

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							

受領印を受けた後、本票を受け取り 大切に保管してください。	受領印

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								

(^ 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

[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]

寺
田
佳
輔

田胡
研究室

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

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】

現在機械学習などの分野では，人工知能や人工無脳，また，Deep learning を始めとした革新的な技術が注目を浴びている．

現在、その人工知能を開発するだけでなく，その開発した人工知能を用いて特定の分野で応用することにも注目が集まっており，人工知能を搭載したソフトバンクのロボット Pepper など，各企業が人工知能やを用いた製品を開発している．

そこで本研究では人工知能の活用を目的としてキャラクターと対話を行うことのできるプログラムの開発を支援する人工知能を利用するためのフレームワークを開発した．

既存の人工知能自体の開発を目的としている，人工知能フレームワークでは人工知能自体を作成する過程をサポートするものであるが，本研究では，近年注目を浴びている事項知能を有効的に利用するためのフレームワークを作成した．

このフレームワークを用いることで，人工知能の利用例の 1 つであるキャラクターとの対話プログラムを簡単に作成することが可能となり，利用者が発案した対話アルゴリズムをすぐに開発し，キャラクターとの対話という形で実現することができる．

また，キャラクターとの対話を行うことで，人工知能を開発している際に気付きにくい点である，このキャラクターに実際にこのセリフを言われたときに，どのような気持ちになるだろうというキャラクターを目の前に対話することで初めてわかる情報，フィードバックをシミュレートし，開発者に伝えることができる点も，この人工知能利用フレームワークの特徴の 1 つである．

目次

第 1 章	現状	1
1.1	近年の機械学習	1
1.1.1	人工無能	1
1.1.2	人工知能	1
1.1.3	Neural Network	2
1.1.4	DeepLearning	2
1.2	一般的な人工知能開発フレームワーク	2
第 2 章	提案	3
2.1	開発した人工知能の活用	3
2.1.1	知能の開発をサポートする既存フレームワーク	3
2.2	開発した知能を試す環境	3
2.3	人工知能利用フレームワークの提案	4
2.3.1	提案する全体構成	4
2.3.2	アルゴリズムのみを簡単に追加可能な知能ハブ	4
2.3.3	作成したアルゴリズムを Unity ですぐに試せる機構	5
2.3.4	Unity が利用可能なモーションを追加する機構	5
第 3 章	設計	7
3.1	入力された情報を解析する機構	7
3.1.1	解析する情報別にアルゴリズムを保持する機能	8
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	8
3.1.3	解析アルゴリズムを簡単に追加する機能	11
3.2	解析結果を保存する機構	12
3.2.1	解析情報を保存する機能	12
3.2.2	解析情報を取得する機能	12
3.3	解析情報を元に出力内容を作成する機構	13
3.3.1	返答を行うタイミング	13
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	14
3.3.3	会話の話題別に返答アルゴリズムを選ぶ機能	14
3.4	作成した知能を Unity で試す機構	15
3.4.1	UnityWebPlayer での出力について	15
3.4.2	Unity との連携に利用する WebSocket	15

3.4.3	Unity への送信フォーマットと作成	16
3.4.4	Unity からの受信フォーマット	16
3.5	アルゴリズムを選定する際に用いる GoogleAPI	17
3.5.1	GoogleAPI について	17
3.5.2	GoogleAPI の有効性	17
第 4 章	実装	18
4.1	開発環境	18
4.1.1	Java の利用	18
4.1.2	Maven フレームワーク	18
4.2	解析部分の実装	18
4.2.1	解析コントローラー	18
4.2.2	解析知能ハブ	19
4.2.3	解析アルゴリズムの解析知能ハブへの追加	20
4.2.4	現在実装している解析アルゴリズム	21
4.3	データベースの実装	22
4.3.1	全ての解析情報を保存する機構	22
4.3.2	解析した情報を取得する機構	22
4.4	出力を行う部分の実装	23
4.4.1	出力コントローラー	23
4.4.2	出力知能ハブ	23
4.4.3	出力アルゴリズムの出力知能ハブへの追加	25
4.4.4	現在実装している出力アルゴリズム	26
4.5	Unity との通信の実装	27
4.5.1	通信方式	27
4.5.2	Unity からの入力情報の受信	28
4.5.3	Unity への命令の送信	28
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	29
4.6.1	動作選択アルゴリズムの実装	29
4.7	GoogleAPI による頻出単語表の作成	30
4.7.1	形態素解析による検索ワードの作成	30
4.7.2	GoogleAPI を利用して検索結果を取得	31
4.7.3	検索結果のフィルタリング	38
4.7.4	頻出単語表の作成	38
第 5 章	実行結果	41
5.1	Unity の出力画面の図	41
5.2	実際の会話	42
5.3	アルゴリズムを追加した後の会話	42
第 6 章	結論	44
6.1	結論	44

6.1.1	アルゴリズムの追加による出力の変化	44
6.1.2	簡単にアルゴリズムを追加できたか	44
	謝辞	45
	参考文献	46

図目次

2.1	全体の構成図	4
3.1	解析が行われるまでの図	7
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	8
3.3	2015 年 12 月 20 日現在の「クッパ 落ちた」の Google 検索結果	9
3.4	2015 年 12 月 20 日現在の「クッパ 美味しい」の Google 検索結果	10
3.5	抽象クラスの関係と抽象クラスの実装例	11
3.6	出力情報を作成するまでの流れ	13
3.7	Unity Web Player によるキャラクターの表示画面	15
5.1	キャラクターとの対話画面	41

表目次

4.1	実装した主要メソッド	18
4.2	Abstract Mode に実装した主要メソッド	19
4.3	Abstract Mode の実装	20
4.4	実装した主要なメソッド	23
4.5	実装した主要なメソッド	24
4.6	実装した主要なメソッド	26
5.1	キャラクターとの対話例	42
5.2	アルゴリズム追加後の会話	43

第 1 章

現状

近年様々な機械学習や人工知能，人工無脳といった分野が注目を浴びている，ここではそれらの現状について説明する．

1.1 近年の機械学習

まず初めに，機械学習とはデータから反復的に学習し，そこに潜むパターンを見つけ出すことであり，近年その機械学習を会話に用いる動きがある．

1.1.1 人工無能

人工無脳 [1] とは，人工知能に対応する用語で，英語圏では chatterbot，もしくは chatbot と呼ばれ，その訳語として会話ボットあるいはおしゃべりボットとも呼ばれることがある．

この人工無脳はテキストや音声などを用いて，会話をシミュレートすることができるプログラムであり，一見知的に人間の様な応答をしているように見えるが，多くの場合は会話の中の特定のキーワードを拾いそのキーワードに対応する返答を行っている場合が多いのが現状である．

1.1.2 人工知能

人工知能 [2] とは人工的にコンピュータ上などで人間と同様の知能を実現させようという試み，或いはそのための一連の基礎技術を指すものであり，1956 年にダートマス会議でジョン・マッカーシーにより命名された．

しかし，人工知能の定義は未だ不確定な部分が多く，完全に正確な定義は存在していないのが現状である．

この人工知能は人工無脳とは異なり，ただ単にキーワードを広くだけでなく，取得した情報を用いて解析や情報の取得を行い，状況に応じた返答などが可能なプログラムのことを指すことが多い．

1.1.3 Neural Network

Neural Network とは、脳細胞を構成する「Neuron (ニューロン)」の活動を単純化したモデルであり、これを利用することによって人間の思考をシミュレーションすることができるものであり、人工知能分野で活用されている。

1.1.4 DeepLearning

DeepLearning[3] とは多層構造のニューラルネットワークの機械学習の事であり、ニューラルネットワークを多層積み重ねたモデルを機械学習させればディープラーニングということになる、また、一般的には 3 層以上のニューラルネットワークがあるものとされている。

第一層から入った情報は、より深い 3 層へと行く過程で学習が行われ、その学習を行っていくことで概念を認識する特徴量と呼ばれる重要な変数を自動で発見することができる。

この DeepLearning により、東京大学の松尾豊准教授 [4] などの AI 研究に関する大きなブレイクスルーであり、学習方法に関する技術的な革新であると指摘している。

1.2 一般的な人工知能開発フレームワーク

フレームワークは開発を行う際に開発自体をサポートするものであり、技術自体ではないが、今回開発する人工知能を利用するフレームワーク上に実装できる技術として説明する。

現在一般的な人工知能を開発するフレームワークとして chainer や Google の TensorFlow などがある。

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、人工知能自体の開発を行う際に高速な計算が可能なことや、様々なタイプのニューラルネットを実装可能、またネットワーク構造を直感的に記述できる利点がある。

次に、Google の Brain Team の研究者たちが作った機械学習ライブラリである Tensor Flow は、グラフを構築・実行するための Python API と C++ インターフェイス一式が用意されて実際の開発を行う際に非常に有効だと考えられる。

これらの人工知能開発フレームワークは実際に開発を行うときに非常に有効であり、google の検索アルゴリズムやデータ分析などの様々な分野での応用を試みる動きがある。

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やその返答アルゴリズム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おり、その作成した人工知能を用いて会話を行う。

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。

既存のフレームワークは人工知能を作ることに着目して、作る工程をサポートするものが多い、今回提案する人工知能利用フレームワークでは考案し、作成したアルゴリズムや人工知能を複数登録することで Unity 上で動作するキャラクターと会話を行うことができるシステムである。

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、出力画面を準備をするのは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手間と時間がかかるという問題点がある。

その部分をあらかじめ人工知能利用フレームワークで提供することで、準備の手間が不要になるという利点がある。

また、キャラクターとの会話などで出力することで対話することにより、実際にそのキャラクターに言われたらどの様に感じるかをシミュレーションすることができるため、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を目指して、アルゴリズムを考え、開発することが可能になるという利点がある。

今回はその様な人工知能を利用することに着目した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、それらの作成したアルゴリズムをフレームワーク上に適当に記述することで、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 提案する全体構成

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。

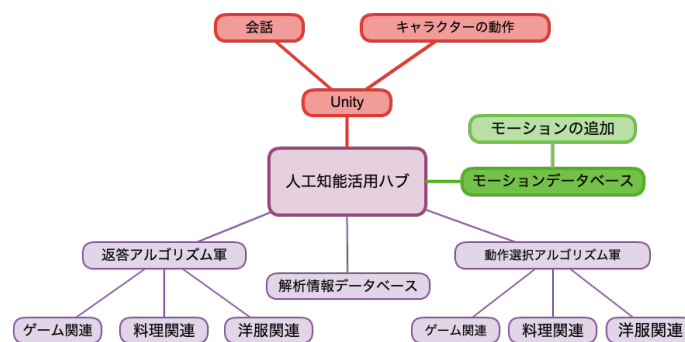


図 2.1 全体の構成図

提案する人工知能利用フレームワークには Unity で作成したキャラクターを出力する部分とキャラクターに行わせる動作を考える人工知能ハブ、また、Unity 上で利用するためのモーションを保存しているモーションデータベースの3つから構成されます。

今回私が担当し、作成したのは人工知能ハブであり、人工知能ハブについて解説を行いたいと思います。

私が担当した、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可能となっている。

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する。

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というものである。

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実装することが可能であり、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というものである。

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設計し、複数の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ような構成を提案する。

また、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、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にする。

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。

共同研究者の藤井克成の論文 [5] 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。

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）はその場で動作を生成することが難しいのが現状である。

そのため、あらかじめモーションデータを作る必要となるのですが、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と考えられる，そこで共同開発の鈴木智博が 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 [6] ．

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になる．

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 Google API の流れで、人工知能利用フレームワークの構成を解説する。

また、全体の構成は 2.3.1 に示した図と同じ構成となっている。

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。

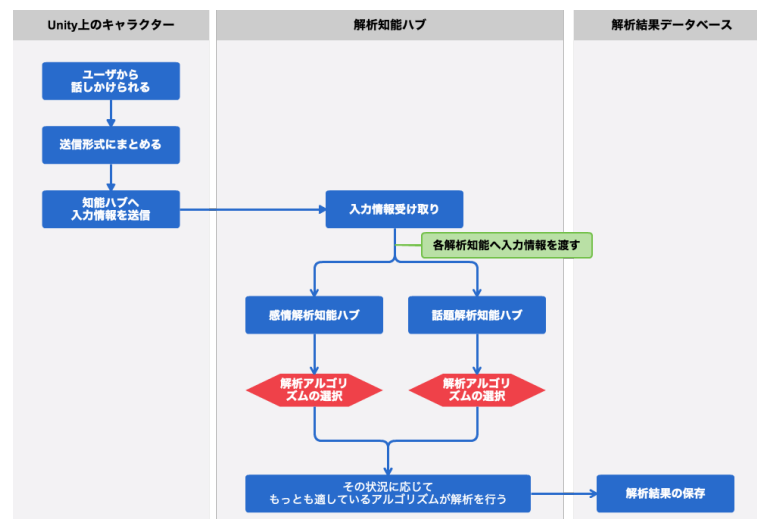


図 3.1 解析が行われるまでの図

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、人工知能ハブへと送信する。

送信した情報は人工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る。

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る。

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，実際の解析を行う。

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る。

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する。

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す。

そして，各解析知能は入力された情報からその情報を解析するためのアルゴリズムを複数所持しており，それを具体例を用いて表した図を以下の図 3.2 に示す。

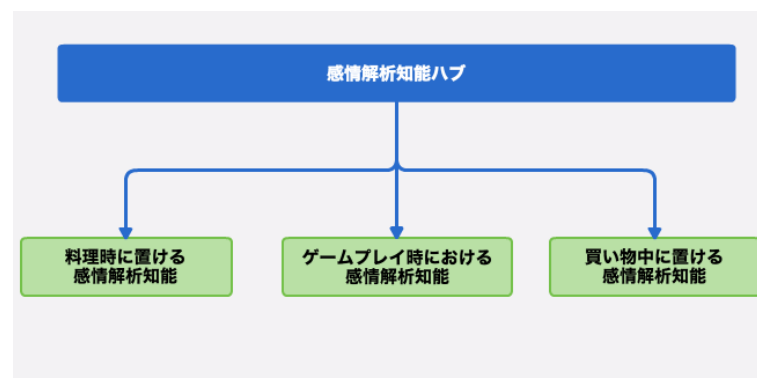


図 3.2 感情解析知能ハブとそれに付随する解析アルゴリズム

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所持している。

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所持し，適切なアルゴリズムが選択されるように設計されている。

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，解析知能ハブにある抽象クラスを保存している配列に追加することでそのアルゴリズムを増やすことができる。

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る。

よって，料理に関する話題をしている際は，料理関連の単語や会話に対応した解析アルゴリズムがが選択され解析を行い，料理関連のアルゴリズムが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る。

話題を推定する際には GoogleAPI^{*1}を用いており、入力された内容を検索にかけて、頻出単語表を作成し、既存の各アルゴリズムと最も頻出単語表が似ている話題をもつ解析アルゴリズムが解析を行います。

この様な構造を用いることで、以下の図 3.3「クッパ^{*2}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 の検索結果の場合、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る。

この時に「クッパ^{*3}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 美味しい」の検索結果を見て分かる通り、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る。

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る。

この機能がない場合、「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

*1 独自実装を行った Google を用いて検索を行う機能

*2 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

*3 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



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

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されてしまうことが予測できる．

またこの両方の文章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ある．

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する。

この、実際に解析を行うアルゴリズムの追加は、予め定義されている子の抽象クラスを実装することで追加することが可能であり、その実装の手順もソースコードの行数に換算すると、そのアルゴリズムの話題の設定、そのアルゴリズム、親クラスへの登録という最短 3 行でアルゴリズムを追加することが可能になっている。

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る。

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、その関係性を図 3.5 に示す。

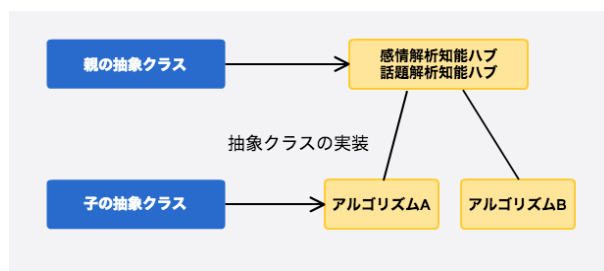


図 3.5 抽象クラスの関係と抽象クラスの実装例

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新たなアルゴリズムを考案した際には子の抽象クラスを実装することによって簡単にそのアルゴリズムの実装を行い、感情を解析する知能ハブや話題を解析する知能ハブに対して特定の話題を持ったアルゴリズムを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。

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いように `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ため、データの衝突が起きないように設計されている。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構造になっている。

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する。

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。

取得する際は `object` 型で取得されるので、取得する際に利用するキー（文字）の最後を見て変数の型名にキャストすることで利用出来る。

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 のキャ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 を用いて説明する。

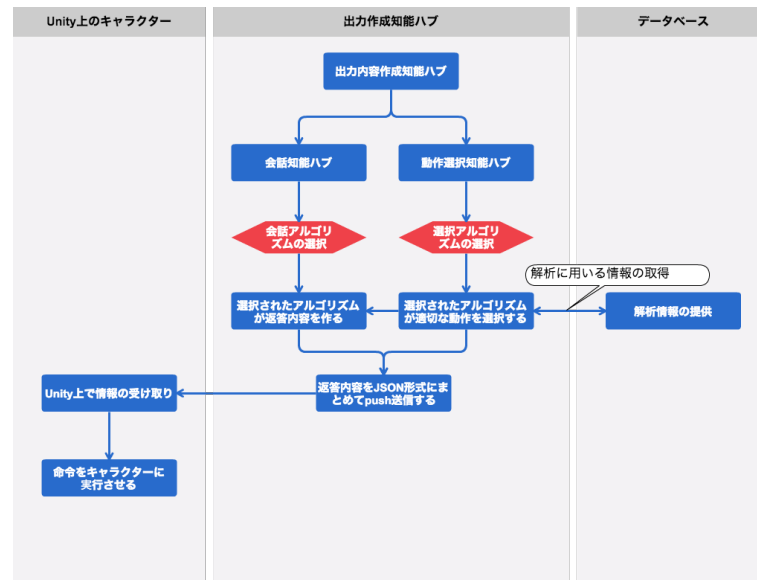


図 3.6 出力情報を作成するまでの流れ

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。

それぞれ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する。

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っている。

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 種類ある。

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ある。

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、返答を行っている。

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つまり、特定の感情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となっている。

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 へ任意のタイミングでの命令の送信が可能のため、自発的に発言しているように見せることが可能である。

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る。

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る。

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、複数の返答アルゴリズムを所持し、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選択され動作します。

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る。

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、複数のプラットフォームに対応するゲームエンジンである Unity を採用した。
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可能である。
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可能になる。

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可能であり、作成したアルゴリズムをすぐに試すことが可能である。

3.4.2 Unity との連携に利用する WebSocket

Unity との通信には Web Socket を用いる。
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 通知が可能な点である。
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る。

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う。
この Unity への送信は汎用性の高い JSON 形式^{*4}を用いて送信を行う。

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、送信を行っている。

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。
JSON 形式のデータを解釈する機能も付いているので、Unity 側から受け取ることができる情報を随時追加することができる設計となっている。

^{*4} 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 について解説する。

3.5.1 GoogleAPI について

この論文内で示す GoogleAPI とは独自に開発した Google 検索を用い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る。

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る。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る。

また、google 検索を用いているので、例えば「サバの味噌煮」と検索をして、検索結果の文字列から頻出単語表を作成した際に、一見料理に全く関係のない単語である文字列である「パッド」という単語が入っていることがある、しかし料理関連の解析知能と単語表を比較した時に料理の検索結果とサバの味噌煮の検索結果で同じサイトである、「クックパッド^{*5}」が表示されることが多く非常に有効性が高いと推測できるため用いることにした。

^{*5} 献立に困った時、すぐに希望の食材で検索ができる、また自分の料理ホームページが簡単にもてる、アクセス数・登録レシピ数ともに日本一のレシピサイトです。

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う。

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る。

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る。

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る。

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ある。

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^{*1}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る。

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する。

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)を実装した。

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ある。

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ある、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あり、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る。
その部分のソースコードを以下のリスト 4.2 に示す。

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: }
```

^{*1} 検索結果の文字列から固有名詞だけをのこし、出現した単語とその数をカウントしたもの

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、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り、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する。

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 + データ型の名前をつけ、保存する。

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する。

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る。

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行なっている。

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 を加えて数字でその感情に関する文字列が出てきた回数を表現する仕組みになっている。

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おり、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る。

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る。

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:     access++;  
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する必要がなく、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可能となっている。

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 メソッドを呼び出す。
以下のリスト 4.5 に getData メソッドのソースコードを示す。

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:     }
```

```

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 型の鍵が必要である。
 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 HashMap 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す。

4.4 出力を行う部分の実装

Unity へ返答するための情報を作成する部分の実装について説明する。

4.4.1 出力コントローラー

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する。
 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、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る。

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、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す様になっている。

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う。

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 Abstract 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り、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る。

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、独自実装をおこなった 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る。

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おり、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る。

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る。

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する．

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る．

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る．

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，取得する．

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る．

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ある，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票は先ほど説明したアル

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

ゴリズムの選定に利用される。

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する。

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 種類である。

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，返答を行う。

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る。

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する。

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る。

その具体的なアルゴリズムや通信に関しては 4.6 に記述する。

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る．

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 を実装した．

以下のリスト 4.8 にソースコードを示す．

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 UnsupportedOperationException,
    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る。

これで、セッションが確立され、受信を行う準備が完了した。

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 が呼ばれる。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 上のキャラクター以外の端末からメッセージを受け取った際に呼ばれる。

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る。

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 行目で全ての接続クライアントに対して命令を送信している。

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る。

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る。

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おり、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するように実装をおこなった。

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する。

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る．

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る．

また，このコントローラーの実装に関しては共同開発の鈴木さんの論文 [6] を参照すると，このコントローラーからデータベースへの接続を行なっていることがわかる．

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 行目をみると，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る．

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る．

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検索結果やアルゴリズムを選択する際の制度に関わるため，非常に重要である．

そのため Java の形態素解析器である `kuromoji`[11] 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 `mwSoft blog`[10] を参照，参考にしつつ実装を行った．

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「今日の夕飯何にしよう」を「今日 夕飯 何 しよ」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る, 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。

また, この機構を実現するために mwSoft blog[9] を参考にした。

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
        UnsupportedOperationException, 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: // 解析結果を試みる
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.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>
335:         googleAll, GoogleApi google) throws IOException {
336:         this.counter = new WordCounter();
337:         search = last;
338:         googleResultsAllData = google.googleResultsAllData;
339:         this.google = google;
340:         this.database = google.database;
341:         database.setData("googleResultsAllData", googleResultsAllData);
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>)
350:             google.database.getData("googleResultsAllData");
351:
352:         if (google.googleResultsAllData.containsKey(search)) {
353:             //System.out.println(search + "を検索しなくて省エネだよ!");
354:             return google.googleResultsAllData.get(search);
355:         }
356:
357:         System.out.println("第0" + search);
358:
359:         LinkedHashMap<String, Integer> wordcount = counter.wordcount(google.
360:             serch(search, 800), 0);
361:         System.out.println("第1");
362:         google.database.setData("googleResultsAllData", google.
363:             googleResultsAllData);
364:         System.out.println("第2");
365:         return wordcount;
366:     }
367: }

```

リスト 4.10 の 20 行めに検索先の 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 検索の結果の内容を取得している。

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なっている。

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、kuromoji を用いた形態素解析を行なっている。

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 行めに行っている。

最後にカウントした結果を HashMap 形式にし、41 行めでその HashMap を返していることがわかる。以下のリスト 4.12 に例として「今日の夕飯何にしよう」で検索を行った結果をもとに作成した頻出単語表を記述する。

リスト 4.12 頻出単語表 (words.json)

```

1: 今日の夕飯何にしよう
2:
3: {
4:     マイ=1, ヨシエ=1, ブログ=1, ミクル=1, 和=1, 成=1, 椒=1,
5:     神戸=1, ケイ=1, 原村=1, 弘=1, 門倉=1, マス=1, とわ=1,
6:     ガルズ=1, 次新=1, 沖縄=1, 真理子=1, 弥=1, マネ=1, 諏訪=1,
7:     ちの=1, ゴマ=1, キッコマン=1, 読売新聞=1, ポートランド=1,
8:     小池=1, 仁=1, カレー=1, TOP=1, ヨシ=1, 倉田=1, 柏=1,
9:     なみ=1, キモ=1, 翔=1, さんま=1, ハリー=1, 月島=1, 祥=1,
10:    千葉=1, オークワ=1, 華=1, 英=1, 平=1, 米=1, 集英社=1,
11:    茅野=1, 佐賀=1, 原田=1, 金閣寺=1, 久喜=1, 松尾=2, 千趣会=2,
12:    ドイツ=2, 亜=2, ココスジャパン=2, ココロ=2, ナナ=2, パ=2,
13:    太郎=2, ー=2, 中=3, はるみ=3, 栗原=3, クック=4, 笑=4, 日=5, 毎日=8
14: }

```

リスト 4.12 の頻出単語表を見ると国名が多く、その国々の料理の検索結果を取得していることがわ

かる。

例えば日という単語であれば日 = 5 の様に表記されており，日という単語が 5 回出てきたことがわかる。

また，そのほかにも料理のレシピを検索することができるウェブサービスであるクックパッドのクックの文字を多く取得しているほか，両魅了の名前であるキッコーマンなどの単語も取得することが出来ている。

以上の様に検索結果からその検索した単語と同じ様な意味を持つ単語を取得することができる。

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。

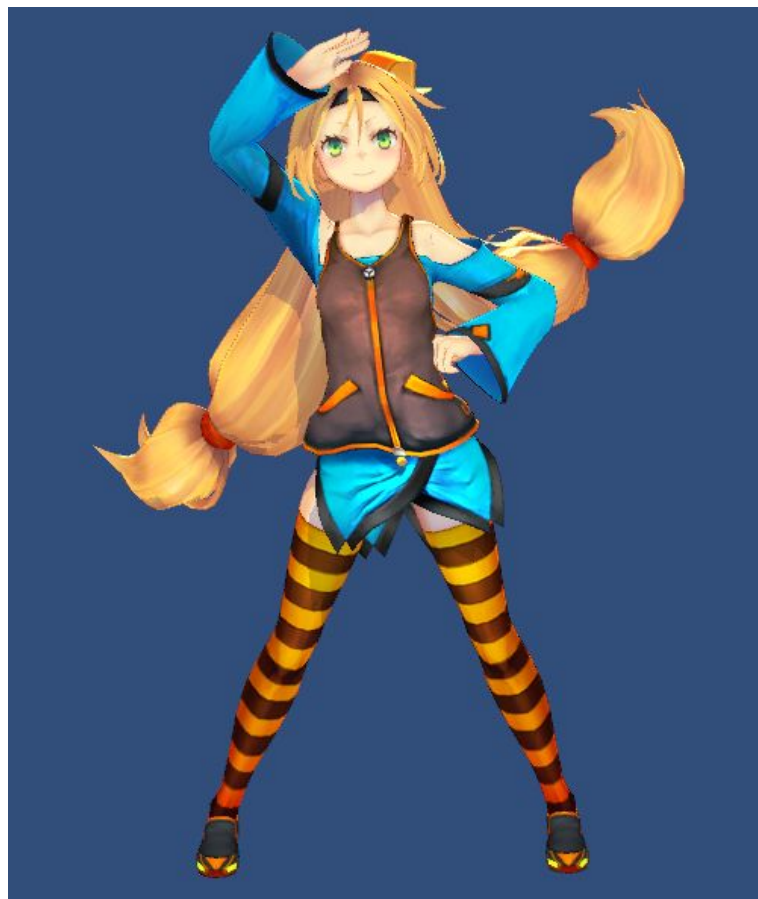


図 5.1 キャラクターとの対話画面

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ル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

ユーザーの発言	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 うわお！クッパ強い，負けたよ	隠し味にチョコレートいれちゃう？ ゲームオーバーだねー	悪巧みの動作 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よう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ゲームの「スーパーマリオブラザーズ^{*1}」(以下，マリオ)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「マリオ」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「マリオ」の話題に対応，特化する出力アルゴリズムを追加した。

このマリオ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「マリオ」は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う，ここで「マリオ」というゲーム固有の単語である「クッパ^{*2}」という単語を含めることで，表 5.2 の様にマリオの解析と出力を行うアルゴリズムが選択される。返答返答内容も「負けたよ」という発言から，同ゲームのプレイヤー強化アイテムである「きのこ」を

^{*1} 『スーパーマリオブラザーズ』(Super Mario Bros.) は，任天堂が発売したファミリーコンピュータ用ゲームソフト。

^{*2} クッパとは、マリオシリーズに登場するキャラクターであり，敵キャラクター正式名称は「大魔王クッパ」

表 5.2 アルゴリズム追加後の会話

ユーザーの発言	キャラクターからの返答	動作
うわお！クッパ強い，負けたよ	きのこを取っておこう！	がっかり

取って再度，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ることがわかる．

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る．

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る。

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る。

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、可能となった。

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。

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出来た。

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った。

謝辞

何か色々と感謝する。

参考文献

- [1] wikipedia : “人工無脳 Wikipedia” <https://ja.wikipedia.org/wiki/人工無脳> (2015 年 12 月 27 日) .
- [2] wikipedia : “人工知能 Wikipedia” <https://ja.wikipedia.org/wiki/人工知能> (2015 年 12 月 27 日) .
- [3] wikipedia : “DeepLearning” <https://ja.wikipedia.org/wiki/ディープラーニング> (2015 年 12 月 27 日) .
- [4] 東京工科大学 : “東京工科大学公式ホームページ教員紹介” <http://www.teu.ac.jp/info/lab/teacher/cs/index.html?id=1635> (2015 年 12 月 27 日) .
- [5] 藤井克成 : “Unity によるキャラクターの制御” (卒業論文 2016 年度) .
- [6] 鈴木智博 : “モーションデータベースの開発” (卒業論文 2016 年度) .
- [7] 奥村晴彦 著 : “ \LaTeX 美文書作成入門 改訂第 3 版” (技術評論社 2004, 403pp) .
- [8] マルチェッロ・マッスィミーニジュリオ・トノーニ 著 : “ \LaTeX 意識はいつ生まれるのか” (亜紀書房) .
- [9] mwSoft blog : “HttpClient と HttpCleaner で Google 検索結果を解析する例” <http://blog.mwsoft.jp/article/34841195.html> (2015 年 8 月 12 日) .
- [10] mwSoft blog : “Java 製形態素解析器「Kuromoji」を試してみる” <http://www.mwsoft.jp/programming/lucene/kuromoji.html> (2015 年 8 月 12 日) .
- [11] github : “形態素解析器 kuromoji” <https://github.com/atilika/kuromoji> (2015 年 8 月 12 日) .