# Voice Recognition RPG

Baron Khan

# Adding Voice Commands to a Game

- Commands to attack with a sword:
  - "attack with a sword"
  - "hit with something sharp"
  - "use a sword to fight"
  - "launch an assault with the sword"
  - "obliterate the enemy with a long weapon“
- Commands to heal the player:
  - "heal"
  - "recover"
  - "rest"
  - "heal with a potion"
  - "regenerate using an elixir"

# 3 years ago…

```
switch(input) {
  case "attack with a sword":
  case "hit with something sharp":
  //...
  case "obliterate the enemy with a pointy weapon":
     attackWithWeapon();
     break;
  case "heal":
  case "recover":
  //...
  case "rest":
     heal();
     break;
  case "heal with a potion":
  case "recover with a potion":
  //...
  case "regenerate using an elixir":
     healWithPotion();
     break;
  //...
```

Ad infinitum…

* Strings can be evaluated with a switch statement since Java 7.

# 1 year ago…

("attack" | "hit") . "with" . ["a"] . ("sword" | "blade")

# 1 year ago…

("attack" | "hit") . "with" . ["a"] . ("sword" | "blade")

Several minutes later…

("attack" | "hit" | "obliterate" | ("launch" . "an" . "assault")) . ("with" | "using") . ["a"] . ("sword" | "blade" | ("something" . ("pointy" | "sharp")))

An expression for each intent in the game

# Now…

- CSV file:

| | **attack** | **heal** |
|---|---|---|
| *default* | AttackDefault | HealDefault |
| *weapon* | AttackWeapon | |
| *weapon-sharp* | AtkWeaponSharp | |
| *weapon-blunt* | AtkWeaponBlunt | |
| *healing-item* | | HealWithItem |

# Voice Recognition RPG project

- Create a text-based role-playing game controlled using voice commands
- Reduce developer workload as much as possible

- Three key areas for reducing workload:
  1. **Adding voice commands without hard-coding every acceptable phrase**
  2. Automatically assign physical properties to objects
  3. Generating new rooms in the game without manually placing objects

# Motivation

- Online APIs such as Dialogflow and IBM’s Watson Conversation can be used to easily add commands

A Star Trek VR game using IBM’s Watson Conversation

# Motivation

- Online APIs such as Dialogflow and IBM's Watson Conversation can be used to easily add commands

☹ 1 request = $$$$$

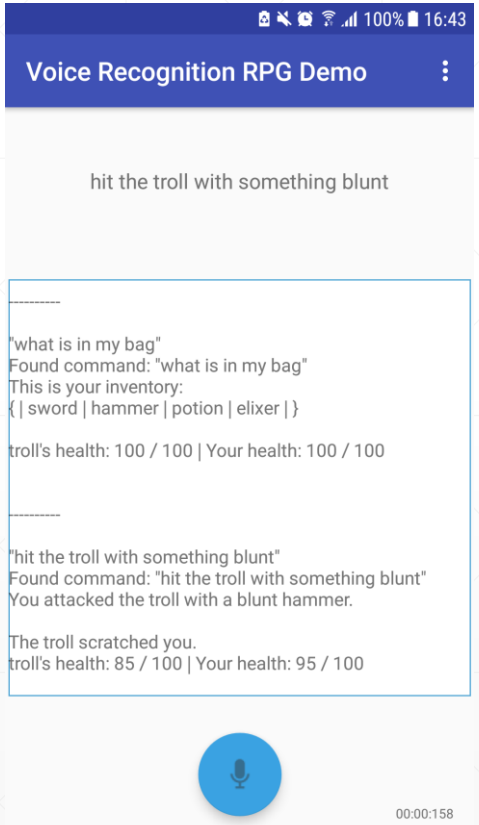
☹ Internet connection required

☹ Privacy?

A Star Trek VR game using IBM's Watson Conversation

# RPG Demo

- Two different gameplay styles:
  - *Overworld* Mode
    - Exploration – interacting with objects
    - Examples: Zork, point-and-click adventure games
  - *Battle* Mode
    - Turn-based – fighting enemies
    - Examples: Pokémon, Final Fantasy



# How it Works: Context-Action Maps

- Overworld Mode:

action name

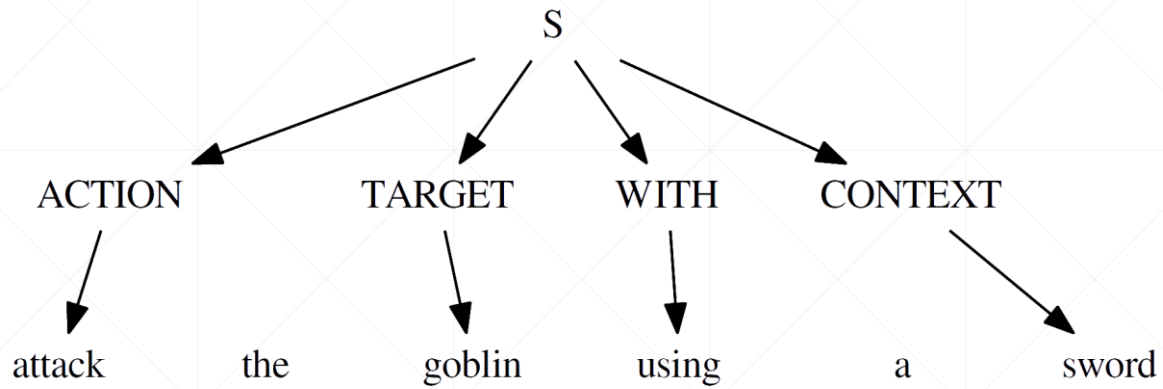
context type

| | **look** | **show** | **cut** | **break** | **grab** | **open** |
|---|---|---|---|---|---|---|
| *default* | LookDefault | ShowDefault | CutDefault | BreakDefault | GrabObject | OpenObject |
| *weapon* | | | CutWeaponNotSharp | BreakWeaponNotBlunt | | |
| *weapon-sharp* | | | CutWeaponSharp | BreakWeaponNotBlunt | | |
| *weapon-blunt* | | | CutWeaponNotSharp | BreakWeaponBlunt | | |
| *vision-item* | LookDefault | | | | | |

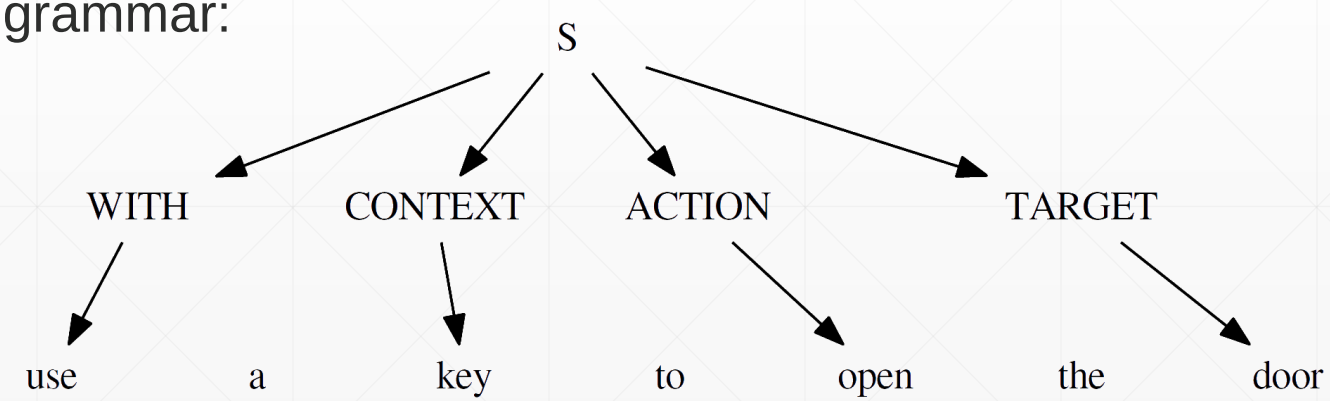
- Battle Mode:

| | **attack** | **heal** | **show** | **look** |
|---|---|---|---|---|
| *default* | AttackDefault | HealDefault | ShowDefault | LookDefault |
| *weapon* | AttackWeapon | | | |
| *weapon-sharp* | AttackWeaponSharp | | | |
| *weapon-blunt* | AttackWeaponBlunt | | | |
| *healing-item* | | HealItem | | |

# How it Works: Slot-Filling

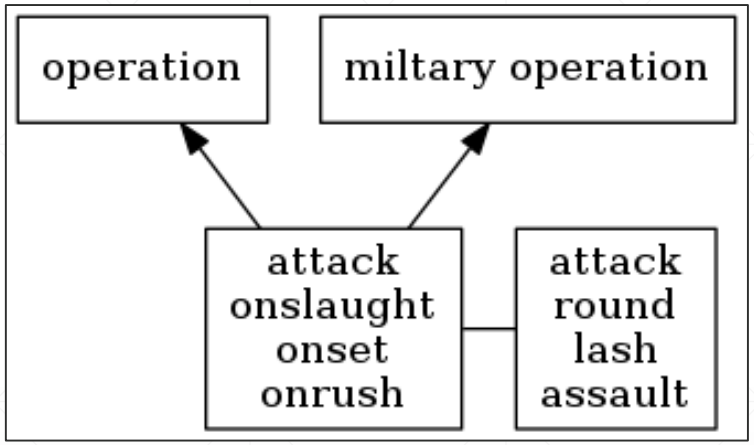



- Alternative grammar:



# Heart of the System: WordNet

- Created by Princeton University
- Large lexical database of English words
- Forms hyper-tree of words
  - Each node is a set of synonyms (*synset*)
  - Parent nodes: hypernyms




- Used to calculate semantic similarity between two words

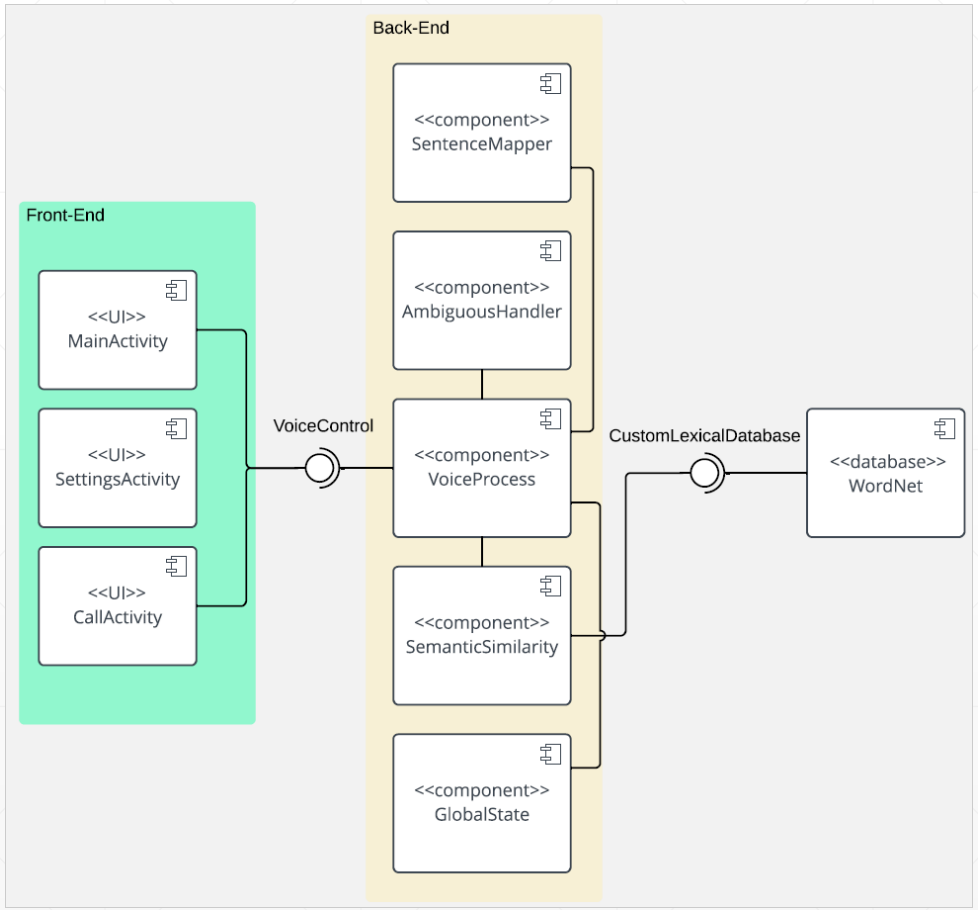
# Semantic Similarity Methods

- Algorithms to calculate similarity of two words using WordNet
- Numerous algorithms
  - Path-based (Wu and Palmer, Leacock and Chodorow)
  - Information Content / Sense Frequency (Lin, Resnik)
  - Overlaps in Definitions (Lesk)

$$sim_{wup} = \frac{2 * depth(LCS(w_1, w_2))}{depth(w_1) + depth(w_2)}$$

- Most implementations provided by WS4J library
- Some implemented manually (COS, FAST LESK)

# System Architecture



# System Features Improve Accuracy

- Confirmation and suggestions on ambiguous intents
- Chaining multiple commands in one utterance
- Detect commands with multiple targets or contexts
- Synonym-mapping
- Ignoring incorrect matches
- Sentence-matching

# Sentence-Matching

- If slot-filling grammar fails (i.e. not imperative)
- For questions, greetings, etc.
  - “What actions can I do?”
  - “Hello, how are you?”
- Cosine similarity of sentences

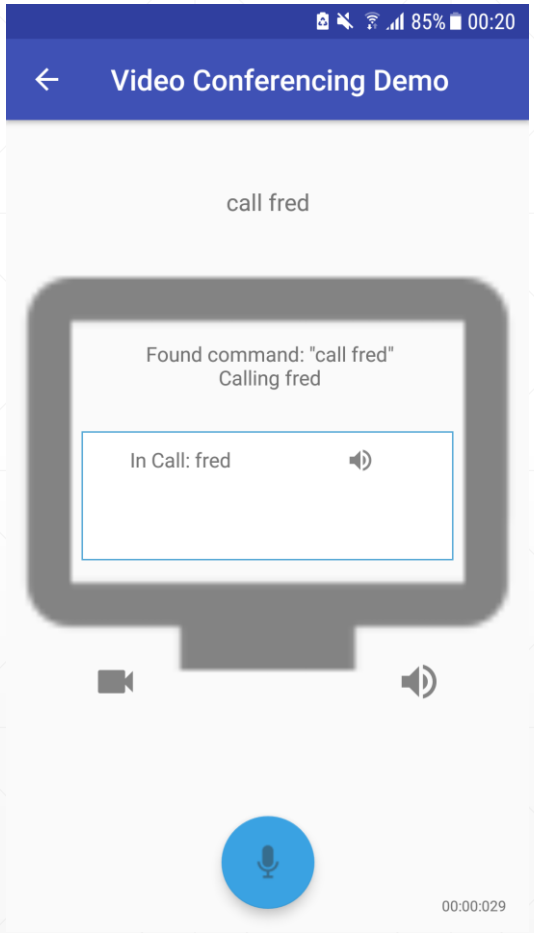
```
addSentenceMatch(new ShowDefault(), "inventory",
        "what is in my inventory",
        "what are the contents of my bag",
        "what items do i have"
);

addSentenceMatch(new ShowDefault(), "actions",
        "what can i do",
        "what are my actions",
        "what are the commands",
        "what action can i do",
        "what are my options"
);
```

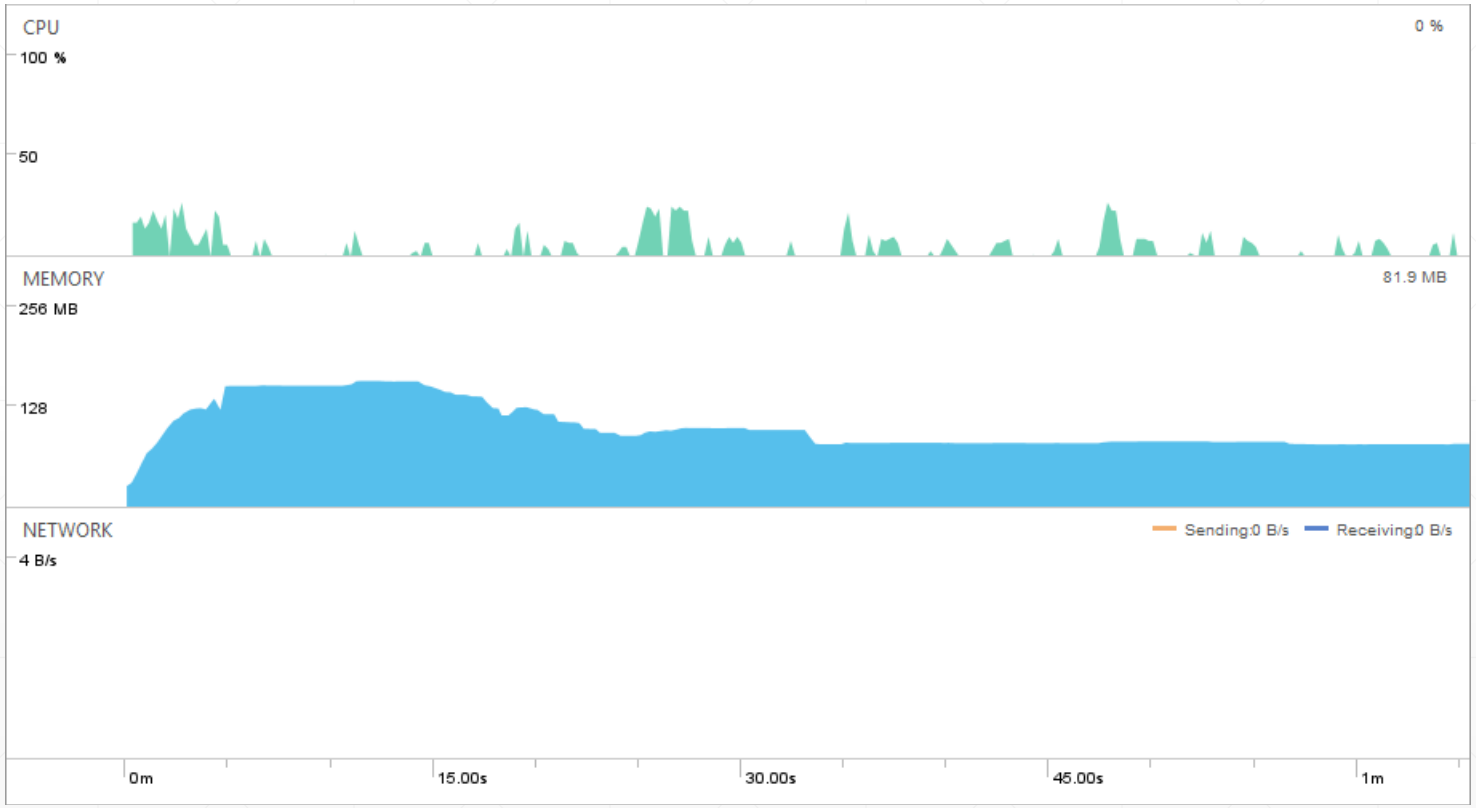
# Applied to Other Domains

- Video Conferencing commands
  - “call fred and jane”
  - “mute my video”

- Cooking commands
  - “boil the eggs”
  - “use a spoon to stir the soup”



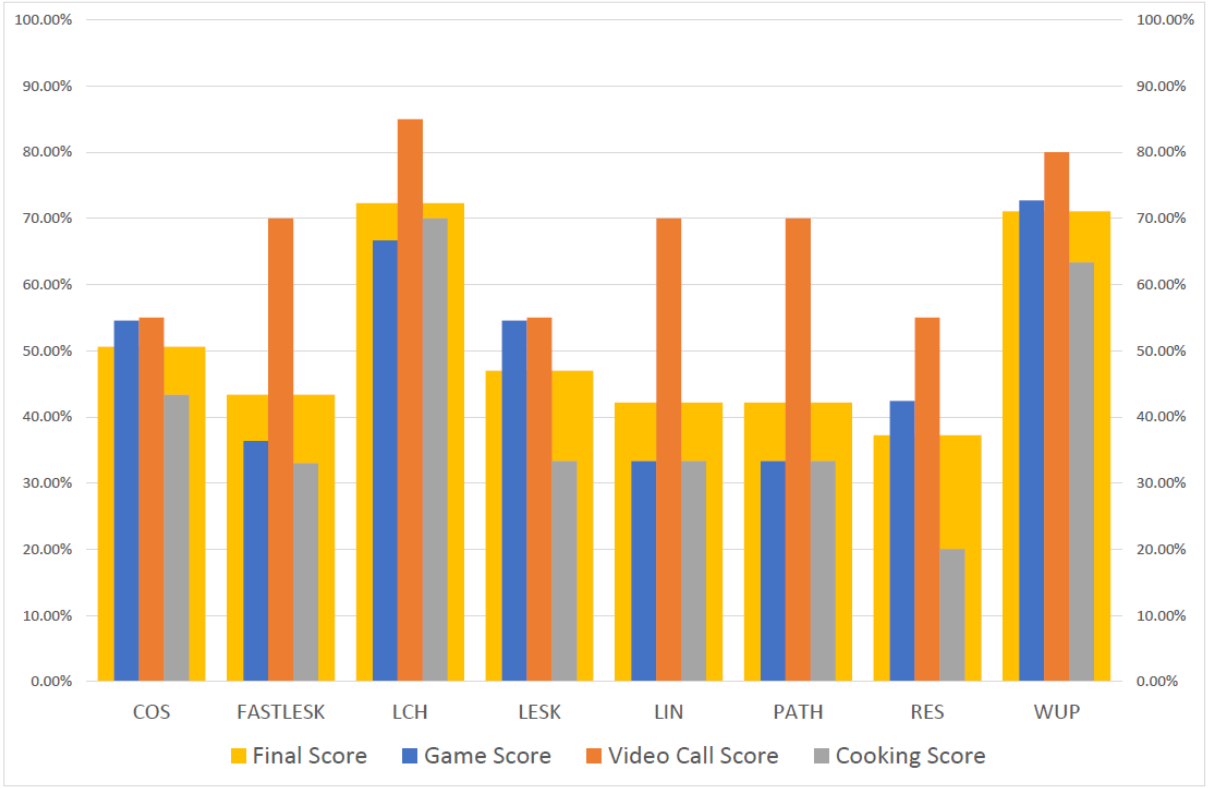
# Performance of Application



# Evaluation of Semantic Similarity Methods

- Path-based methods have highest accuracy and speed
  - Wu and Palmer (WUP)
  - Leacock and Chodorow (LCH)
- Different methods perform better in different domains
- WUP method chosen for RPG demo

# Correctness of Semantic Similarity Methods



# Performance of Semantic Similarity Methods

| Method | Average Time per Command / s | | |
|---|---|---|---|
| | *PC* | *Phone* | *Raspberry Pi* |
| COS | 0.023 | 0.85 | 6.40 |
| FASTLESK | 0.012 | 1.5 | 5.00 |
| LCH | 0.004 | 0.099 | 3.29 |
| LESK | 1.35 | 280.02 | 2068.81 |
| LIN | 0.009 | 0.29 | 5.19 |
| PATH | 0.004 | 0.088 | 2.96 |
| RES | 0.007 | 0.12 | 3.29 |
| WUP | 0.015 | 0.16 | 5.15 |

- Three key areas for reducing workload:
  1. Adding voice commands without hard-coding every acceptable phrase
  2. **Automatically assign physical properties to objects**
  3. **Generating new rooms in the game without manually placing objects**

# Room Generation from Text

- Text description of room → Java source file for room
- Use semantic similarity engine to find similar objects in text description
- Binary relationships between two objects as conditionals

There is a *table in the middle of the room.
An *armchair is underneath the table.
A *potion is on the table.
A *knife is with the potion.



```java
package com.khan.baron.voicerecrpg.game.rooms;
/* TODO: insert object imports */

public class RoomPuzzle extends Room {
    public RoomPuzzle() {
        super();
        addDescriptionWithObject(
            "There is a table in the middle of the room.",
            new GlassTable());
        addDescriptionWithObjectCond(
            "An armchair is underneath the table.",
            "An armchair is in the room.",
            new Chair(),
            () -> getRoomObjectCount("table") > 0);
        addDescriptionWithObjectCond(
            "A potion is on the table.",
            "A potion is now on the floor.",
            new Potion("potion"),
            () -> getRoomObjectCount("table") > 0);
        addDescriptionWithObjectCond(
            "A knife is with the potion.",
            "A knife is in the room.",
            new Weapon("knife"),
            () -> getRoomObjectCount("potion") > 0);
    }
}
```

# Summary

- Created prototype for a voice-controlled, text-based RPG on Android
- Created offline voice recognition system using WordNet
  - Created standalone Java library
- Evaluation of different semantic similarity methods
- Explored other areas for improving development of RPG