必要事項を記入し，卒業論文と一緒に提出すること

卒業論文受領証　学生保管

学籍番号	C	0	1	1	2	3	3	6
氏　　名	寺田 佳輔							

	受領印
受領印を受けた後，本票を受け取り 大切に保管してください.	

卒業論文受領証　事務局保管

学籍番号	C	0	1	1	2	3	3	6	受領印
氏　　名	寺田 佳輔								
指導教員	田胡 和哉								
論文題目	人工知能ハブの 開発・利用および評価								

┌(┌^ o ^)┐ キレイに切り取ってね ┌(^ o ^┐)┐

2015
年 度

人工知能ハブの開発と評価

寺田 佳輔

田胡
研究室

[卒 業 論 文]

人工知能ハブの開発と評価

(指 導 教 員)　　田胡 和哉

コンピュータサイエンス学部 田胡研究室

学籍番号　C0112336

寺田 佳輔

[2015 年度]

東 京 工 科 大 学

卒 業 論 文

論 文 題 目

人工知能ハブの開発と評価

指 導 教 員	田胡 和哉	

提 出 日　2016 年 01 月 18 日

提 出 者

学　部	コンピュータサイエンス 学 部
学籍番号	C0112336
氏　名	寺田 佳輔

2015 年度　卒　業　論　文　概　　要

論 文 題 目

人工知能ハブの開発と評価

コンピュータ サイエンス学部	氏 名	寺田 佳輔	指 導 教 員	田胡 和哉
学籍番号 C0112336				

【概　要】

ルいず

目次

図目次

表目次

第 1 章

現状

1.1 近年の機械学習

1.1.1 人工無能

1.1.2 人工知能

1.1.3 neural network

1.1.4 DeepLearning

1.2 一般的な人工知能開発フレームワーク

1.3 一般的な人工知能の返答の流れ

第 2 章

提案

2.1 開発した人工知能の活用

今回提案するのは先ほど説明した一般的な人工知能フレームワークを用いて開発を行った人工知能を活用するためのフレームワークである．

2.1.1 知能の開発をサポートする既存フレームワーク

既存の人工知能フレームワークは，人工知能自体を作成することをサポートしている．具体例で言うと Chainer は Preferred Networks が開発したニューラルネットワークを実装するためのライブラリであり，実際の開発をすることの手助けをしている．

2.2 開発した知能を試す環境

今回提案するのは，フレームワークを用いて開発したアルゴリズムや，独自のアルゴリズムを考え，作成したプログラムを実際に動かし試す環境である．

通常，人工知能のアルゴリズムを試したいと考えた場合，そのプログラムに対して入力を与える入力の部分とその処理結果を出力する出力の部分を作成する必要がある．

作成した知能の出力結果がただ単に文字で入力して，文字で出力されれば良い場合は，準備をするのはほぼ手間が不要であるが，キャラクターとの会話などで試したい場合，非常に入出力の部分を作成するのに時間がかかってしまう．

そこで，今回は作成したアルゴリズムをすぐに試す環境を提供するフレームワークを提案する．

2.3 人工知能利用フレームワークの提案

人工知能利用フレームワークは会話や動作などの返答アルゴリズムを作成した際に，それらの作成したプログラムをフレームワーク上に適当に配置することで，状況や話題に応じて適切な作成した返答アルゴリズムが選択され Unity 上のキャラクターと会話を楽しむことができる，人工知能を利用すること

に焦点を当てたフレームワークである．

2.3.1 全体構成

この人工知能利用フレームワークの全体の構成を次の**図 2.1** に示す．

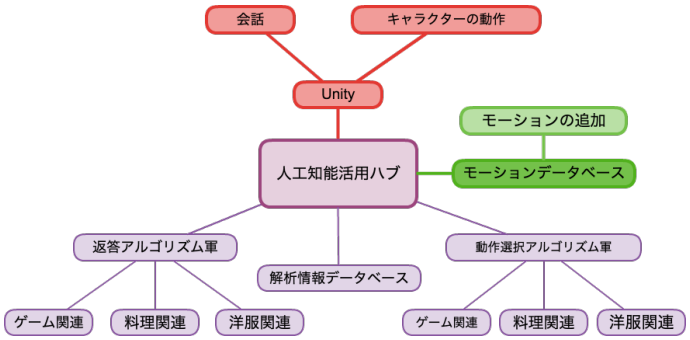

図 2.1　全体の構成図

この人工知能ハブは大きく分けて 3 つの要素で構成され，大きな流れで説明をすると Unity でユーザーが入力した内容をもとに人工知能活用ハブがその入力内容を受け取る．そして人工知能活用ハブの中で返答する内容が作り出され，モーションデータベースから適切な動作を選択し Unity へ動作と返答内容を出力する．この流れによってユーザーはキャラクターとの会話を行うことができる．

2.3.2 アルゴリズムのみを簡単に追加可能な知能ハブ

それではまずはじめに私が開発する，アルゴリズムのみを簡単に追加することができる人工知能活用ハブを提案します．

このハブでは作成した会話の返答，もしくはキャラクターの動作を選択するアルゴリズムを簡単に追加し話題によって追加したアルゴリズムの中から適切なアルゴリズムを用いて返答を行えるようにしている．

例えばゲーム関連の返答アルゴリズムを作り，試したいと考えた場合は，そのアルゴリズムを実装したプログラムをあらかじめ準備されている抽象クラスを用いて素早く作成し，**図 2.1** の返答アルゴリズム軍のプログラムに作成したプログラムを登録するだけで，ゲームの話題が来た時にそのアルゴリズムでキャラクターが返答するシステムを作ることができる．

同様にゲーム関連のキャラクターの動作を選択するアルゴリズムを作る場合は，そのアルゴリズムを抽象クラスを用いて素早く実装し，**図 2.1** の動作選択アルゴリズム軍の中に作成したプログラムを登録するだけで，ゲーム関連の会話をしている最中は，そのアルゴリズムを用いて動作を決定する仕組みを作ることができる．

はじめは，これらのアルゴリズムや人工知能が一つだけ実装されており，何も追加知能がない場合は

そのデフォルトアルゴリズムが選択されるようになっていますが，料理の話題に特化した話題解析アルゴリズムやゲームの話題に特化した感情解析のアルゴリズムが実装されることでより正確な解析が可能になるだけではなく，返答する際もゲーム専用の返答アルゴリズムなどがあることでより円滑なコミュニケーションが可能になると提案します.

様々なアルゴリズムが必要になることを考え，複数人で開発を行った際にも解析情報のデータベースによる共有などにより，よりスムーズに連携を行うことができるほか.

人工知能ハブでは，すでに解析した感情情報などの情報は全てデータベースによって共有されているので，ユーザーの入力した情報の解析を行うプログラムの開発は行わずに，すでにある感情解析プログラムの解析結果を使ってユーザーの感情状態を考慮した「会話ボット」などの開発を行うことも可能です.

2.3.3 作成したアルゴリズムを Unity ですぐに試せる機構

この人工知能利用フレームワークの人工知能活用ハブに登録された知能は Unity 上でのキャラクターとの対話ですぐに試すことができる.

共同研究者の藤井さんによると，MMD モデルを利用しているため好きなキャラクターで動作させることが出来，また，この人工知能利用フレームワークのために開発した，リアルタイムに動作を保管しながら動かす技術により，よりリアルな円滑なコミュニケーションが可能になっているという.

このように作成した人工知能をすぐにキャラクターとの対話という形で実行することができるため，入出力をどのような設計にするかや，開発はどうするかに迷うことなく，独自の対話アルゴリズムや人工知能の開発に専念することが可能になり，より高精度な対話を実現できると提案する.

2.3.4 Unity が利用可能なモーションを追加する機構

この人工知能利用フレームワークでは現在会話と動作の 2 つの出力を実装している.

ここで返答パターンは文字列で生成され，Unity で実行されるので無限のバリエーションで返答することができるが，動作（モーション）においては，現状その場で動作を生成することが難しい.

そのため，あらかじめモーションデータを作る必要があるが，そのモーション（動作）を定義するファイルを生成することは一般的には難しい，そこで共同開発の鈴木が Kinect で動作を定義し，データベースに保存，人工知能活用ハブと通信可能なプログラムを開発した.

この機構があることによって，人工知能活用ハブの中で新しい動きのパターンを追加したいとなった時にもすぐに Kinect を用いて動作ファイルを生成し，データベースに登録することで使えるようなる.

第 3 章

設計

知能ハブの構成を 3 段階に分けてと，Unity のキャラクターとの連携，解析や出力をするアルゴリズムを選定する時に利用している GoogkeAPI の流れで，人工知能利用フレームワークの構成を解説します．
最初に人工知能ハブの全体的な解析から出力の流れを示す図を以下の　　に提示します

3.1　入力された情報を解析する機構

まず初めに Unity 上のキャラクターへユーザが発言し，知能ハブが受け取った情報を解析する際の機構について解説したいと思います．

以下**図 3.1** に入力された情報が解析され，解析結果がデータベースに格納されるまでの構成を示す．

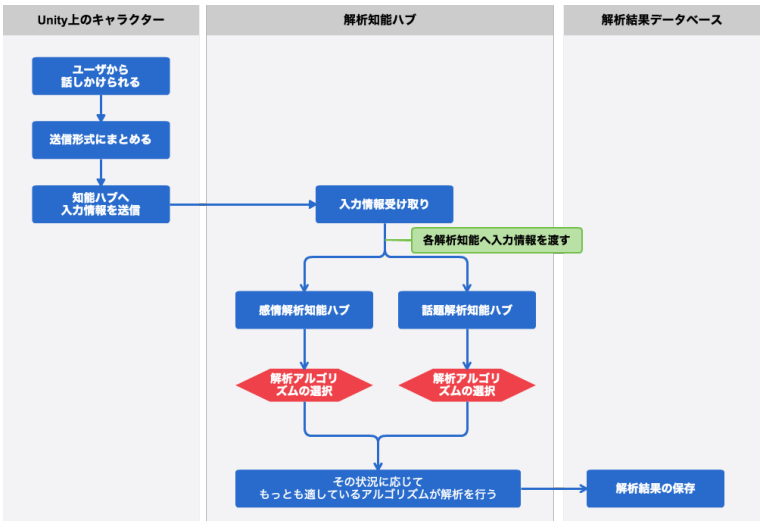

図 3.1　解析が行われるまでの図

図 3.1 の大まかな流れを説明すると，初めにユーザーが Unity 上で動作しているキャラクターに話しかけると，その内容を Unity が聞き取り，知能ハブへと送信します．送信した情報は知能ハブで受け取られ，その入力された情報は，それぞれ解析したい情報ごとに作られているハブへと渡されます．

図 3.1 の場合，感情を解析する感情解析知能ハブと話題を解析する話題解析知能ハブがあるため，入力された情報はこの 2 つの解析ハブへと送信されます．

情報が送信されると各解析ハブは入力された情報をもとに，登録されている各解析アルゴリズムの中からもっとも適切な解析アルゴリズムを選択し，実際の解析を行わせます．

実際に解析された情報は知能ハブ全体で共有されているデータベースへ保存することで，様々な場所から利用することができるようになります．

それでは以下の章で解析知能ハブの中のそれぞれの機能について説明したいと思います．

3.1.1 解析する情報別にアルゴリズムを保持する機能

図 3.1 を見て分かる通り，入力された情報は各解析する情報ごとに入力データを渡していきます．そして，各解析知能は入力された情報からその物事を解析するためのアルゴリズムを複数持っており，それを具体例を用いて表した図を以下の**図 3.2** に示します．

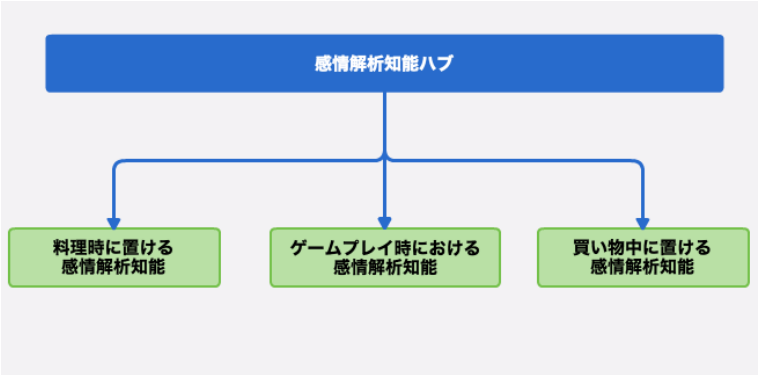

図 3.2　感情解析知能ハブとそれに付随する解析アルゴリズム

具体的に説明をすると，**図 3.2** の感情解析知能ハブは，感情を解析するための複数のアルゴリズムを持っていることになります．

各，感情解析知能ハブや話題解析知能ハブなどはアルゴリズム型の配列を持っており，その配列内に解析アルゴリズムの抽象クラスを実装したものを格納するだけで複数のアルゴリズムを保持し，適切なアルゴリズムが選択されるように設計されています．

また，解析する情報である感情や，話題といった種類はその他にも抽象クラスを実装し，解析知能ハブに登録することで追加することができます．

3.1.2　会話の話題別に解析するアルゴリズムを選ぶ機能

この人工知能ハブでは現在話している話題をもとに，どの解析アルゴリズムを選択するかを判定しています．
なので料理に関する話題をしているときは，料理関連の単語や会話に対応した解析アルゴリズムがあればそれが解析を行い，ない場合はその他の解析アルゴリズムの中でもっとも適した解析アルゴリズムが解析を行う設計になっています．
この話題を推定する際には google 検索の API を用いており，入力された内容を検索にかけてその結果から話題を推定しています．
こうすることで例えば，以下の**図 3.3**「クッパ[*1]が落ちた」という入力があったときに「クッパ　落ちた」で検索をした結果を取得します．

図 3.3　2015 年 12 月 20 日現在の「クッパ　落ちた」の Google 検索結果

図 3.3 の上位 5 件の検索結果を見るとゲームの話題であると判定され，ゲームに特化した感情解析を行うアルゴリズムが選択されます．

また，この時に「クッパ[*2]って美味しいよね」という入力があった場合，以下の**図 3.4** の「クッパ

[*1] クッパ：ゲーム「スーパーマリオブラザーズ」に登場する敵キャラクター
[*2] クッパ：クッパは韓国料理の一種。 スープとご飯を組み合わせた雑炊のような料理

美味しい」の検索上位 5 件を見て分かる通り，韓国料理のクッパの話題となるため料理に特化した感情解析を行うアルゴリズムが選択されます．

図 3.4　2015 年 12 月 20 日現在の「クッパ　美味しい」の Google 検索結果

このようにその時々に合わせて，適切な解析を行うアルゴリズムが選択されるような構造があり，これによって，より高精度な解析を行うことができます．

もし，このような機能がない場合，「クッパが落ちた」という文章は「落ちた」というキーワードから，たとえクッパという単語が料理名だと判明しても，ゲームの敵キャラクターとわからない限りはマイナスイメージな文と解析されると推測できます．

また，この両方を適切に解析できる，つまり現在の話題に限らず，感情を解析できるアルゴリズムを作成した場合は，そのアルゴリズムのみを感情解析知能ハブに登録することで確実にそのアルゴリズムが解析を行うように設定することが可能です．

3.1.3　解析アルゴリズムを簡単に追加する機能

実際に解析を行うアルゴリズム自体を簡単に追加する機能について解説します．
この，実際に解析を行うアルゴリズムの実装はあらかじめ定義されている抽象クラスを実装することで完了し，その実装の手順もソースコードの行数に換算すると最短 3 行でアルゴリズムを追加することが可能です．

図 3.2 のゲームプレイ時における感情解析知能を追加したい場合は，抽象クラスを実装後，感情解析知能ハブにある抽象クラス型を保持する配列に対して，作成した抽象クラスを拡張したプログラムを入れることで実装を行うことができます．

話題を解析する知能ハブが親の抽象クラスを実装したもので，それに付随する解析アルゴリズムは子の抽象クラスを実装したものという構図になり，その関係性を以下の**図 3.5** に示します．

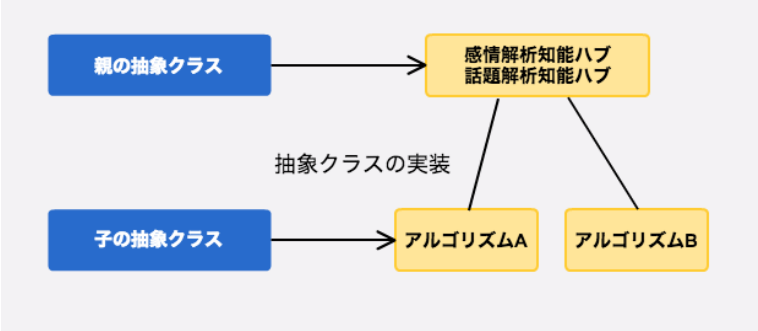

図 3.5　抽象クラスの関係と抽象クラスの実装例

図 3.5 の通り，親の抽象クラスだけでなく，子の抽象クラスに関しても簡単に実装を行い，追加する機構がある．

3.2　解析結果を保存する機構

この人工知能ハブには解析を行った情報やその他の様々な情報を保存するためのデータベースクラスが実装されている．

そして，このデータベースクラスはすべてのクラスで共有で利用できるように，すべての解析知能や出力を作成する知能の抽象クラスに含まれている．

3.2.1　解析情報を保存する機能

このデータベースは，解析した情報を保存する機能がある．
しかし，情報を保存するにあたり，解析を行うアルゴリズムを作る人によって解析結果の形式が異なることが予想できるため，どのようなオブジェクトでも保存が可能なように object 型を利用している．

実際に保存を行う場合は各解析アルゴリズム内でデータベースオブジェクトのメソッドに対して保存したい内容を引数で渡すだけで保存を行うことができる．

保存する際に付けられる名前は明確性と同一名のデータが存在しないように，その解析アルゴリズムのプログラム名＋データの形式という形で保存する．

例えば Mode-Topic-Game というゲーム話題解析知能が文字列で話題を保存したい場合はそのアルゴリズムの中で，解析が終わった時にデータベースオブジェクトの保存を行うメソッドに対して値を渡す．

そして，その保存した情報に対して Mode-Topic-Game-String という名前をつけることで明確性と同一名のデータが存在しないようにしている．

3.2.2　解析情報を取得する機能

その次に解析した情報を出力内容を作成するアルゴリズムの中から呼び出す方法について解説します．
実際に解析を行う際にはデータベースオブジェクトの情報取得メソッドに対して先ほどの解析情報の保存で説明した，欲しい情報の名前を指定することでその情報を取得することができる．

3.3　解析情報を元に出力内容を作成する機構

解析された情報を元に Unity のヤラクターに送信する出力内容を作成する工程についてユーザーが Unity 上のキャラクターとの会話をする例を表した**図 3.6** を用いて説明します.

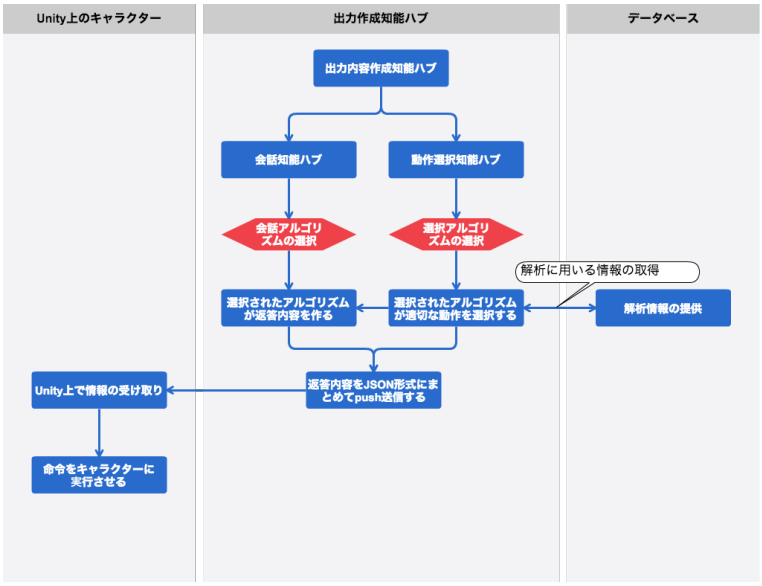

図 3.6　出力情報を作成するまでの流れ

図 3.6 を見て分かる通り出力作成知能ハブにも出力したい情報ごとにアルゴリズムを保持する機構があり，今回の**図 3.6** の場合は会話を行うための会話知能ハブとキャラクターの動作を選択するための動作選択知能ハブの 2 つがある.

それぞのハブでは入力された情報を元に，解析知能ハブの時と同じように最適なアルゴリズムを選択します，選択されたアルゴリズムはそれぞれデータベースにある，利用したい情報を取得し，返答内容や動作を決定します.

それぞれのアルゴリズム処理結果は Unity への命令形式である JSON 形式にまとめられ，websocket 通信で Unity へと送信され，Unity が命令を解釈，キャラクターが動作するという流れになります.

3.3.1　返答を行うタイミング

人工知能ハブが返答を行うことができるタイミングは 2 種類あります.
1 つめは相手から入力があった場合の返答，2 つ目が自ら発言する場合に返答する場合です.

相手から入力があった場合の返答は Unity から情報が送信されてきた時に出力を作成する知能ハブを呼び出すことで出力内容を作成し，返答を行っています.

自ら発言を行う場合は実装を行ってあるタイマーを利用して発言を行います．特定の時間や変数の値になった時に発言を行うように設定することが可能であり，出力内容作成知能ハブの自発的に発言する出力内容を作成するメソッドをそのタイミングで呼び出すことが可能になっています．

感情値や時間経過，状況の変化のあった時に websocket を用いて通信を Unity へ送信できるため，自発的に発言しているように見せることが可能です．

また，自ら発言する場合と，返答を行う場合でアルゴリズムが異なることが多いため，**図 3.5** のアルゴリズムを実装するための子の抽象クラスには返答する際のアルゴリズムと自ら発言する際のアルゴリズムを書くメソッドが用意されています．

3.3.2 会話の話題別に返答アルゴリズムを保持する機能

返答を行う際も，解析を行うときと同様に話題別に返答アルゴリズムを保持しています．

また，返答アルゴリズムを保持する仕組みに関しても同じく，返答アルゴリズム型の配列を持っており，その配列内に返答アルゴリズムの抽象クラスを実装したものを格納するだけで複数のアルゴリズムを保持し，適切なアルゴリズムが選択される部分についても同じ仕組みで動いています．

3.3.3 会話の話題別に返答アルゴリズムを選ぶ機能

出力情報ごとにアルゴリズムを保持しているため，解析知能ハブの時と同じように，その時々に合わせて最適なアルゴリズムが解析を行うようになっています．

3.4　作成した知能を Unity で試す機構

作成したアルゴリズムをすぐに実行し，試す環境として今回は統合開発環境を内蔵し、複数のプラットホームに対応するゲームエンジンである Unity を採用しました.

このゲームエンジンを用いることでウェブブラウザ上で動作するキャラクターを簡単に作成することができ，ブラウザ上で動作するため，様々なプラットフォームで試すことができます.

また，ブラウザを搭載していないデバイスの場合でも Unity 自体が複数のプラットフォームに対応しているため，様々な人が開発したアルゴリズムをすぐに試すことができます.

3.4.1　UnityWebPlayer での出力について

今回作成した人工知能利用フレームワークでは UnityWebPlayer を用いてブラウザ上でキャラクターとのコミュニケーションを取れるように設計しました.

図 3.7　Unity Web Player によるキャラクターの表示画面

図 3.7 のようにブラウザを搭載している PC や mac などのデバイスならばキャラクターを表示することが出来，作成したアルゴリズムをすぐに試すことが可能です.

3.4.2　Unity との連携に利用する WebSocket

Unity との通信には Web Socket を用いています.

web socket とはウェブサーバーとウェブブラウザとの間の通信のために規定を予定している双方向

通信用の技術規格であり，それを採用した理由としてあげられるのが任意のタイミングでの push 通知が可能な点です.

push 通知が可能になることによって，人工知能利用フレームワークから好きなタイミングで命令を送信し，Unity 上のキャラクターを動作させることができるようになるだけではなく，Unity 側のプログラムとしても命令がきた時にだけキャラクターを動作させ，ユーザーから入力があった時だけサーバへ入力情報を送信すればよいので処理が軽減されるという利点もあります.

3.4.3 Unity への送信フォーマットと作成

図 3.6 の「返答内容を JSON 形式にまとめて push 送信する」という部分の解説を行います.
この Unity への送信は汎用性の高い JSON 形式[*3]を用いて送信を行っています.

この JSON 形式のデータを実際に作成しているのは**図 3.6** の出力知能作成ハブであり，各それぞれの返答内容作成知能ハブや動作選択知能ハブから受け取った情報をまとめて JSON 形式にしています.

3.4.4 Unity からの受信フォーマット

Unity から情報を受け取る際にも JSON 形式を用いており，現在はユーザーが発言した内容を取得している.

[*3] JSON 形式：軽量なデータ記述言語の 1 つ

3.5 アルゴリズムを選定する際に用いる GoogleAPI

上記で説明したアルゴリズムを選定する際に用いている GoogleAPI について解説します.

3.5.1 GoogleAPI について

この GooogleAPI は独自に開発したプログラムであり，HttpClient を用いてウェブ上から Google の検索結果を取得し，その結果を HttpCleaner で整形している.
その後整形した情報を形態素解析にかけて，固有名詞のみを抽出し，固有名詞の単語票を作っています.

3.5.2 GoogleAPI の有効性

Google の検索結果を用いることで，常に最新の検索ワードに関するキーワードが手に入ります.
そのほかにも一見「料理」という単語と「ごはん」という単語には関連性がないように見える単語も，この 2 つの検索結果の単語票を比較することで同じ分野の単語であることを知ることができます.

第 4 章

実装

4.1 開発環境

4.1.1 Java の利用

今回の開発では実行が高速かつ，オブジェクト指向が今回開発する人工知能フレームワークに適していると判断したため Java を用いて開発を行った．
また，当研究室に所属する学生は Java の開発に慣れており，学習コストが低いため採用しました．

4.1.2 Maven フレームワーク

Unity との通信を行うため Maven フレームワークを用いて開発を行った．
なおかつ，ライブラリのバージョン管理をソースコードで行えるため，引き継ぐ際の学習コストが低いことが利点である．

4.2 解析部分の実装

4.2.1 解析コントローラー

この解析コントローラーではすべての解析を行う分野を管理するクラスであり，解析話題別に解析アルゴリズムを保持する解析知能ハブを作るために InputController を作成しました，それに実装した主要なメソッドなどを以下の**表 4.1** に示します．

表 4.1　実装した主要メソッド

コンストラクタ	各解析知能ハブを登録する
InputData	入力があった時に各解析知能ハブへデータを渡す

表 4.1 のコンストラクタでは各解析分野を登録する処理を行っている．
その部分のソースコードを以下の**リスト 4.1** に示す．

リスト 4.1　新しい解析分野を登録する際のソースコード (inCnt.java)

```
1:  public class InputController {
2:          public InputController(DataBase database) throws IOException {
3:           AnalyzeMode = new LinkedList<Abstract_Mode>();
4:           //-----入力の種類別のモード----
5:           //話題解析
6:           AnalyzeMode.add(new Mode_Topic(database));
7:           //感情解析
8:           AnalyzeMode.add(new Mode_Feeling(database));
9:           //ログの保存
10:          AnalyzeMode.add(new Mode_ChatLog(database));
11:          //-----------------------------
12:      }
13: }
```

このコンストラクタに登録された解析分野は入力があった時に呼び出され，あらかじめ抽象クラスで定義された適切なメソッドの中のアルゴリズムを記述することで解析を行うことができる．

表 4.1 の InputData メソッドではユーザーから入力があった時に入力された情報を各解析分野の実装済みクラスに値を渡すことができます．

4.2.2　解析知能ハブ

初めに解析知能ハブの例を挙げると，感情解析知能ハブや話題解析知能ハブに当たるものであり，各ハブはそれぞれ様々な解析アルゴリズムを所持しています．

解析する話題別にアルゴリズムを保持するために Abstract Mode（以下親抽象クラス）という抽象クラスを実装した．

親抽象クラスには解析する情報ごとにプログラムを保持するための機構が全て記述されており，この親抽象クラスに実装した主要なメソッドなどを次の**表 4.2** に示します．

表 4.2　Abstract Mode に実装した主要メソッド

init	初期化を行うメソッド
getAnalyzeParts	解析アルゴリズムを選択するメソッド
analyzeChat	入力があった際に解析を行わせるメソッド

表 4.2 の init メソッドでは各解析アルゴリズムがもっている話題分野に登録されている単語を Google 検索にかけて，その検索結果の頻出単語を取得しています．
表 4.2 の getAnalyzeParts メソッドでは実際に解析を行うアルゴリズムを生成された頻出単語をもとに決めるメソッドです．
具体的には各解析アルゴリズムごとに生成された頻出単語票の HashMap の単語表とユーザーが入力した文章を比較して，もっとも似ている頻出単語表を持つ解析アルゴリズムが解析を行うという実装に

なっています.

4.2.3 解析アルゴリズムの解析知能ハブへの追加

この人工知能利用フレームワークの解析アルゴリズムを簡単に実装する構成について説明します.

簡単にハブに対してアルゴリズムを追加実装するために，実際に解析アルゴリズムを実装する際に用いる，抽象クラス「Abstract Mode Parts」(以下，子抽象クラスと表記します) を実装しました.

この子抽象クラスには親抽象クラスである Abstract Mode を実装したクラスから解析する際に呼び出されるメソッドやデータベースなどとの連携をが記述されているため，アルゴリズムを試したい場合これらの部分については追記する必要がない点がメリットです.

以下の**表 4.3** に実際に解析を行う解析知能を作るために必要な抽象クラスである Abstract Mode Parts に実装した主要なメソッドなどを示します.

表 4.3　Abstract Mode の実装

クラス変数	保存を行うための変数が定義されている
コンストラクタ	アルゴリズムの分野を記述する場所
ChatAnalyze	アルゴリズムを記述する部分
saveData	解析結果を保存

表 4.3 のクラス変数は解析した情報を保存するための変数であり，解析結果をクラス変数に入れることで処理が終わった後に適切な形式でデータベースに保存される.

表 4.3 のコンストラクタは，その解析アルゴリズムの分野について記述する必要性がる，具体的に言うと about という変数に話題の名前を入れる必要がます，また，そうすることで同じ話題をユーザーが話した時にそのアルゴリズムが選択されるという構造が実装されます.
その部分のソースコードを以下の**リスト 4.2** に示します.

リスト 4.2　コンストラクタで話題を設定するソースコード (about.java)

```
1: public Mode_Topic_Ryori(DataBase database) throws IOException {
2:         super(database);
3:         about = "料理";
4: }
```

リスト 4.2 は料理に関する話題を解析するプログラムのコンストラクタをソースコードから抜粋したものですが，ここで 1 行目のプログラムを記述する.

表 4.3 の ChatAnalyze は，実際に解析アルゴリズムを書く部分である.

ここで入力された内容が String 型で引数として渡されてくるので，その内容を用いて解析を行う．
解析した情報はあらかじめ定義してあるクラス変数に保存することで，自動でデータベースに格納されるようになっている．

リスト 4.3　解析アルゴリズムを記述するメソッドのソースコード (anaAlgo.java)

```
1: public class Mode_Topic_Ryori extends Abstract_Mode_parts {
2:     @Override
3:     public boolean ChatAnalyze(String chat) {
4:         stack.push(chat);    //解析アルゴリズムを記述する
5:         return true;
6:     }
7: }
```

リスト 4.3 は解析を行うアルゴリズムを記述するメソッドのみを抜粋したもので，ここでデータベースに保存する情報を解析し，解析した情報を変数に格納するプログラムを記述します．

表 4.3 の saveData は全ての解析が終わった後に呼び出され，変数の中に値が入っていた場合のみその内容をデータベースにそのクラス名＋データ型の名前で保存します．
最後にこの抽象クラスを拡張して作成したクラスは親クラスである Abstract Mode を実装した，感情解析知能ハブなどの親抽象クラスを実装したクラスのコンストラクタに登録する必要があり，登録することでアルゴリズムの追加が完了します．

4.2.4　現在実装している解析アルゴリズム

現在実装している解析プログラムは 2 種類あり，感情の解析と話題の解析である．

話題の解析に関しては，例えばゲームの話題解析知能を作った場合，さらに細かい何のゲームかということを解析することを目的に作成した．
1 つ目に感情の解析を行うアルゴリズムの実装について説明する．

感情の解析を行うプログラムは親抽象クラスである感情解析知能ハブに所属する解析知能の 1 つで，現在感情の解析を行うプログラムはこの 1 つなため，必ずこのアルゴリズムがデフォルト解析アルゴリズムとして，選ばれるようになっています．

具体的な解析を行うアルゴリズムに関しては「哀れ」「恥」「怒り」「嫌」「怖い」「驚き」「好き」「高ぶり」「安らか」「喜び」の 10 種類の感情に分類して感情の解析をおこなっています．

このそれぞれの感情にはその感情に対応する単語が付いており，入力した文章の中にその単語があった時にその感情値に 1 を加えて数字で表現する仕組みになっています．
2 つ目の話題を解析する知能では料理とゲームに関する話題を解析するプログラムが実装してあり，料理の分野では「作る」「食べる」「片付ける」の 3 つの話題にさらに細かく解析する仕組みがあります．
また，ゲームの分野では「戦闘」「負け」「勝利」の 3 つの話題にさらに細かく解析する仕組みを実装し

ています.

4.3 データベースの実装

4.3.1 全ての解析情報を保存する機構

まず初めにデータベースは独自実装したデータベースクラスを用いて実現しました.

データベースの実装では様々な解析情報を保存する必要があるため，複数の変数型に対応するために，HashMap のキーを String にし，保存する値である value を Object 型に指定しました.

このデータベースクラスにはデータを保存するためのメソッドが用意されており，以下にソースコードを示します.

リスト 4.4　データベースの解析結果を保存するメソッド (dbIn.java)

```
1: public void setData(String className, Object value) {
2:     objList.put(className, value);
3:     acsess++;
4: }
```

リスト 4.4 のメソッド setData に対して値を保存する際は解析アルゴリズムの子抽象クラスを実装する際には記述しません.
子の抽象クラスの中であらかじめ定義されている変数に値を保存することで，親クラスが saveData メソッドを呼び出し，saveData メソッドにはこのデータベースクラスの setData に対してクラス名と値を送るように記述してあるため，解析結果のデータを保存することがができます.

4.3.2 解析した情報を取得する機構

解析した情報を取得する際はこのデータベースクラスの getData メソッドを呼び出します.
以下の**リスト 4.5** に getData メソッドのソースコードを示します.

リスト 4.5　データベースの解析結果をするメソッド (dbOut.java)

```
1: //解析済みデータを取得する
2: public Object getData(String key) {
3:   if(objList.get(key)==null){
4:         System.out.println("Databaseより出力、getData("+key+")がしっぱいしました");
5:         return null;
6:     }
7:    return objList.get(key);
8: }
```

リスト 4.5 の 2 行目を見るとデータを取得する際に String 型の鍵が必要になります.
この String 型の鍵はデータベースの中にあるデータを保存しているハッシュマップの鍵を示しており，取得したい情報の鍵を showData というメソッドを用いてデータベースの中身を見ることで調べ，その値を指定して取り出します.

4.4 出力コントローラー

4.4.1 出力情報別にアルゴリズムを保持する機構

出力する情報をまとめ，JSON 形式などを形成する出力コントローラーについて解説したいと思います.
まず初めに出力コントローラーに実装した主要なメソッド一覧を以下の**表 4.4** に示します.

表 4.4 実装した主要なメソッド

コンストラクタ	各出力知能ハブを登録する
getJson	Unity へ出力情報を送るときに呼ばれるメソッド
getTimeAction	キャラクターが自発的に発言する際に用いられるメソッド

表 4.4 のコンストラクタの部分では，各出力知能ハブを登録します.
登録された出力知能ハブは getJson メソッドが呼ばれたときに出力内容を作成するようになっています.

表 4.4 の getJson メソッドでは Unity から入力があった際に返答を行います，そのときに呼ばれるメソッドであり，あらかじめ登録されている出力知能ハブの出力を作成するメソッドを呼び出します.

表 4.4 の getTimeAction メソッドでは時間経過に応じてキャラクターが発言する設定を有効にしているときに呼び出されるメソッドであり，このメソッドが呼ばれると各出力知能ハブの自発的に発言する際に用いるメソッドを呼び出します.

4.4.2 出力知能ハブ

出力する動作や返答内容といった出力情報別にアルゴリズムを保持する機構について解説したいと思います.

この機構も情報解析の時と同じ仕組みで構成されており，この出力の情報別に保持する機構についても Abstact Mode という親抽象クラスが作成されているので，その抽象クラスを実装することで出力知能ハブである，返答知能ハブや動作選択知能ハブを作成することができます.
以下の**表 4.5** に出力知能ハブを作成するために必要な抽象クラスである Abstract Mode の主要メソッドを示します.

表 4.5　実装した主要なメソッド

init	初期化
getOutput	返答内容の作成
getTimerAction	キャラクターが自発的に発言する際に用いられるメソッド

表 4.5 の init メソッドでは初期設定を行っており，Google の検索結果データベースを最新の情報に更新をするなどの処理を行っています．

表 4.5 の getOutput メソッドでは Unity にユーザーが話しかけてきたときに返答内容を作成するアルゴリズムから出力内容を取得して，その内容を返すメソッドになっています．
また，返答するアルゴリズムを選択する機構もこの部分にあります．
その選択は以下のように HashMap を用いてユーザーが発言した内容から作成した頻出単語票である HashMap と各解析を実際に行うアルゴリズムが持っている話題をもとに作成した頻出単語票である HashMap を比較して，もっとも頻出単語票が似ているものを選ぶという仕組みになっています．

リスト 4.6　getOutput.java のソースコード (getOutput.java)

```
1:  public String getOutput() throws IOException {
2:      google = googleUpdate();
3:      int height = 0;
4:
5:      LinkedHashMap<String, Integer> result = new LinkedHashMap<String, Integer
            >();
6:      LinkedHashMap<String, LinkedHashMap> searchResult = new LinkedHashMap();
7:
8:      Stack chatLog = (Stack) database.getData("Mode_ChatLog_SaveLog_stack");
9:      String last = chatLog.peek().toString();
10:     searchResult.put(last, counter.wordcount(google.serch(last, 10000), 0));
11:     for (String key1 : searchResult.keySet()) {
12:         LinkedHashMap<String, Integer> map1 = searchResult.get(key1);
13:         for (String key2 : googleResults.keySet()) {
14:             LinkedHashMap<String, Integer> map2 = googleResults.get(key2);
15:             for (String key4 : map1.keySet()) {
16:                 for (String key3 : map2.keySet()) {
17:                     if (key3.equals(key4)) {
18:                         height += Math.min(map1.get(key3), map2.get(key3));
19:                     }
20:                 }
21:             }
22:             result.put(key2, height);
23:             height = 0;
24:         }
25:     }
26:     Abstract_Mode_parts kotaeru = etcTalker.get(0);
27:     int a = 0;
28:     int most = 0;
29:     String ansKey = "";
```

```
30:     for (String key : result.keySet()) {
31:         System.out.println(key + " : " + result.get(key));
32:         if (most <= result.get(key)) {
33:             most = result.get(key);
34:             ansKey = key;
35:         }
36:     }
37:     for(Abstract_Mode_parts tk :etcTalker){
38:         if(tk.about.equals(ansKey)){
39:             kotaeru = etcTalker.get(a);
40:         }
41:         a++;
42:     }
43:     kotaeru.dataRefresh();
44:     database.setData("google", google);
45:     return kotaeru.Action();
46: }
```

まず初めに，**リスト 4.6** の 8,9 行目で最新の発言情報を取得し，10 行めで発言内容を GoogleAPI に渡すことで頻出単語票を作成します．
次にあらかじめ作成してある各解析アルゴリズムごとの話題単語の頻出単語票と GoogleAPI を用いて取得した頻出単語票を 11 行めから 25 行めにかけて比較して，一番最適な解析アルゴリズムを選択しています．
最後に 45 行目にて選択された解析アルゴリズムに解析を行わせ，解析結果をそのまま返しています．

表 4.5 の getTimerAction メソッドは時間経過に応じて反応するときに呼び出され，実際に出力内容を作成するアルゴリズムに対して，自発的に発言する際の出力内容を作成させ，取得します．

4.4.3 出力アルゴリズムの出力知能ハブへの追加

実際に出力情報を作成するためのアルゴリズムを記述するプログラムを簡単に出力知能ハブへ追加するために出力専用の Abstract Mode Parts という抽象クラスを作成しました．

その抽象クラスを用いることで 3 行プログラムを書くだけで新しいアルゴリズムを追加できるようになっています．

それではまず初めに，その抽象クラスに実装した以下の**表 4.6** に示した主要なメソッドについて解説したいと思います．

表 4.6 のコンストラクタでは出力を行う際に担当する分野や話題について記述する部分です，解析の時と同じように変数 about に対して適切な担当する話題名を入れることで，GoogleAPI を用いてその話題名に関する頻出単語票が自動で生成されます，その生成された頻出単語表は先ほど説明したアルゴリズムの選定に利用されます．

表 4.6　実装した主要なメソッド

コンストラクタ	担当分野の設定
Action	返答アルゴリズムの
TimeAction	キャラクターが自発的に発言する際に用いられるメソッド
dataRefresh	常にデータベースを最新に保つためのメソッド

以下の**リスト 4.7** に挨拶の分野を指定する場合のプログラムの例を示します.

リスト 4.7　話題を指定する際のサンプルソースコード (aisatu.java)

```
1: public Abstract_Mode_parts(DataBase database) throws IOException {
2:     this.database = database;
3:     about = "挨拶";
4: }
```

リスト 4.7 では話題を挨拶に指定しており，ユーザーが挨拶と関係のある単語を発話した時にこのアルゴリズムが選択され，実際に返答内容を作成します.

4.4.4　現在実装している出力アルゴリズム

現在実装している出力知能ハブは 2 つあり，会話を行う知能ハブと動作選択を行う知能ハブの 2 種類です.

会話を行う知能ハブには 2 つのアルゴリズムが搭載されており，料理とゲームの話題に関するアルゴリズムを実装しました.

料理出力のアルゴリズムでは，解析した時に取得した作る，食べる，片付けるの状態を用いて，返答を行います.
また，ゲームのアルゴリズムでは戦闘，負け，勝利の状態を解析しているのでそれを用いて返答を行っています.

次に動作を選択するアルゴリズムでは共同開発の鈴木さんのデータベースから動作一覧，その 1 つ 1 つの動作に関係する単語，その関係する単語を GoogleAPI をもちいて頻出単語表を作ったものを取得します.

動作を選択するアルゴリズムはその動作に関連付けられている頻出単語表と，ユーザーの発言内容から作成した頻出単語表を比較して，最も関連性のあるモーションを選択するようになっています.

その具体的なアルゴリズムや通信に関しては 4.6 に記述します.

4.5 Unity との通信の実装

4.5.1 通信方式

Unity との通信には WebSocket を用いており，双方向任意のタイミングでの情報の送受信が可能となっています.
Unity との情報の送受信を行うために人工知能利用フレームワークの中に送受信を行うためのクラスである，NewWSEndpoint を実装しました.

以下の**リスト 4.8** にソースコードを示します.

リスト 4.8　動作選択アルゴリズムの一部抜粋 (endpoint.java)

```
1:  @ServerEndpoint("/endpoint")
2:  public class NewWSEndpoint {
3:      public NewWSEndpoint() throws IOException {
4:          this.timer = new Timer(1000, TimerAction);
5:          this.TimerAction = new TimerTick(outputter, database);
6:          this.outputter = new OutputController(database);
7:          this.inputter = new InputController(database);
8:          TimerAction.setController(timer);
9:          timer.start();
10:     }
11:
12:     private static ArrayList<Session> sessionList = new ArrayList<Session>();
13:     DataBase database = new DataBase();
14:
15:     InputController inputter;
16:     OutputController outputter;
17:     TimerTick TimerAction;
18:     Timer timer;
19:
20:     @OnOpen
21:     public void onOpen(Session session) {
22:         System.out.println(session.toString()+"が接続しました");
23:         sessionList.add(session);
24:     }
25:
26:     @OnClose
27:     public void onClose(Session session) {
28:         System.out.println(session.toString()+"が切断されました");
29:         sessionList.remove(session);
30:     }
31:
32:     @OnMessage
33:     public void onMessage(String msg) throws IOException {
34:         System.out.println(msg);
35:         inputter.InputData("{'mode':'1','message':'" + msg + "','bui':''}");
36:         String json = outputter.getJson();
37:         System.out.println("クライアントへ送る：" + json);
38:         for (Session session : sessionList) {
```

```
39:                 session.getBasicRemote().sendText(json);
40:             }
41:         }
42:
43:         @OnMessage
44:         public void processUpload(byte[] b) throws UnsupportedEncodingException,
                IOException {
45:           System.out.println("Unityからの入力");
46:           System.out.println(b);
47:           String result = new String(b, "UTF-8");
48:           inputter.InputData(result);
49:           String json = outputter.getJson();
50:           for (Session session : sessionList) {
51:             session.getBasicRemote().sendText(json);
52:         }
53:     }
```

4.5.2 Unity からの入力情報の受信

リスト 4.8 のソースコードでは Unity（クライアント）が接続を行った時にそのセッションを保存するために 23 行目でリストにセッションを保存しています.

セッションが確立され，受信を行う準備が完了しました.

Unity からメッセージが来た場合は**リスト 4.8** の 44 行目の processUpload が呼ばれます．また 33 行目には同じく受信するメソッドである OnMessage というメソッドがありますが，これは Unity 以外の端末からメッセージを受け取った際に呼ばれます.

processUpload メソッドではバイト形式で入力情報を受け取るため，47 行目にてバイトを String 型に変換する必要があります.
変換した後は 48 行目でその値を入力情報の解析を行う解析知能ハブへ渡し，その処理が終わった後の 49 行目で返答する値を生成，作成した json 形式の出力情報をもらい 51 行めで全ての接続クライアントに対して命令を送信します.

最後にこれ以上通信を行わない場合は**リスト 4.8** の 27 行目で，セッションが切断された時にリストから削除する処理を行っています.

4.5.3 Unity への命令の送信

リスト 4.8 の 51 行目で全ての接続クライアントに対して出力知能ハブから得た json 形式の値を送信しています.

また，送信する先が全てのクライアントに対して送信することになっていますが，これは人工知能利用フレームワークが家庭で利用されることを目標にしているからです.

例えばテレビのブラウザでキャラクターと対話，PC の画面でキャラクターと対話，スマホの画面でキャラクターと対話を行った際に全てのデバイスから同じキャラクターと対話することができるということを実現するために 51 行目では全てのクライアントへ対して情報を送信しています．

4.6 人工知能利用フレームワークに追加したモーションの利用

4.6.1 動作選択アルゴリズムの実装

先ほど 4.4.4 で説明した動作選択を行う際に用いているアルゴリズムについて解説します．

そのため，以下に動作を選択する際に用いている動作選択アルゴリズムを一部抜粋したものを示します．

リスト 4.9 動作選択アルゴリズムの一部抜粋 (motion.java)

```
1:  public class Mode_Motion_Nomal extends Abstract_Mode_parts {
2:      @Override
3:      public String Action() throws IOException {
4:          String motion = match();
5:          return motion;
6:      }
7:
8:      private String match() {
9:          GoogleApi google = (GoogleApi) database.getData("google");
10:         LinkedHashMap<String, LinkedHashMap> googleResults = google.
                getGoogleResultALLData();
11:         HashMap<String, Integer> last = new HashMap<String, Integer>();
12:
13:         for (String key : googleResults.keySet()) {last = googleResults.get(key
                );}
14:
15:         MongoList MList = new MongoList();
16:         MList.SplitArray();
17:         HashMap<String, HashMap<String, HashMap<String, Integer>>>
                SuzukiDatabase = MList.getdbMap();
18:
19:         int score = 0;
20:         String name = "";
21:
22:         for (String key : SuzukiDatabase.keySet()) {
23:             int thisscore = 0;
24:             HashMap<String, HashMap<String, Integer>> serchwords =
                    SuzukiDatabase.get(key);
25:             for (String key2 : serchwords.keySet()) {
26:                 System.out.println("     " + key2);
27:                 HashMap<String, Integer> hikaku = serchwords.get(key2);
28:                 for (String key3 : hikaku.keySet()) {
29:                     System.out.println(key3+":"+last.get(key3));
30:                     if (last.get(key3) != null) {
31:                         thisscore = Math.min(last.get(key3), hikaku.get(key3));
32:                     }
33:                 }
```

```
34:                 }
35:                 if (score <= thisscore) {
36:                     name = key;
37:                     score = thisscore;
38:                 }
39:             }
40:             return name;
41:         }
42: }
```

まず初めに，**リスト 4.9** の 3 行目に動作選択が行われる際に呼び出される Action メソッドが記述されています，このメソッドでは 8 行目に記述されたメソッド match を呼び出し，その中で適切なモーションを選択しています.

リスト 4.9 の match メソッドでは 15 行目から 17 行目にかけてデータベースを管理するコントローラーからデータベースの情報を取得しています，このコントローラーがサーバー上の MongoDB からモーションデータなどの情報を取得しています.
また，このコントローラの実装に関しては共同開発の鈴木さんの論文を参照してください.

次に実際にどの動作を実行するかを判定しているアルゴリズムですが**リスト 4.9** の 22 行目から 39 行目をみてください，そこではユーザーが発言した内容から作成した頻出単語表とデータベースの中にあるモーションごとに関連付けられている頻出単語表を比較しています.

その 2 つの頻出単語表の比較結果の中で一番同じ単語が多く出現した動作が選択され，40 行目でその動作名が返される仕組みとなっています.

4.7 GoogleAPI による頻出単語表の作成

今回実装した GoogleAPI では Google 検索を用いてウェブ上から情報を取得する機能から kuromoji を用いて形態素解析を行わせる機能，頻出単語表を作成する機能と 3 つの機能から成り立っている.

4.7.1 形態素解析による検索ワードの作成

ユーザーが入力した情報を Google 検索を用いて検索し，どのような分野の単語なのかということを調べるにあたり，その検索する際のキーワードというものは非常に大切である.

そのため Java の形態素解析器である kuromoji を用いて形態素解析を行い，適切な検索ワードを指定できるように実装を行った.

具体的には，kuromoji には複数のモードがあり，Search モードを利用することで「日本経済新聞」を「日本 — 経済 — 新聞」のように検索で利用しやすい形に分解してくれる機能があるのでその機能

を用いて入力を行った情報をそのまま検索するのではなく形態素解析を行ってから検索を行っている．

4.7.2 GoogleAPI を利用して検索結果を取得

検索をかける際には HttpClient を用いて検索を行っています，HttpClient は POST 通信を用いてサーバーへ接続を行い，XML 形式で結果を受け取るもので，これを用いることで google の検索結果をそのまま取得することができる．

以下の**リスト 4.10** に実装を行った GoogleAPI のソースを一部抜粋したものを記載する．

リスト 4.10　GoogleAPI の一部抜粋ソースコード (google.java)

```
1:
2:  public class GoogleApi {
3:
4:      DataBase database;
5:      LinkedHashMap<String, LinkedHashMap> googleResultsAllData = new
            LinkedHashMap<String, LinkedHashMap>();
6:      TextWriter tw;
7:
8:      public GoogleApi(DataBase database) throws IOException {
9:          this.tw = new TextWriter();
10:         googleResultsAllData = tw.getHashMap();
11:         this.database = database;
12:     }
13:
14:     public LinkedHashMap<String, LinkedHashMap> getGoogleResultALLData() {
15:         return googleResultsAllData;
16:     }
17:
18:     public String serch(String str, int kensu, int mode) throws
            UnsupportedEncodingException, IOException {
19:         // Googleの検索用URL
20:         String url = "http://www.google.co.jp/search?&num=" + kensu + "&q=";
21:
22:         //------------------------------検索用キーワード生
                成----------------------------------
23:         Tokenizer.Builder builder = Tokenizer.builder();
24:         builder.mode(Tokenizer.Mode.SEARCH);
25:         Tokenizer search = builder.build();
26:         List<org.atilika.kuromoji.Token> tokensSearch = search.tokenize(str);
27:         String SearchString = "";
28:
29:         //形態素解析を用いて適切な検索を行えるようにする
30:         for (org.atilika.kuromoji.Token token : tokensSearch) {
31:             if (!token.getPartOfSpeech().contains("助
                    詞") && !token.getPartOfSpeech().contains("助動詞")) {
32:                 SearchString += token.getSurfaceForm();
33:                 SearchString += " ";
34:             }
35:         }
36:
37:         url += URLEncoder.encode(SearchString, "utf-8");
38:         //
```

```
39:         //System.out.println(url);
40:         System.out.println(SearchString + "    重複チェック：
                " + check(SearchString));
41:         System.out.println("検索しますAPI利用");
42:
43:         // HttpClientでリクエスト
44:         DefaultHttpClient client = new DefaultHttpClient();
45:         HttpGet httpGet = new HttpGet(url);
46:         ResponseHandler<List<SearchModel>> handler = new
                GoogleSerchResultAnalyzer();
47:         List<SearchModel> list = client.execute(httpGet, handler);
48:         String ret = "\n";
49:         // 解析結果を見てみる
50:         for (SearchModel model : list) {
51:             switch (mode) {
52:                 case 5://タイトルのみ
53:                     ret += model.getTitle();
54:                     ret += "\n";
55:                     break;
56:
57:                 case 6://URLのみ
58:                     ret += model.getHref();
59:                     ret += "\n";
60:                     break;
61:
62:                 case 3://内容のみ
63:                     ret += model.getDescription();
64:                     ret += "\n";
65:                     break;
66:
67:                 case 4://タイトルと内容
68:                     ret += model.getTitle();
69:                     ret += model.getDescription();
70:                     ret += "\n";
71:                     break;
72:                 case 1://日常会話モード（日本語のみ利用する）
73:                     ret += model.getTitle();
74:                     ret += model.getDescription();
75:                     break;
76:                 case 2://記号を含まない英数字を含む検索
77:                     ret += model.getTitle();
78:                     ret += model.getDescription();
79:                     break;
80:             }
81:         }
82:         //System.out.println("検索結果：" + ret);
83:         //無駄な文字をフィルタリングする
84:         if (mode == 1) {
85:             ret = ret.replaceAll("[a-zA-Z.‖-○&;]", "");
86:             ret = filterling(ret);
87:         } else if (mode == 2) {
88:             ret = ret = ret.replaceAll("[.‖-○&;]", "");
89:             ret = filterling(ret);
90:         }
91:
92:         // 終了処理
93:         client.getConnectionManager().shutdown();
```

```
94:         return ret;
95:     }
96:
97:     private String filterling(String ret) {
98:         ret = ret.replaceAll("日本", "");
99:         ret = ret.replaceAll("日前", "");
100:        ret = ret.replaceAll("キャッシュ", "");
101:        ret = ret.replaceAll("類似", "");
102:        ret = ret.replaceAll("http", "");
103:        ret = ret.replaceAll("//", "");
104:        ret = ret.replaceAll(":", "");
105:        ret = ret.replaceAll(".jp", "");
106:        return ret;
107:
108:    }
109:
110:    public String serch(String str) throws UnsupportedEncodingException,
        IOException {
111:        return serch(str, 2, 1);
112:    }
113:
114:    public String serch(String str, int kensu) throws
        UnsupportedEncodingException, IOException {
115:        return serch(str, kensu, 1);
116:    }
117:
118:    //こっちを使うとすでに検索済みの単語も一緒に獲得できる（便利）
119:    public LinkedHashMap<String, LinkedHashMap> searchWords_Merge(String[]
        searchs) throws IOException {
120:        searchWords(searchs);
121:        return googleResultsAllData;
122:    }
123:
124: //文字列の配列を渡すと検索結果と検索ワードがマップになって帰ってくる便利ちゃん
125:    public LinkedHashMap<String, LinkedHashMap> searchWords(String[] searchs)
        throws IOException {
126:        LinkedHashMap<String, LinkedHashMap> googleResults = new LinkedHashMap<
            String, LinkedHashMap>();
127:        int threadNumber = searchs.length;
128:        // 8スレッド用意
129:        ExecutorService executor = Executors.newFixedThreadPool(threadNumber);
130:
131:        // 結果を入れる配列
132:        LinkedHashMap<String, Integer>[] results = new LinkedHashMap[
            threadNumber];
133:
134:        // タスクのリストを作る
135:        List<Callable<LinkedHashMap<String, Integer>>> tasks = new ArrayList<
            Callable<LinkedHashMap<String, Integer>>>();
136:
137:        //System.out.println("同時に初期化するほうが圧倒的に早い");
138:        for (int i = 0; threadNumber > i; i++) {
139:            tasks.add(new ParallelInit(searchs[i], googleResultsAllData, this
                ));
140:        }
141:
142:        try {
143:            // 並列実行
144:            List<Future<LinkedHashMap<String, Integer>>> futures = null;
```

```
145:             try {
146:                 futures = executor.invokeAll(tasks);
147:             } catch (InterruptedException e) {
148:                 System.out.println(e);
149:             }
150:             //System.out.println("-----------");
151:
152:             // 結果をresultsに入れる
153:             for (int i = 0; i < threadNumber; i++) {
154:                 try {
155:                     results[i] = (futures.get(i)).get();
156:                 } catch (Exception e) {
157:                     System.out.println(e);
158:                 }
159:             }
160:         } finally {
161:             // 終了
162:             if (executor != null) {
163:                 executor.shutdown();
164:             }
165:
166:             int id = 0;
167:             // 結果の配列の中身
168:             for (LinkedHashMap<String, Integer> result : results) {
169:                 System.out.println(searchs[id] + result);
170:                 googleResults.put(searchs[id], result);
171:                 id++;
172:             }
173:         }
174: System.out.println("第3");
175:         for (String key : googleResults.keySet()) {
176:             googleResultsAllData.put(key, googleResults.get(key));
177:         }
178:         System.out.println("第4");
179:         tw.saveHashMap(googleResultsAllData);
180: System.out.println("第5");
181:         //マージはしておくけど、今回のしか返さない
182:         return googleResults;
183:     }
184:
185:     public boolean check(String str) {
186:         if (googleResultsAllData.containsKey(str)) {
187:             return true;
188:         }
189:         return false;
190:     }
191:
192:     public boolean check(String[] str) {
193:         System.out.println("現在ある単語をチェック");
194:         for (String one : str) {
195:             System.out.println(one);
196:             if (googleResultsAllData.containsKey(one)) {
197:                 System.out.println("はすでにある");
198:                 return true;
199:             }
200:         }
201:         System.out.println("はない");
202:         return false;
203:     }
```

```
204:
205: }
206:
207: /**
208:  * Google検索の中身をHttpCleanerを使用して解析し、 検索結果のリンク先をList<
          String>で返す
209:  */
210: class GoogleSerchResultAnalyzer
211:         implements ResponseHandler<List<SearchModel>> {
212:
213:     @Override
214:     public List<SearchModel> handleResponse(final HttpResponse response)
215:             throws HttpResponseException, IOException {
216:         StatusLine statusLine = response.getStatusLine();
217:
218:         // ステータスが400以上の場合は、例外をthrow
219:         if (statusLine.getStatusCode() >= 400) {
220:             throw new HttpResponseException(statusLine.getStatusCode(),
221:                     statusLine.getReasonPhrase());
222:         }
223:
224:         // contentsの取得
225:         HttpEntity entity = response.getEntity();
226:         String contents = EntityUtils.toString(entity);
227:
228:         // HtmlCleaner召還
229:         HtmlCleaner cleaner = new HtmlCleaner();
230:         TagNode node = cleaner.clean(contents);
231:
232:         // 解析結果格納用リスト
233:         List<SearchModel> list = new ArrayList<SearchModel>();
234:
235:         // まずliを対象にする
236:         TagNode[] liNodes = node.getElementsByName("li", true);
237:         for (TagNode liNode : liNodes) {
238:             // クラスがgじゃなかったら、多分リンクじゃないので除外
239:             if (!"g".equals(liNode.getAttributeByName("class"))) {
240:                 continue;
241:             }
242:
243:             SearchModel model = new SearchModel();
244:
245:             // タイトルの取得
246:             TagNode h3Node = liNode.findElementByName("h3", false);
247:             if (h3Node == null) {
248:                 continue;
249:             }
250:             model.setTitle(h3Node.getText().toString());
251:
252:             // URLの取得
253:             TagNode aNode = h3Node.findElementByName("a", false);
254:             if (aNode == null) {
255:                 continue;
256:             }
257:             model.setHref(aNode.getAttributeByName("href"));
258:
259:             // 概要の取得
260:             TagNode divNode = liNode.findElementByName("div", false);
261:             if (divNode == null) {
```

```
262:                 continue;
263:             }
264:             model.setDescription(divNode.getText().toString());
265: 
266:             list.add(model);
267:         }
268: 
269:         return list;
270:     }
271: 
272: }
273: 
274: /**
275:  * 検索結果格納クラス
276:  */
277: class SearchModel {
278: 
279:     String href;
280:     String title;
281:     String description;
282:     String div;
283:     String span;
284: 
285:     public String getHref() {
286:         return href;
287:     }
288: 
289:     public String getDiv() {
290:         return div;
291:     }
292: 
293:     public String getSpan() {
294:         return span;
295:     }
296: 
297:     public void setHref(String href) {
298:         this.href = href;
299:     }
300: 
301:     public String getTitle() {
302:         return title;
303:     }
304: 
305:     public void setTitle(String title) {
306:         this.title = title;
307:     }
308: 
309:     public void setDiv(String div) {
310:         this.div = div;
311:     }
312: 
313:     public void setSpan(String span) {
314: 
315:     }
316: 
317:     public String getDescription() {
318:         return description;
319:     }
320: 
```

```
321:     public void setDescription(String description) {
322:         this.description = description;
323:     }
324: }
325:
326: class ParallelInit implements Callable<LinkedHashMap<String, Integer>> {
327:
328:     String search;
329:     LinkedHashMap<String, LinkedHashMap> googleResultsAllData;
330:     GoogleApi google, google2;
331:     WordCounter counter;
332:     DataBase database;
333:
334:     public ParallelInit(String last, LinkedHashMap<String, LinkedHashMap>
          googleAll, GoogleApi google) throws IOException {
335:         this.counter = new WordCounter();
336:         search = last;
337:         googleResultsAllData = google.googleResultsAllData;
338:         this.google = google;
339:         this.database = google.database;
340:         database.setData("googleResultsAllData", googleResultsAllData);
341:
342:     }
343:
344:     //ここだけ並列処理できれば良い
345:     @Override
346:     public LinkedHashMap<String, Integer> call() throws Exception {
347:         //もし、過去に検索したことのある単語の場合検索を省略する
348:
349:         google.googleResultsAllData = (LinkedHashMap<String, LinkedHashMap>)
              google.database.getData("googleResultsAllData");
350:
351:         if (google.googleResultsAllData.containsKey(search)) {
352:             //System.out.println(search + "を検索しなくて省エネだよ！");
353:             return google.googleResultsAllData.get(search);
354:         }
355:                 System.out.println("第0" + search);
356:
357:         LinkedHashMap<String, Integer> wordcount = counter.wordcount(google.
              serch(search, 800), 0);
358:         System.out.println("第一");
359:         google.database.setData("googleResultsAllData", google.
              googleResultsAllData);
360:         System.out.println("第2");
361:         return wordcount;
362:     }
363:
364: }
```

リスト 4.10 の 20 行めにて検索先の URL を作成しており，google の検索結果ページの URL を作成し，情報を取得することで Google 検索の結果と同等の内容を取得することができる．

リスト 4.10 の 23 行めから 37 行めにかけて検索キーワードを作成しており，先ほど説明した kuromoji の検索しやすい単語ごとに区切る Search モードの機能だけではなく，今回は助詞と助動詞を検索ワードから排除して検索を行っている．

リスト 4.10 の 47 行めでは実際にウェブサイト上から情報を取得しており，取得した結果を GoogleSerchResultAnalyzer クラスを用いて解析を行い，リストに格納している．

4.7.3 検索結果のフィルタリング

検索結果にはどの単語を検索しても必ず含まれる単語が複数ある．
例としては「キャッシュ」といった単語や「類似」という単語である．
これらの単語は頻出単語表を作成する上で不要なため，フィルタリングを行っており，実際にフィルタリングを行っているのは**リスト 4.10** の 84 行めから 108 行めである．また，84 行めから行われているのは記号を取り除く作業であり，頻出単語とはなりえない記号を取り除いている．

4.7.4 頻出単語表の作成

頻出単語表を作成するにあたって独自に WordCounter というクラスを実装した．

以下に WordCounter から一部抜粋したソースコードを記載する．

リスト 4.11 WordCounter の一部抜粋ソースコード (word.java)

```
1:  public class WordCounter {
2:
3:      HashMap<String, Integer> wordMap;
4:      public HashMap<String, Integer> wordcount(String text) {
5:         return wordcount(text,0);
6:     }
7:     public LinkedHashMap<String, Integer> wordcount(String text,int border) {
8:      wordMap = new HashMap<String, Integer>();
9:          // この２行で解析できる
10:     Tokenizer tokenizer = Tokenizer.builder().build();
11:     List<Token> tokens = tokenizer.tokenize(text);
12:
13:         // 結果を出力してみる
14:     for (Token token : tokens) {
15:         if (token.getPartOfSpeech().contains("固有名
                詞") && !token.getPartOfSpeech().contains("助
                詞") && !token.getPartOfSpeech().contains("動
                詞") && !token.getPartOfSpeech().contains("副
                詞") && !token.getPartOfSpeech().contains("助動
                詞") && !token.getPartOfSpeech().contains("記
                号") && !token.getPartOfSpeech().contains("接頭詞")) {
16:             String word = token.getSurfaceForm();
17:                 //System.out.println(token.getSurfaceForm() + " : " + token.
                        getPartOfSpeech());
18:             if (wordMap.get(word) == null) {
19:                 wordMap.put(word, 1);
20:             } else {
21:                 wordMap.put(word, wordMap.get(word) + 1);
22:             }
23:         }
24:     }
```

```
25:     List<Map.Entry> entries = new ArrayList<Map.Entry>(wordMap.entrySet());
26:     Collections.sort(entries, new Comparator() {
27:         public int compare(Object o1, Object o2) {
28:             Map.Entry e1 = (Map.Entry) o1;
29:             Map.Entry e2 = (Map.Entry) o2;
30:             return ((Integer) e1.getValue()).compareTo((Integer) e2.getValue
                    ());
31:         }
32:     });
33:     LinkedHashMap<String,Integer> result = new LinkedHashMap<String,Integer>();
34:     for (Map.Entry entry : entries) {
35:         String key = entry.getKey().toString();
36:         int val = Integer.parseInt(entry.getValue().toString());
37:         if(val>border){
38:             result.put(key, val);
39:         }
40:     }
41:     return result;
42: }
```

リスト 4.11 の 7 行めに定義されている wordcount というメソッドに対して単語ごとにカウントを行って欲しい文字列を渡すことで，解析を行うことができる，また引数としては border を設定することができ，1 回しか出てこない単語に関してはカウントを行わないという設定を行うことができる．

初めにカウントを行うにあたって入力された文章を形態素解析にかける必要がある，実装では**リスト 4.11** の 11 行めにて，kuromoji を用いた形態素解析を行っています．

次に形態素解析を行った文章に出てくる単語をカウントする部分については**リスト 4.11** の 14 行めから 24 行めにて行っています．

最後にカウントした結果を HashMap 形式にし，41 行めでその HashMap を返しています．

第 5 章

実行結果

5.1　Unity の出力画面の図

実際に Unity でのキャラクターとの対話を行う際の画面は以下のの様な画面で対話を行うことが出来る.

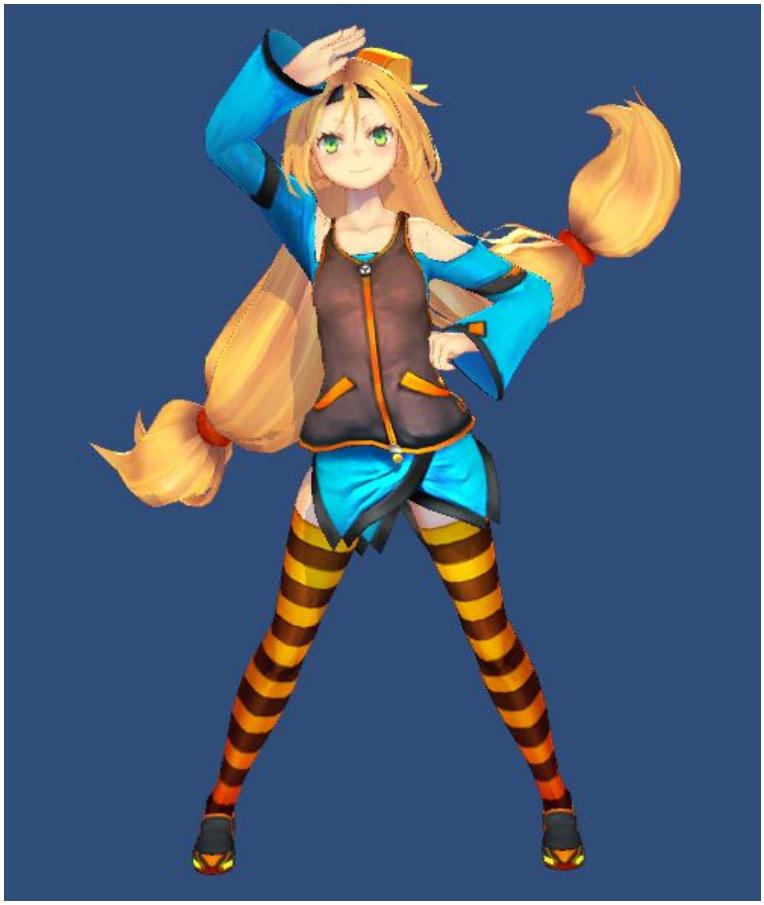

図 5.1　キャラクターとの対話画面

図 5.1 の画面にて実際にキャラクターと対話を行う形式で作成した人工知能のアリゴリズムが正しく動作しているのかを確認することができる.

5.2 実際の会話

実際にこの人工知能利用フレームワークを用いて会話を行った例を以下の**表 5.1** に示す.

表 5.1 キャラクターとの対話例

ユーザーの入力	キャラクターからの返答	動作
カルボナーラ作ってるんだ	隠し味にチョコレートいれちゃう？	悪巧みの動作
うわーマシンヘラ強すぎ，負けたよ	あーぁーゲームオーバーじゃん	がっかり

以上の**表 5.1** の様にユーザーが話しかけた内容に対して返答を行う仕組みが実装されていることがわかる.

また，ユーザーが話しかけた内容の分野を解析し，その分野に対応している解析や出力アルゴリズムが対応しているため，その分野のより詳しい会話を行うことができるもの**表 5.1** を見るとわかる.

5.3 アルゴリズムを追加した後の会話

実際にアルゴリズムを追加した後に会話を行うとどの様な変化があるかを検証してみたいと思う.

今回追加を行うアルゴリズムの分野は「パズドラ」[*1] という分野をもつ解析と出力を行うアルゴリズムであり，解析を行う部分の話題を解析する分野に対して，は簡易的なパズドラというゲーム内でよく行われる会話の話題を推定するアルゴリズムを追加した.

出力を行う，返答アルゴリズムにも「パズドラ」の話題に対応，特化する出力アルゴリズムを追加した.

このパズドラのアルゴリズムを追加した後の会話を以下の**表 5.2** に示す.

このパズドラは現在実装してある，ゲームの分野の中のゲームのタイトル名であり，ただ単にゲームに関連のある単語であれば，ゲームの分野が解析を行いますが，ここでパズドラというゲーム固有の単語である「マシンヘラ[*2]」という単語を含めることで，**表 5.2** の様にパズドラの解析と出力を行うアル

[*1] パズル&ドラゴンズ』は、ガンホーオンラインエンターテイメントから配信されている iOS，Android，Kindle Fire 用ゲームアプリ（パズル RPG）. 略称は『パズドラ』.

[*2] 2015 年 12 月 26 日現在最大のヒットポイントを所有する敵

表 5.2　アルゴリズム追加後の会話

ユーザーの入力	キャラクターからの返答	動作
うわーマシンヘラ強すぎ，負けたよ	魔法石課金してコンテニューしよ！	悪巧みの動作

ゴリズムが選択されます.
返答返答内容も「負けたよ」という発言から，このスマホゲームの課金アイテムである魔法石を用いて再度，敵に対して挑戦しようという返答を行うことができています.
この様に料理やゲームといった種類を横に広げていくアルゴリズムの追加方法と，料理やゲームのさらに細かい話題に対して対応させるアルゴリズムの追加方法などがあります.

第 6 章

結論

6.1 結論

今回作成した人工知能利用フレームワークを使い，考案したアルゴリズムを作成しキャラクターと会話することで試すことはできたと考えています.
しかし，話題が固定された状況で返答アルゴリズムを書く必要があり，現在どの話題で，どのアルゴリズムが回答するかを選択する部分のプログラムを書きたい場合はその部分のプログラムを直接書き換えなくてはならない欠点があります.

6.1.1 アルゴリズムの追加による出力の変化

5.3 の実行結果を見て分かる通り，アルゴリズムを追加することでその分野の話題になった時に追加したアルゴリズムが応答していることが分かる.
また，解析を行うアルゴリズムを追加したことによって，その分野のさらに詳しい話題の分析が 5.3 の返答を見て分かる通り，可能になりました.

6.1.2 簡単にアルゴリズムを追加できたか

考案したアルゴリズムを素早く試すために，アルゴリズムの追加する際の構造を簡単化したため，特定の話題の時に解析や出力情報のアルゴリズムの作成と追加を非常に簡単に実現できる機構は達成できたと考えています.
会話を行うことができるアルゴリズムを，この人工知能利用フレームワークでは 3 行のプログラムソースコードの記述で実現できるため，非常に開発を開始するまでの時間を短くすることができたと考えています.
また，Unity による出力先のサポートにより，この会話内容を本物のキャラクターに言われたらどの様に感じるかもシミュレーションを行うことができるため，文字だけでの出力しか行えない場合よりもよりリアルなコミュニケーションを行う人工知能や人工無脳の作成を目指すことができることがわかりました.

謝辞

何か色々と感謝する.

参考文献

[1] ゼロの使い魔制作委員会：“ゼロの使い魔公式ウェブサイト” http://www.zero-tsukaima.com/zero/index.html (2012/12/28).

[2] 奥村晴彦 著：“LaTeX 2_ε 美文書作成入門 改訂第 3 版” (技術評論社 2004, 403pp).