In [ ]:

```
import spacy
from spacy.symbols import *
import numpy as np
from word2number import w2n

# Load trained nlp model
# trained_model_path = "I:\\My Drive\\UCI\\Winter 2023\\COMPSCI 175\\text_parse\\text_parse\models\\trained-trf-pos-model"
nlp = spacy.load ("en_core_web_trf")

from gensim.models import KeyedVectors
import gensim
from gensim import models
from gensim.models import Word2Vec

# Load pretrained model (since intermediate data is not included, the model cannot be refined with additional data)
# vector_model_path = "I:\\My Drive\\UCI\\Winter 2023\\COMPSCI 175\\text_parse\\text_parse\\models\\GoogleNews-vectors-negative300.bin.gz"
# model = Word2Vec.load_word2vec_format('GoogleNews-vectors-negative300.bin", binary=True, norm_only=True) -> Deprecated

# vector_model_path = "I:\\My Drive\\UCI\\Winter 2023\\COMPSCI 175\\text_parse\\text_parse\\models\\GoogleNews-vectors-negative300.bin.gz"
# model = gensim.models.KeyedVectors.load_word2vec_format (vector_model_path, binary=True) # without *norm_only* param

# model.init_sims(replace=True)
# model.save("models/GoogleNews")

# Load saved vector model
# saved_vector_model_path = "models/GoogleNews"
saved_vector_model_path = "I:\\My Drive\\UCI\\Winter 2023\\COMPSCI 175\\text_parse\\text_parse\\models\\GoogleNews"
model = KeyedVectors.load (saved_vector_model_path, mmap='r')

import warnings
warnings.filterwarnings(action='ignore', category=UserWarning)  # suppress TracerWarning
```

In [ ]:

```
DEBUG = False

command_map = {
            "move": {
                "stop": ["move 0", "strafe 0"],
                "forward": "move 1", "forwards": "move 1", "back": "move -1", "backward": "move -1", "backwards": "move -1", "right": "strafe 1", "left": "strafe -1",
                "north": "movenorth 1", "south": "movesouth 1", "east": "moveeast 1", "west": "movewest 1",
                "to": "OBJECT"
            },
            "strafe": {"right": "strafe 1", "left": "strafe -1", "stop": ["strafe 0"]},
            "turn": {"right": "turn 1", "left": "turn -1", "stop": ["turn 0", "pitch 0"],
                     "up": "pitch -1", "down": "pitch 1"},
            "look": {"up": "look -1", "down": "look 1"},
            "pitch": {"up": "pitch -1", "down": "pitch 1", "stop": ["pitch 0"]},
            "jump": {
                "": "jump 1", "up": "jump 1", "stop": ["jump 0"], "forward": "jumpmove 1", "back": "jumpmove -1",
                "backwards": "jumpmove -1", "right": "jumpstrafe 1", "left": "jumpstrafe -1",
                "north": "jumpnorth 1", "south": "jumpsouth 1", "east": "jumpeast 1", "west": "jumpwest 1"
            },
            "crouch": {"": "crouch 1", "stop": ["crouch 0"]},
            "attack": {"": "attack 1", "stop": ["attack 0"]},
            "use": {},
            "get": {"": "OBJECT"},
            "discard": {"": "discardCurrentItem"},
            "stop": ["move 0", "jump 0", "turn 0", "strafe 0", "pitch 0", "crouch 0", "attack 0"],
            "quit": {"": "quit"}
        }

def word_similarity_score (word1, word2):
    score = model.similarity (word1, word2)
    return score

def get_best_match (word, commands_map, threshold):
    scores = []
    for commands in commands_map:
        score = word_similarity_score (word, commands)
        scores.append (score)
    keys = list (commands_map.keys())
    command = keys[np.argmax (scores)]
    return command

def get_similar_command (verb, commands_map):
    lemma_verb = verb.lemma_
    # synonyms?
    # left
    if verb.pos != VERB:
        return verb
    # get synonym?
    t = 0.5
    command = get_best_match (lemma_verb, commands_map, t)
    return command

def send_command (verb, commands_map):
    if str (verb) == "stop":
        return send_stop_command (commands_map)
    command = commands_map.get (str (verb)).get('')
    return [command]

def send_command_option (verb, option, commands_map):
    for a in option.ancestors:
        if a.pos == VERB:
            for r in a.rights:
                if r.pos == NOUN:
```

```
                if r.pos == NOUN:
                    return None
                elif r.lemma_ != option.lemma_:
                    break

    for l in option.lefts:
        if l.pos == NOUN:
            for l in l.lefts:
                if l.pos == NUM:
                    command = commands_map.get (verb).get (option.lemma_)
                    n  = w2n.word_to_num (l.lemma_)
                    commands = []
                    for i in range (n):
                        commands.append (command)
                    return commands
        else:
            break

    command = commands_map.get (verb).get (option.lemma_)
    return [command]

def send_prop_command (verb, prep, commands_map):
    if prep.lemma_ in commands_map.get ("move"):
        for r in prep.rights:
            if r.lemma_ in commands_map.get ("move"):
                c = send_command_option (verb, r, commands_map)
                if c:
                    return c

def send_object_command (verb, object, commands_map):
    # objString = getObjectString (object)
    for l in object.lefts:
        if l.pos == NUM:
            for a in l.ancestors:
                if a.pos == VERB:
                    for r in a.rights:
                        if r.pos == ADV:
                            command = commands_map.get (verb).get (r.lemma_)
                            n  = w2n.word_to_num (l.lemma_)
                            commands = []
                            for i in range (n):
                                commands.append (command)
                            return commands

    # print ("Command: TODO: ", verb, "object: ", object)
    return ("Command :TODO: " + str (verb) + " object: " + str (object))

    if verb == 'use':
        # hotkey = getHotKeyForItem (objString)
        print
    elif verb == 'attack':
        pass
    elif verb == 'grab':
        pass

def send_stop_command (commands_map):
    command = commands_map.get ("stop")
    return command

def parse_root_verb (verb, commands_map):
    malmo_command = get_similar_command (verb, commands_map)

    if DEBUG:
        print ("malmo_command: ", malmo_command)
```

```
    print ("malmo command: ", malmo_command)

commands = []
for word in verb.rights:
    if DEBUG:
        print ("word: ", word, word.pos_)
    if word.pos == CCONJ:
        c = send_command (malmo_command, commands_map)
        if c:
            commands.append (c)
    if word.pos == ADV:
        # move forward
        # move backwards
        if word.lemma_ in commands_map.get (malmo_command):
            # print (command_map[malmo_command][word.lemma_])
            c = send_command_option (malmo_command, word, commands_map)
            if c:
                commands.append (c)
    elif word.pos == NOUN or word.pos == PROPN:
        # move 1 block forward
        # left registers
        if word.lemma_ == "left" and word.lemma_ in commands_map.get (malmo_command):
            c = send_command_option (malmo_command, word, commands_map)
            if c:
                commands.append (c)
        else:
            c = send_object_command (malmo_command, word, commands_map)
            if c:
                commands.append (c)
    elif word.pos == ADP:
        # preposition object
        # move to the left
        # move to the right
        # move to
        c = send_prop_command (malmo_command, word, commands_map)
        if c:
            commands.append (c)
    elif word.pos == VERB:
        # subsequent command
        # move forward and dig
        if word.lemma_ == "stop":
            c = send_stop_command (commands_map)
            if c:
                commands.append (c)
        else:
            c = parse_root_verb (word, commands_map)
            if c:
                for c in c:
                    commands.append (c)
    return commands

def parse_string_command (string, commands_map = command_map):
    doc = nlp (string)
    commands = []
    for sentence in doc.sents:
            r = sentence.root
            c = parse_root_verb (r, commands_map)
            if c:
                for c in c:
                    commands.append (c)
    return commands

# if __name__ == "__main__":
#     command = input (": ")
```

```
#     while command.lower() != "quit":
#         commands = parse_string_command (command, command_map)

#         for c in commands:
#             print (c)

#         command = input (": ")
```

In [ ]:
```
command = "jump and then move five steps forwards"
commands = parse_string_command (nlp (command), command_map)
for c in commands:
    print (c, len (c))
```

```
['jump 1'] 1
['move 1', 'move 1', 'move 1', 'move 1', 'move 1'] 5
```

In [ ]:
```
command = "move forward three steps and then move backwards three steps"
commands = parse_string_command (command, command_map)
for c in commands:
    print (c, len (c))
```

```
['move 1', 'move 1', 'move 1'] 3
[None] 1
['move -1', 'move -1', 'move -1'] 3
```

In [ ]:
```
command = "crouch and then move left and then stop and then move right"
commands = parse_string_command (command, command_map)
for c in commands:
    print (c, len (c))
```

```
['crouch 1'] 1
['strafe -1'] 1
[None] 1
['move 0', 'jump 0', 'turn 0', 'strafe 0', 'pitch 0', 'crouch 0', 'attack 0'] 7
```

In [ ]:
```
command = "move backwards two steps and then turn left"
commands = parse_string_command (command, command_map)
for c in commands:
    print (c, len (c))
```

```
['move -1', 'move -1'] 2
[None] 1
['turn -1'] 1
```

In [ ]:
```
words = {
    "move": ["go", "step", "walk"],
    "strafe": ["sidestep"],
    "turn": ["pivot"],
    "pitch": ["view"],
    "jump": ["leap"],
    "crouch": ["duck"],
    "attack": ["strike"],
    "use": ["equip"],
    "get": ["find"],
    "discard": ["drop"],
    "stop": ["halt"],
    "quit": ["exit"]
}

padding = 10
print (f"{'Word':<{padding}} {'Word':<{padding}} {'Similarity':<{padding}}")
for word in words:
    for w in words[word]:
        similarity = word_similarity_score (word, w)
        print (f"{word: <{padding}} {w: <{padding}} {similarity: <{padding}}")
```

```
Word       Word       Similarity
move       go         0.4947884976863861
move       step       0.5192893743515015
move       walk       0.3199070692062378
strafe     sidestep   0.17268125712871552
turn       pivot      0.17803087830543518
pitch      view       0.02832110971212387
jump       leap       0.6579698920249939
crouch     duck       0.24699382483959198
attack     strike     0.4166296422481537
use        equip      0.2946905195713043
get        find       0.5191866755485535
discard    drop       0.22030499577522278
stop       halt       0.6082638502120972
quit       exit       0.2591541111469269
```

In [ ]:
```
nlp_lg = spacy.load ("en_core_web_lg")
```

In [ ]:
```
padding = 10
print (f"{'Word':<{padding}} {'Word':<{padding}} {'Similarity':<{padding}}")
for word in words:
    for w in words[word]:
        similarity = nlp_lg (word).similarity (nlp_lg (w))
        print (f"{word: <{padding}} {w: <{padding}} {similarity: <{padding}}")
```

```
Word       Word       Similarity
move       go         0.5897374760834442
move       step       0.5469602414922016
move       walk       0.49438361674912584
strafe     sidestep   0.38665418602587875
turn       pivot      0.41988964908126747
pitch      view       0.14201842049567842
jump       leap       0.6101012704815728
crouch     duck       0.33205503538954984
attack     strike     0.6477336955049335
use        equip      0.4294328014842282
get        find       0.6782121170318263
discard    drop       0.3424075704170956
stop       halt       0.47539938550231325
quit       exit       0.11366727556320882
```