Introduction:

1. A narrative chain is dataset which consists of a set of events that follow a common protagonist. The events are ordered so the events can be inferred as following a script or some kind of a story. The events must happen in order for it be a narrative chain i.e. in a set {e1, e2} e2 must happen after e1. For example:

X goes to a train station.

X buys a ticket.

X waits for the train to arrive.  
X get on train.

X sits on an empty seat.

X leaves train at destination.

X leaves train station at destination.

1. The cloze test is a test where a part of a dataset is removed and the computer is given the task to find the missing data through its own knowledge. This test is used to check a systems language skill. An example of a cloze test would be (using the same dataset from above) removing what happens after X gets on the train.

The computer has <go, train station>, <buy, ticket>, <wait, train>, <get on, train> as parameters and it should be able to give <sit, empty seat> and/or <leave, train> to pass the test.

Methods:

1. The algorithm implements a system that is designed to calculate how much information a narrative chain gives us by using the ordering of events following a protagonist in the narrative chain. By doing so, the system is expected to be able to use the knowledge it gains from analyzing the narrative chains to predict following events in the chain. It does this by “training” using a data set to see how often what events are followed by others.
2. We changed this system by using spacy to parse the events and train the dataset based on the dependencies spacy gave us. Everything else is very similar, it is still based on verb-dependencies. We still follow a protagonist based on the same heuristic as the paper.
3. I implemented the cloze test for our system where I got the dependency pairs for each verb that had the root to the protagonist in each narrative chain. By doing so I got two lists with two alternate endings for each chain. Then using the formula “pmi(E1,E2)=log\frac{P(E1,E2)}{P(E1)P(E2)}” to get the amount of information one would get by comparing each verb to the optional verb the system is choosing between. The pmi is obtained through our trained dataset in which this case was “all.json” . The option that would get higher pmi on average across the story would be chosen.  
   In the code, functions protagonist and coreferring\_pairs play an important part in extracting the necessary tokens to process. All the magic happens in the class ProbabilityTable where our tokens are turned into probabilities based on our trained data set. The unigram object returns the probability of an event occurring on its own while the bigram returns the probability of two events happening one after other. And the pmi function returns the amount of information gained by the relevant verbs in their dependency forms. After that I take the average of all the pmis per option and the one with the higher average is chosen.
4. Pseudocode:
5. Get narrative chains.
6. Parse narrative chains.
7. get protagonist of each story  
   (different heuristics:   
   1st entity

most frequent entity

combination of both

)

1. extract all verb dependency pairs with protagonist
2. make trained dataset where it is a list of events (verb dependency pairs) occurring one after the other
3. implement a way to compare verb-dependency pairs and get their pmi (information based on formula: pmi(E1,E2)=log\frac{P(E1,E2)}{P(E1)P(E2))
4. get test data for cloze test (wit halternate endings).
5. Load and parse test data and extract all verb dependency pairs
6. Pass each pair through pmi against the optional event to get list of pmis
7. Choose higher mean of pmis between alternate endings

Evaluation:

1. I am performing the cloze test on my system where we give alternate endings to the narrative chain and test the systems knowledge of narrative chains by making it predict which ending suits the chain.
2. Using ‘train.csv’ as the training dataset and ‘val.csv’ as the test dataset. Train.csv is a csv file containing a ton of narrative chains (id, title, storysentences [1-5]) while ‘val.csv’ is a file containing narrative chains but with alternative endings (id, storysentence [1-4], option1sentence, option2sentence). (Also using “all.json” which is an already trained data set)
3. Results based on “all.json” = approx. 33%
4. Results can vary A TON! Basically, showing how imperfect or rather inconsistent our system is.

Conclusion:

1. We made a system that gains knowledge of events through narrative chains and we used a cloze test to test its knowledge.
2. The system basically gets one in every three sentences based on the trainingset (all.json) and the testdata (val.csv).
3. We could increase our success rate by a lot if we handled the cases where the verb-dependency is ignored (sentences like: He is happy now.) Cases where the protagonist is not detected with one heuristic go to another heuristic and so on.