueta asignada a cada una. A continuación se desarrollaron tres algoritmos. El primero identifica las entidades y, a su vez, obtiene las descripciones de las mismas. El segundo reconoce las posesiones de las entidades. El último, por su parte, obtiene las acciones realizadas por las entidades.

Para la implementación de los algoritmos se realizaron clases independientes entre sí, las cuales poseen atributos de clases y diferentes funciones, privadas y públicas, que permiten una lectura más sencilla del código. Todas las clases deben crearse y, para facilitar su uso, se debe llamar a la función **process**, la cual se encarga de correr el algoritmo con los parámetros de entrada que ingresa el usuario al objeto de la clase. En caso de no haber ingresado ningún parámetro, se utilizan los valores por defecto. Una vez que la función process finaliza su ejecución, el usuario puede acceder a diferentes atributos de la clase para observar los resultados. El resto de las funciones son privadas, evitando que el usuario tenga acceso y pueda generar, sin quererlo, resultados no esperados. Además, el objetivo final es llevar este parser a una librería para poder importarla en C++ para generar la visualización con Unreal Engine, por lo que mientras más sencilla sea su utilización, menos posibilidades de errores futuros.

Esta idea fue tomada de SpaCy, librería en la cual una vez que ingresas un texto, se procesa y se puede acceder a los atributos de clase para obtener los resultados.

1. Análisis del texto

Como se dijo anteriormente, con el análisis de texto se buscó encontrar similitudes y diferencias entre las oraciones, la relación entre las palabras y la etiqueta asignada a cada una. En la Figura 1, se muestra un ejemplo de la relación entre las palabras de la primera oración. Se puede observar que “pedro” y “aznar” son sustantivos propios, en donde ambos están enlazados por ser una composición. “Aznar” tiene relación con “knight”, en la que este último es modificador aposicional del primero. Por su parte, “living” lo clasifica como VERB (verbo), relacionado con “kingdom” (objeto de la proposición) por medio de una preposición. A su vez “Larion” (sustantivo propio) se encuentra también relacionado con “kingdom”.

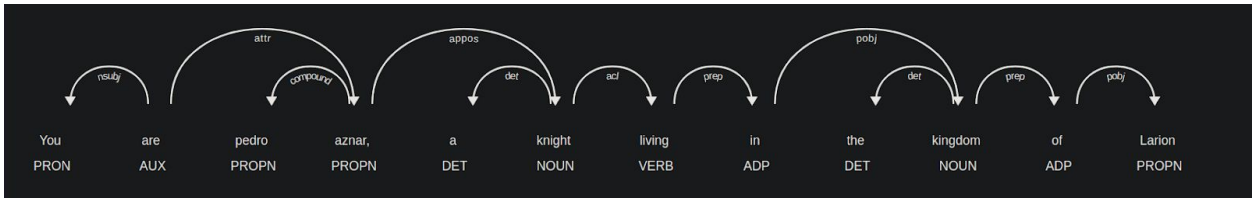

Figura 1. Parsing del texto utilizando el pipeline por defecto de SpaCy. Pueden observarse las dependencias entre las palabras y su etiqueta POS.

A partir del análisis realizado, se planteó la obtención de diferentes valores de utilidad:

- Entidades: personas, lugares u objetos que tengan relevancia en la historia. En el caso explicado anteriormente (Figura 1) las entidades están representadas por “pedro aznar” y “Larion”.
- Descripciones: están relacionadas a las entidades, ya que facilitan comprender cómo son o cómo están compuestas. En nuestro ejemplo, “knight” describe “pedro aznar” y “kingdom” a Larion. Sin estas descripciones no se podría saber qué son ni cómo se ven las entidades.
- Posesiones: objetos o cosas que poseen las entidades. En el ejemplo no hay ninguna posesión, aunque si analizamos la segunda oración “You have a steel longsword and a wooden shield” las posesiones son “longsword” y “shield” para la entidad “You”, la cual podría ser reemplazada por pedro aznar.
- Acciones: busca responder **quién** hizo **qué** (acción), hacia **quién** o en **dónde**. En el ejemplo la acción podría estar compuesta como: **quién** pedro aznar **qué** living (vive) **en dónde** (Larion).

2. Entidades y descripciones

Para obtener las entidades y las descripciones se creó una clase llamada EntityParser (Figura 2), la cual posee como parámetro de entrada obligatorio el nombre del personaje y como parámetro de entrada opcional si se permite el uso de la librería neuralcoref, además del modelo de SpaCy a utilizar y si se desean ver los mensajes de debug de los procesos que se ejecutan. Esto se realiza colocando parámetros de entrada a la función __init__. En esta función se plantean todos los atributos de clase necesarios para funcionar. A pesar de no ser estrictamente necesario, ya que se pueden crear atributos de clase en cualquier lugar y momento, se realizó de este modo para tener una clara visualización de los atributos utilizados y poder encontrarlos más fácilmente.

```
class EntityParser():

  def __init__(self, character_name, use_coref=False, nlp=None, debug=False):
    """
    Init the global variables used in most parts
    """
    self.debug_ = debug
    if nlp is None:
      self.nlp = spacy.load('en_core_web_sm')
      neuralcoref.add_to_pipe(self.nlp)
    else:
      self.nlp = nlp
    self.use_coref_ = use_coref

    self.character_name_ = character_name
    self.text_ = ""
    self.full_history_ = ""

    self.last_person_entity_ = ""
    self.all_entities_ = {}
    self.description_ = {}
    self.indexes_ = {"PERSON": 0,
                     "GPE": 0,
                     "ADDEDENTITY": 0}
    self.subject_words_ = {"he", "she", "they", "him", "her", "his"}
```

Figura 2. Definición de la clase EntityParser junto a su método __init__, sus parámetros de entrada y sus atributos de clase.

Como se nombró anteriormente, la librería SpaCy posee un modelo entrenado para obtener las entidades del texto, por lo que la clase EntityParser puede verse como una implementación que no es necesaria. Además de hacer el llamado a la función, coloca los resultados en un diccionario del tipo {TipoEntidad_I: nombre", ...}, diferenciando cada entidad ya que puede haber más de una de cada tipo. A su vez, permite la recolección de entidades basadas en los verbos. Esto quiere decir que se pueden reconocer entidades más allá de lo que brinda SpaCy buscando entre los hijos (viendo el texto como un grafo) de los verbos de la oración y quedándose con los sustantivos, reconociendo si se trata de una entidad persona o una entidad ubicación/lugar. En caso de encontrar un sustantivo propio, se utiliza como su descripción los sustantivos no propios encontrados. En el ejemplo de la Figura 1, el sustantivo "kingdom" servirá como descripción de "Larion".

Para obtener las entidades finales junto a sus descripciones, se utilizaron diferentes conceptos vistos en la materia, tales como:

- Llamados a métodos privados de la clase: el objetivo de realizar diferentes métodos o funciones es abstraer el comportamiento que se va a realizar en varios lados, o bien para simplificar el código y que quede más legible. En la Figura 3 puede observarse un ejemplo en el código realizado.

```python
def process(self, text, by_verb=True):
  """
  Main function which does the heavy process in order
  to obtain the entities
  """
  if self.debug_:
    print("DEBUG: process")
  self.__set_text(text)
  self.__process_entities()
  if by_verb:
    self.__process_entities_by_verb()
```

Figura 3. Función process dentro de EntityParser. En esta función se observan varios llamados a métodos privados de la clase.

- Tratamiento de strings: los strings en python son muy utilizados y se pueden manipular de modo sencillo empleando los métodos que poseen por defecto. En el ejemplo de la Figura 4 puede observarse cómo se tomó una historia completa (un párrafo), se transformó a string (antes era un texto SpaCy) y luego a una lista con las diferentes oraciones separadas por ". y se tomó la última oración. En la Figura 5 puede observarse cómo se unen todos los elementos de una lista en un string separando los elementos por un espacio " ".

```python
str(self.full_history_.split(". ")[-2])
```

Figura 4. Ejemplo de manipulación de string en el cual se castea al tipo String, luego se separa para formar una lista de todos los elementos que estaban separados por ". ", es decir, todas las oraciones, y luego se toma el último elemento.

```python
text_preprocessed = " ".join(new_text)
```

Figura 5. Ejemplo de manipulación de listas en el cual se transforma la lista en un string con todos los elementos unidos por un espacio " ".

- Uso de recursión: la recursión se utiliza cuando es necesario realizar el mismo procedimiento una y otra vez, modificando algunos parámetros de entrada en cada repetición. Esto es posible realizarlo mediante un loop, aunque es mucho más entendible y corto el código al aplicar este tipo de técnicas. En la Figura 6 podemos observar el uso de recursión para obtener todos los elementos hijos de algún verbo del grafo de texto que son sustantivos y almacenarlos como nuevas entidades.

```python
def __get_noun_children(self, word, new_entities):
  """
  Recursively obtains all nouns that are connected
  with a verb (directly or indirectly)
  """
  if self.debug_:
    print("DEBUG: __get_noun_children")
  for next_word in word.children:
    if "NOUN" in next_word.pos_ or "PROPN" in next_word.pos_:
      new_entities.add(next_word)
      if self.debug_:
        print(f"Word: {next_word} /// PosTag: {next_word.pos_} /// Dep: {next_word.dep_}")
    else:
      self.__get_noun_children(next_word, new_entities)
```

Figura 6. Ejemplo de recursión en la cual se busca a todos los elementos hijos del grafo de texto y se agregan como entidades las palabras que sean sustantivos.

- Manejo de excepciones: en caso de que suceda algún error inesperado en el código, se pueden utilizar las sentencias del try except para capturarlo y evitar que el código genere errores en plena ejecución. Permite observar cuál fue el error o realizar algún tratamiento en caso de tenerlo previsto. En la Figura 7 puede observarse el uso de las sentencias try except para lanzar un warning informando que no se encontró una etiqueta para la entidad. Una posible solución es que si no conoce la etiqueta, se redireccione a alguna predefinida. Esto no sería correcto ya que podría ser tanto una entidad persona como una entidad lugar/ubicación, por lo que se debe reconocer la etiqueta error y ajustar el algoritmo según corresponda.

```python
try:
  new_label = label
  if "ORG" in new_label:
    new_label = "GPE"
  elif not isinstance(entity, spacy.tokens.span.Span):
    if "dobj" in entity.dep_:
      new_label = "PERSON"
    elif "pobj" in entity.dep_:
      new_label = "GPE"

  self.indexes_[new_label] += 1

  if self.debug_:
    print(f"Entity: {entity} /// Label: {label} /// Assigned_label: {new_label}")
  return f"{new_label}_{self.indexes_[new_label] - 1}"
except Exception as err:
  print(f"Warning: {new_label} not recognized. Error {err}")
```

Figura 7. Ejemplo de sentencias try except para capturar errores. En esta imagen estas sentencias fueron utilizadas para capturar las etiquetas que no se conocen y que imprima un warning. De este modo se puede saber cuál fue la causa del error y asignar esa etiqueta según corresponda.

Continuando con el texto de la Figura 1, las entidades y las descripciones encontradas para este caso fueron:

- entities: {'PERSON_0': 'pedro aznar', 'GPE_0': 'Larion'}
- descriptions: {'GPE_0': 'kingdom', 'PERSON_0': 'knight'}

Se pudo observar que el parser realiza un muy buen trabajo en obtener las entidades y sus descripciones, además de estar relacionadas las mismas mediante la key del diccionario, lo cual facilitará su uso y relación.

3. Posesiones

En el caso de las posesiones, la clase creada se llama PossesionsParser, la cual sigue una lógica similar al EntityParser. En la Figura 8 se observa la definición de la clase y del método de inicialización de la misma, junto a sus parámetros de entrada y sus atributos. Se utilizaron métodos privados, atributos de clase y recursión (Figura 9).

```
class PossesionsParser():

  def __init__(self, nlp=None, debug=False):
    self.debug_ = debug
    if nlp is None:
      self.nlp = spacy.load('en_core_web_sm')
    else:
      self.nlp = nlp
    self.possesions_ = {}
    self.possesives_ = {"have", "has", "wearing"}
```

Figura 8. Definición de la clase PossesionsParser junto a su método __init__, sus parámetros de entrada y sus atributos de clase.

```
def __get_possesions(self, word, empty=True, entity_possesions=[]):
  """
  Recursively obtains all entity possesions looking if the childrens
  are connected via dobj or conj
  """
  if self.debug_:
    print("DEBUG: __get_possesions")

  if empty:
    entity_possesions = []

  for children in word.children:
    if "conj" in children.dep_ or "dobj" in children.dep_:
      entity_possesions.append(str(children))
      self.__get_possesions(children, False, entity_possesions)

  return entity_possesions
```

Figura 9. Ejemplo de método de la clase PossesionsParser en el cual se utiliza recursión para encontrar todas las posesiones de una misma entidad.

Esta clase recorre todas las entidades ya encontradas por medio del EntityParser y busca si alguna de ellas tiene algún posesivo (atributo de clase de tipo set que contiene la lista de posesivos hardcodeada) en la oración. En caso de encontrar algún posesivo vinculado a la entidad, se va a buscar alguna de sus pertenencias (dobj) y si posee más de una, se encontraran vinculadas entre sí (conj).

4. Acciones

La clase ActionsParser (definición en Figura 10) es la encargada de devolver un diccionario de listas de diccionarios, los cuales son de la forma {**quién**: [{"V": **qué**}, {"RECEIVER": **hacia quién**}, {"LOC": **dónde**}]. Para esto se obtienen las entidades mediante list comprehension, ya que se desea aplicar una función a cada entidad de este tipo (Figura 11). Luego se asignan a la lista correspondiente las descripciones de cada entidad, ya que pueden ser vistas como sujeto de la acción y se busca reconocer esos casos.

```
class ActionsParser():

  def __init__(self, threshold=0.5, nlp=None, predictor=None, debug=False):
    self.debug_ = debug
    if predictor is None:
      self.predictor = Predictor.from_path("https://storage.googleapis.com/a
    else:
      self.predictor = predictor
    if nlp is None:
      self.nlp = spacy.load('en_core_web_sm')
    else:
      self.nlp = nlp
    self.threshold_ = threshold
    self.actions_ = []
```

Figura 10. Definición de la clase ActionsParser junto a su método __init__, sus parámetros de entrada y sus atributos de clase.

```
persons = [self.nlp(e) for l, e in entities.items() if "PERSON" in l]
locations = [self.nlp(e) for l, e in entities.items() if "GPE" in l]
```

Figura 11. Obtención de diferentes tipos de entidades, a las cuales se les aplica el pipeline completo de SpaCy.

Se utiliza el predictor de AllenNLP para semantic role labeling, el cual parsea todo el texto. Una vez obtenidas todas las acciones mediante el modelo de SRL de AllenNLP, se procesan para extraer únicamente el sujeto que realiza la acción, el verbo, hacia quién y en qué lugar y se matchean con las entidades ya conseguidas con la clase EntityParser. En caso de no matchear con ninguna, se marcarán como "Deleted" para tener un control más estricto de las mismas y eliminarlas en caso de ser necesario. Hay varios tipos de roles semánticos posibles (Figura 12), aunque no se reconocen todos ya que el algoritmo aún se encuentra en etapa de desarrollo y se planteó ir de a poco y con pasos seguros, ya que parsear todo desde el principio es una tarea muy compleja.

Tag	Description	Tag	Description
adv	adverbial modification	mod	modal
cau	cause	neg	negation
dir	direction	prd	secondary predication
dis	discourse	prp	purpose
dsp	direct speech	pnc	purpose not cause
ext	extent	rcl	relative clause link
gol	goal	rec	recipricol (eg herself, etc)
loc	location	slc	selectional constraint link
mnr	manner	tmp	temporal

Figura 12. Posibles roles semánticos a obtener por el modelo de SRL de AllenNLP.

Los resultados obtenidos (Figura 13) para esta clase son satisfactorios pero no perfectos, ya que requiere un análisis de resultados y corrección de las entidades parseadas mayor. Hay que destacar que el modelo de AllenNLP para SRL hace un muy buen trabajo, aunque sin un post procesamiento no habría manera de realizar una visualización de estas acciones, ya que las mismas pueden ser muy complejas.

```
actions:
    {pedro aznar: [{'V': 'living'}, {'RECEIVER': 'Deleted'}, {'LOC': Larion}]}
```

Figura 1. Resultados obtenidos luego de ejecutar el método process de la clase ActionsParser.

5. Conclusiones

El objetivo del trabajo era obtener diferentes valores de interés (entidades, descripción, posesiones y acciones) que ayuden a representar de manera visual una historia generada mediante algoritmos de inteligencia artificial.

Los resultados obtenidos luego de aplicar los algoritmos para todo el “conjunto de entrenamiento” (Tabla 1) fueron satisfactorios, aunque poseen cierto grado de error. El mayor error se encuentra en el reconocimiento de entidades, ya que todos los sustantivos unidos a un verbo son tomados como tales, a pesar de no ser así en todos los casos. Esto no es un problema grave, ya que las entidades permiten reconocer unidades atómicas, es decir, la unidad mínima a distinguir luego en las acciones. Por otro lado, en el caso de estas últimas, hay varias que no son del todo correctas y, de igual modo, no todas son relevantes. Por ejemplo, en el caso de la primera oración, no es necesario saber que pedro aznar vive en Larion, ya que no aporta información de donde está actualmente (lugar a mostrar en el juego).

Valor de interes	Resultado
entities	{'PERSON_0': 'pedro aznar', 'GPE_0': 'Larion', 'PERSON_1': 'dragon', 'GPE_1': 'north', 'GPE_2': 'path', 'GPE_3': 'forest', 'PERSON_3': 'homunculus', 'PERSON_4': 'longsword'}
descriptions	{'GPE_0': 'kingdom', 'PERSON_0': 'knight'}
possesions	{'PERSON_0': ['longsword', 'shield'], 'PERSON_3': ['cloak', 'crown']}
actions	- {pedro aznar: [{'V': 'living'}, {'RECEIVER': 'Deleted'}, {'LOC': Larion}]} - {pedro aznar: [{'V': 'defeat'}, {'RECEIVER': dragon}, {'LOC': 'Deleted'}]} - {pedro aznar: [{'V': 'heard'}, {'RECEIVER': dragon}, {'LOC': 'Deleted'}]} - {dragon: [{'V': 'lives'}, {'RECEIVER': 'Deleted'}, {'LOC': Larion}]} - {pedro aznar: [{'V': 'set'}, {'RECEIVER': 'Deleted'}, {'LOC': 'Deleted'}]} - {pedro aznar: [{'V': 'defeat'}, {'RECEIVER': dragon}, {'LOC': 'Deleted'}]} - {pedro aznar: [{'V': 'walk'}, {'RECEIVER': 'Deleted'}, {'LOC': forest}]} - {pedro aznar: [{'V': 'enter'}, {'RECEIVER': 'Deleted'}, {'LOC': 'Deleted'}]} - {pedro aznar: [{'V': 'see'}, {'RECEIVER': dragon}, {'LOC': 'Deleted'}]} - {homunculus: [{'V': 'wearing'}, {'RECEIVER': dragon}, {'LOC': 'Deleted'}]} - {pedro aznar: [{'V': 'draw'}, {'RECEIVER': longsword}, {'LOC': 'Deleted'}]} - {pedro aznar: [{'V': 'preparing'}, {'RECEIVER': 'Deleted'}, {'LOC': 'Deleted'}]}

Tabla 1. Resultados obtenidos luego de aplicar los algoritmos a todo el “conjunto de entrenamiento”

Se puede concluir que es posible realizar un parser básico para resolver este tipo de tareas, aunque desarrollar un videojuego que se genera automáticamente requiere el desarrollo de varias etapas, por lo que habría que realizar varias iteraciones más para llegar a un producto terminado. Una de las principales limitantes es que el hecho de ofrecer extrema libertad a los usuarios de realizar el tipo de acción que deseen, por lo que se tendrían que parsear todas las acciones posibles. Esto es hardcodear la solución a cada posible acción (verbo), lo cual no es viable. Se seguirá iterando, buscando definir un problema más acotado y mejorando los algoritmos hasta obtener una demo.

6. Bibliografía

1. https://en.wikipedia.org/wiki/Procedural_generation
2. https://en.wikipedia.org/wiki/Minecraft
3. https://www.nomanssky.com/
4. https://en.wikipedia.org/wiki/AI_Dungeon
5. https://openai.com/blog/gpt-2-1-5b-release/
6. https://en.wikipedia.org/wiki/GPT-3
7. https://en.wikipedia.org/wiki/SpaCy
8. https://en.wikipedia.org/wiki/Part-of-speech_tagging
9. https://www.nltk.org/
10. https://github.com/huggingface/neuralcoref
11. https://es.wikipedia.org/wiki/Correferencia
12. https://pypi.org/project/contractions/