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	人工知能利用フレームワークの開発								

┌(┌^ o ^)┐ キレイに切り取ってね ┌(^ o ^┐)┐

2015
年 度

人工知能利用フレームワークの開発

寺田 佳輔

田胡
研究室

[卒 業 論 文]

人工知能利用フレームワークの開発

(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

人工知能利用フレームワークの開発

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

人工知能利用フレームワークの開発

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】　現在機械学習の分野では，人工知能や人工無脳に加え Deepleaning を始めとした革新的な技術が注目を浴びている．

現在人工知能を，開発するだけでは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いる．その例として人工知能を搭載したソフトバンクのロボット Pepper など，各企業が人工知能など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た「キャラクターと対話を行うことのできるプログラム」の開発を支援するフレームワークを提案する．

既存の人工知能自体の開発を目的としている人工知能の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している．

しかし本研究では，近年注目を浴びている人工知能を有効的に利用することをサポートする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「キャラクターとの対話を行うプログラム」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素早く開発し，キャラクターとの対話という形で試すことが可能となる．

キャラクターとの対話を行う事の利点として，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様な気持ちになるだろう」というキャラクターと実際に対話をすることで初めてわかる情報やフィードバックを得られる点がある．

目次

図目次

表目次

第 1 章

現状

近年人工知能や人工無脳などの機械学習の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 [2] 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．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可能なプログラムであり，一見知的に人間の様な応答をしている様に見えるが，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．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うだけではなく，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．

1.2 一般的な人工知能開発フレームワーク

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であり，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．

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，Python API と C++ インターフェイス一式が用意されているため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．

1.3 知能の開発をサポートする既存フレームワークの現状

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．現状開発を行った人工知能を用いてキャラクターと会話を行うというところまでをサポートするものはない．キャラクターとの対話で人工知能を試したい場合は別途キャラクターが動作するプログラムを 1 から作る必要があり，アルゴリズムを作ったとしてもそれを気軽に楽しむ為の環境がないのが現状である．

1.4 人工知能のアプリケーションへの応用

現在 Microsoft のりんな [3] やソフトバンクの pepper[4] などの登場により，人工知能を用いた対話に注目が集まっている．

それによりコンピュータ上でキャラクターとの対話をシミュレートするシステムの開発を，企業だけではなく個人でも開発する動きがある．

これらの対話システムを個人で作成するには対話を行うアルゴリズムを作成するのに加えて，キャラクターとの対話を行うインターフェイスの準備やキャラクターを動作させるための動作ファイルも準備する必要があるのが現状である．

第 2 章

提案

2.1 提案するフレームワークの目標

今回提案するフレームワークの目標はアルゴリズムのみを記述することで，キャラクターとの対話を行う事ができるシステムを開発する事である．

先ほど説明した通り，現在対話システムを開発するには入力を受け付ける部分，返答アルゴリズムの部分及びキャラクターと対話を行うインターフェイスなど様々なものを開発する必要がある．

これらの対話システムを開発する際の手間を軽減し，人工知能の開発により注力する事を可能とする．

2.2 人工知能利用フレームワークの実現方法の提案

今回提案するのは返答アルゴリズムのみを開発するだけで対話システムが完成する，人工知能を利用する事に焦点を当てたフレームワークである．そのフレームワークを実現するために，人工知能ハブという様々な人工知能を所持するプログラムとキャラクターとの対話を行うことができる環境を準備した．

人工知能ハブは様々な人工知能を所持する人工知能のハブであり，websocket で発言内容を送ると返答を得られる人工知能サーバーである．また人工知能ハブは簡単に対話アルゴリズムを追加する事ができるものであり，対話アルゴリズムを追加する事で対話を行う際の精度を向上させる事が可能である．websocet 通信を用いる事ができれば人工知能ハブを利用できる，そのためウェブブラウザからスマートフォンアプリケーションまで様々なクライアントから利用する事ができる．

今回提案する人工知能利用フレームワークのキャラクターとの対話を行う事ができる環境として，Unity で開発したキャラクターと対話を行う事ができるクライアントを準備した．

キャラクターと会話を実際に行う事で，実際に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，フィードバックすることが可能であり，よりリアルなコミュニケーションを行うアルゴリズムの開発をサポートすることが可能になる．

以上の提案を実装する人工知能利用フレームワークをまとめると，人工知能ハブは対話やキャラの動作アルゴリズムを気軽に追加することが可能であり，そのアルゴリズムを元に動くキャラクターと対話を行う事ができるというシステムである．

2.2.1 全体構成の提案

この人工知能利用フレームワークの全体構成を次の**図 2.1** に示す.

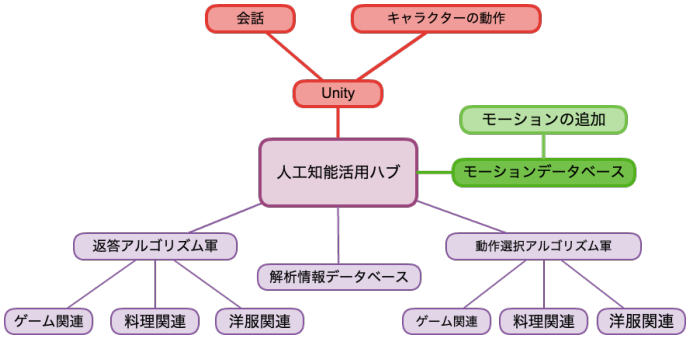

図 2.1　全体の構成図

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，キャラクターに行わせる動作を考える人工知能ハブ，及び Unity 上で利用するためのモーションを保存するモーションデータベースの 3 つから構成される.

今回私が担当する人工知能ハブの構成について提案を行う．人工知能ハブを大きく 3 つの要素に分けて構成する．具体的には 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り解析を行う部分，解析した情報を保存するためのデータベース，及び返答する内容が作り出される部分の 3 つで構成する.

動作を選択する部分に関しては，共同研究の鈴木智博が作成した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とを提案する.

2.2.2 アルゴリズムのみを簡単に追加可能な構成の提案

ここではアルゴリズムのみを簡単に追加することで対話システムが完成する構成を，人工知能利用フレームワークに実装することを提案する．このような構造があることでより簡単に人工知能を作成し，キャラクターとの対話で試すことが可能になると考えている.

人工知能ハブでは作成した会話の返答アルゴリズム，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することが出来るようにする．さらに追加したアルゴリズムに対して，それぞれ話題[*1]を設定することで，ユーザーが話しかけた内容に応じて適切なアルゴリズムを用いて返答を行うことができるようにする.

例えばゲーム関連の返答アルゴリズムを作る場合は，そのアルゴリズムをあらかじめ準備されている抽象クラスを用いて実装し，**図 2.1** の返答アルゴリズム軍にその実装したクラスを登録することで，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可能である.

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

[*1] 話題：そのアルゴリズムはどのような話題の時に解析を行うかを決める単語

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可能になるというものである．

これらの実際に解析を行うアルゴリズムや，人工知能が 1 つだけしか実装されていない場合は，その実装されているアルゴリズムを自動で選択するように設計されている．

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様々な話題に対応出来るようになり，より正確な解析アルゴリズムが選択されるようにすることが可能である．

解析だけではなく返答内容の動作や発言内容を作成する際も，ゲーム専用の返答アルゴリズムや料理に特化した返答アルゴリズムがあることでより円滑なコミュニケーションを行うことが可能になると提案する．

今後様々なアルゴリズムが必要になり，複数人で開発を行うことが想定される．その際に解析情報をデータベースを用いて共有することで効率的に開発を行えるようにするために，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．

この構成にすることで入力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などの情報を使い，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となる．

2.2.3 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ハブに登録されたアルゴリズムを用いて，キャラクターとの対話ですぐに試すことができる環境を提供する事を提案する．

この環境の開発を行うのは共同研究者の藤井克成であり，MMD モデルを利用しているためモデルを入れ替えることで好きなキャラクターで動作させることを可能にする．また，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補完しながらキャラクターを動かす技術により，よりリアルなコミュニケーションを行う事を可能とする．

この環境がある事によって，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，入出力の設計や開発をどのようにするかなどの校庭に時間をかける事なく，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る．

2.2.4 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．返答パターンは文字列で生成され，Unity へ送信されて実行されるので様々なバリエーションで返答することができる構成となる．しかしキャラクターの動作に関しては，動的にプログラムを用いて動作を生成することが難しいため，予めモーションデータを作成しておく必要性がある．

そのモーションデータを定義するファイルを生成することも手間と時間がかかるため，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，人工知能ハブと通信を行う事が可能なプログラムを開発する [7]．

この機構があることによって，人工知能活用フレームワークの中で新しい動きのパターンを追加したい場合，すぐに Kinect を用いて自ら追加したい動作をキャプチャすることで動作ファイルを生成し，データベースに動作ファイルを登録することで人工知能ハブから使えるようにする事を提案する．

第 3 章

設計

知能ハブ[*1]の構成を解説する．構成としては解析，保存，出力及び Unity との通信，GoogleAPI がある．全体の構成は 2.2.1 に示した図と同じ構成である．

3.1 入力された情報を解析する機構

Unity で作成されたキャラクターに対してユーザが発言を行い，解析知能ハブが入力情報を解析する際の機構について解説する．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．

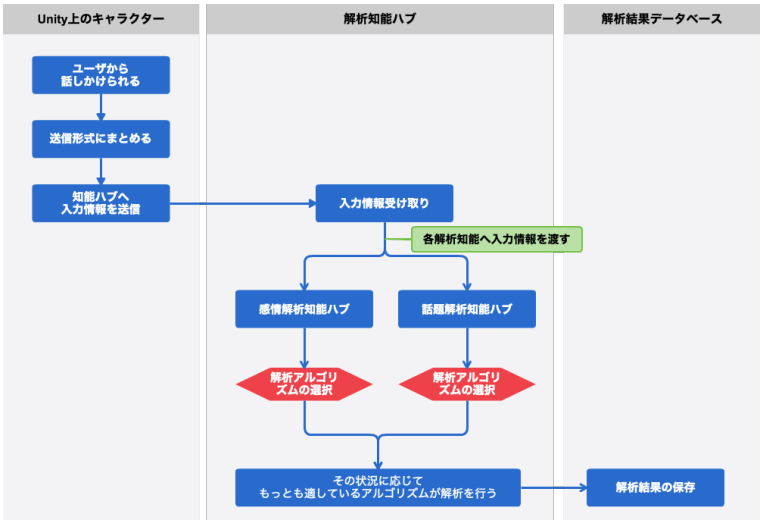

図 3.1　解析が行われるまでの図

図 3.1 の大まかな流れを解説する．ユーザーがキャラクターに話しかけると，その内容を Unity 側で受け取り文字列に変換して，解析知能ハブへと送信する．

送信した情報は解析知能ハブで受け取られ，知能ハブは感情解析知能ハブ[*2]や話題解析知能ハブ[*3]な

[*1] Unity の入力を受け取り，動作や発言を決める知能を複数集めたもの
[*2] 独自実装を行った感情を解析するためのアルゴリズムを複数持つアルゴリズムのハブ
[*3] 独自実装を行った話題を解析するためのアルゴリズムを複数持つアルゴリズムのハブ

どの各解析知能ハブへと渡される．

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，受け取った情報はこの 2 つの解析ハブへと渡される．情報が渡されると各解析ハブはその情報をもとに，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．

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自動で保存される設計となっており，その情報は様々な解析クラスや出力を作成するクラスから利用することが可能である．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，入力された情報はそれぞれ入力データを各解析知能へと渡される．各解析知能は，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，その構図を具体例を用いて表した図を以下の**図 3.2** に示す．

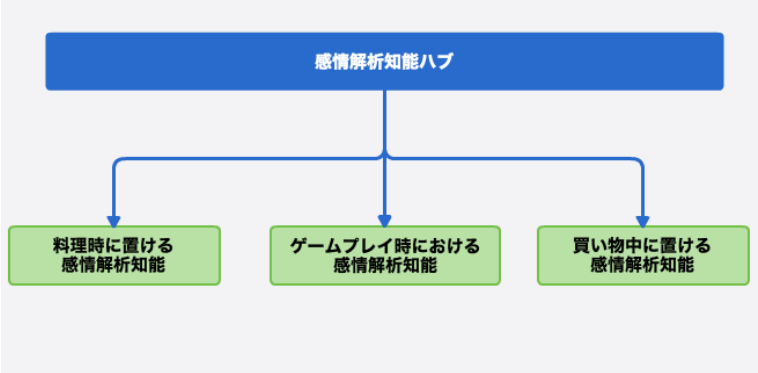

図 3.2　感情解析知能ハブとそれに付随する解析アルゴリズム

図 3.2 の感情解析知能ハブは，複数の話題毎に特化した感情を解析するため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その解析するためのアルゴリズムを保存するために，アルゴリズム型の配列を持っており，**図 3.2** の「料理時における感情解析知能」や「ゲームプレイ時における感情解析知能」などはその配列に含まれている．

解析知能ハブは配列内に，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．

図 3.2 の「感情解析知能ハブ」にあたる解析する情報である感情や，話題などの種類を追加することが可能である．このクラスが継承している「親抽象クラス[*4]」を実装し，解析知能ハブにある親抽象クラスを保存する配列に作成したクラスを追加することで，新しく解析する情報の種類を増やすことができる．

[*4] 独自実装を行った解析を行う感情や話題などの種類を増やす時に用いる抽象クラス

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．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[*5]を用いており，その詳しい実装は後で記述している．アルゴリズムの選択方法は，入力された内容を検索にかけて頻出単語表を作成したものと既存の各アルゴリズムに登録された話題から作成した頻出単語表と比べて，最も似ている頻出単語表をもつ解析アルゴリズムを選択している．

この様な構造を用いることで最適なアルゴリズムが選択されると考えた．例えば以下の**図 3.3**「クッパ[*6]が落ちた」という入力を行った時には「クッパ　落ちた」というキーワードで検索を行い，その結果から頻出単語表を作成する．

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 の検索結果を見ると，私の期待通りスーパーマリオブラザーズ（以下マリオ）に関係する検索結果を取得できていることがわかる．この検索結果に出てくる単語の中から固有名詞だけを取り出し頻

[*5] 独自実装を行った Google を用いて検索を行う機能
[*6] 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

出単語表を作成することでマリオに関連のある単語表が完成する．マリオ専用のアルゴリズムがある場合，同じ工程で「マリオ」の検索結果をもとに頻出単語表が作られる．今回作成した「マリオ　落ちた」で作成した頻出単語表とアルゴリズム側に設定した話題である「マリオ」の検索結果で作成した頻出単語表は，「料理」などの他の単語で検索した際の頻出単語表と比較して一番頻出単語が似ているためマリオの解析アルゴリズムが選択される．

この時マリオに特化したアルゴリズムの他に料理に特化したアルゴリズムも実装されているとする，その状態で「クッパ[*7]って美味しいよね」という入力があった場合については次のようになる．

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を取得している事がわかる．同じクッパという単語は出現するが，料理で検索した結果をもとに作成した頻出単語表の方が出現固有名詞が似ているため，マリオのアルゴリズムが選択される事はなく，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ように設計している．

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

このようにその場の話題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ることで，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「クッパ」が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．

「クッパが落ちた」「クッパって美味しいよね」の両方の文章を適切に解析できる，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選択されるようにする事もできるような設計とした．

[*7] 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

3.1.3　解析アルゴリズムを簡単に追加する機構

解析を行うアルゴリズムを簡単に追加する機構について解説する．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[*8]を実装することで追加することが可能である．その実装の手順はソースコードの行数に換算すると，アルゴリズムの解析を行う話題の設定，アルゴリズム自体，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，子抽象クラスの話題を設定し，アルゴリズムを実装後，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，作成したアルゴリズムを入れることで実装を完了することができる．

話題や感情などを解析する話題解析知能ハブなどの，親の抽象クラスを実装したものが，子の抽象クラスを実装したアルゴリズムを複数所持する構図となり，その関係性を**図 3.5** に示す．

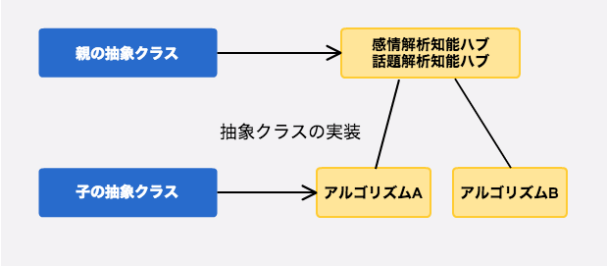

図 3.5　抽象クラスの関係と抽象クラスの実装例

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．

[*8] アルゴリズムを追加する際に用いる抽象クラスであり，アルゴリズム以外の必要な処理が記述されている抽象クラス

3.2 解析結果を保存する機構

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（以下データベース）が実装されている．また，データベー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

データベースは，解析した情報を保存する機能がある．情報を保存する際に保存を行うデータ型は複数あるが，解析結果がどのような形式でも保存が可能な様に 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に定義済みの変数に対して値を入れるだけで自動で保存が行われ，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される．この構成により複数の製作者がいる際にデータの衝突が起きない様に設計されている．

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際に予め定義されている変数に値を入れるだけで良く，変数に保存した情報は Mode-Topic-Game-String という名前が自動生成され，データベースへと保存される．

3.2.2 解析情報を取得する機能

解析した情報をデータベースから取得する方法について解説する．

データの取得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に用いた，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）の最後を見てその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 のキャラクターに命令を送信する工程について，キャラクターと会話をする例を**図 3.6** に示し，これを用いて説明する．

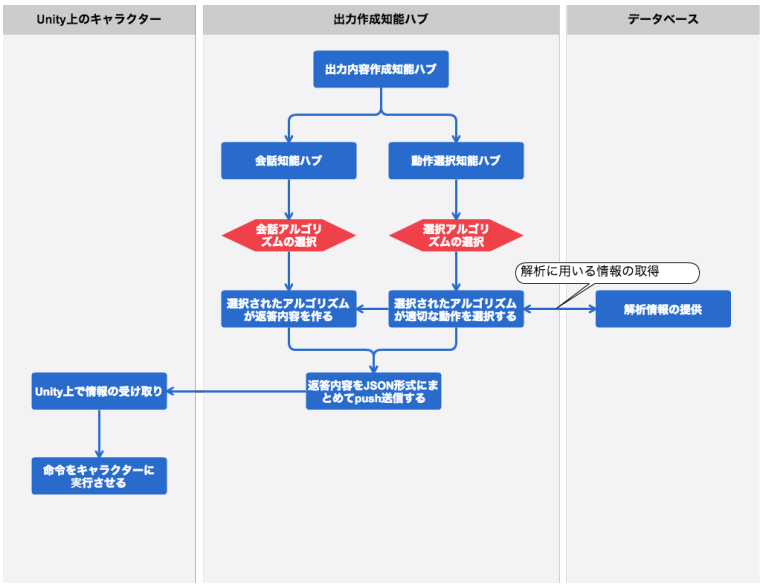

図 3.6　出力情報を作成するまでの流れ

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る．**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話題を推定し最適なアルゴリズムを選択する．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 通信を用いてウェブブラウザ上の Unity で開発されたキャラクター (以下キャラクター) へと送信され，キャラクターが命令を解釈，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や命令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，返答を行うものである．

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う．特定の時間や変数の値，特定の感情値になった際に発言を行うように設定することが可能である．出力内容作成知能ハブの自発的に発言する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で自発的な発言が可能となって

いる．

感情値や時間経過，状況の変化があった時に websocket を用いて Unity へ任意のタイミングで命令が可能なため，自発的に発言しているように見せることが可能である．

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と考え，**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それぞれ用意されている．

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．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．

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キャラクターとの対話を可能にする環境の作成に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可能である．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り，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．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，従来のサーバーへ常時アクセスをする方法よりも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 への命令の送信は，汎用性の高い JSON 形式[*9]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．

[*9] 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 GooogleAPI とは独自に開発した Google 検索を応用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 HttpCleaner で整形し，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，固有名詞とその単語がでてきた回数を保存するプログラムである．

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．加えて一見「料理」という単語と「ごはん」という単語は文字列だけを比較しても関連性はないように見えるが，この 2 つの検索結果の頻出単語表には料理という単語が含まれており単語表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．

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がもつ「料理」の検索結果から作成した単語表にも同じく「パッド」という単語は複数回出ており，料理のアルゴリズムが選択される．

これは同じウェブサイトである，「クックパッド[*10]」が表示されることを活用しているからである．

[*10] 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ある．

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．ライブラリのバージョン管理をソースコードで行える利点もある．

4.2 解析部分の実装

4.2.1 解析コントローラー

解析コントローラーでは，「話題解析」や「感情解析」と言った解析を行う分野自体を管理するクラスである．今回はその解析分野[*1]別に解析アルゴリズムを保持する，各解析知能ハブ[*2]を作るために InputController を作成した．InputController 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　実装した主要構成要素

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「感情」や「話題」などの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一部抜粋したものを以下の**リスト 4.1** に示す．

[*1] 解析分野：話題を解析する場合の解析分野は話題であり，感情を解析する場合の解析分野は話題となる．
[*2] 話題解析知能ハブや感情解析知能ハブなどの解析分野別に作成されるアルゴリズムを複数所持するクラス

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の一部（ 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:     }
14:     public void InputData(String inputJson) throws IOException {
15:         basicData = JSON.decode(inputJson, InputBasicData.class);
16:         for (int i = 0; AnalyzeMode.size() > i; i++) {
17:             AnalyzeMode.get(i).SetBasicData(basicData);
18:             switch(basicData.getMode()){
19:                 case 1:
20:                     AnalyzeMode.get(i).analyzeChat();
21:                     break;
22:                 case 2:
23:                     AnalyzeMode.get(i).analyzeTouch();
24:                     break;
25:                 case 3:
26:                     AnalyzeMode.get(i).analyzeAddObj();
27:                     break;
28:             }
29:         }
30:     }
31: }
```

リスト 4.1 の 6 行目 8 行目 10 行目で解析分野を，この解析知能ハブに登録している事がわかる．

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．その情報を用いて各解析知能ハブはそれぞれの解析分野にあった内容を解析する．今回の場合は**表 4.1** の 20 行目で実際に各解析知能ハブの解析をおこなうメソッドを呼び出している．

4.2.2　解析知能ハブ

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これが解析知能ハブの抽象クラスである．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 の init メソッドでは各解析アルゴリズムが保持している話題を，独自実装を行った GoogleAPI に渡すことで頻出単語表を作成し，その検索結果の頻出単語表を取得している．

表 4.2　Abstract Mode に実装した主要構成要素

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 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表[*3]とユーザーが入力した文章から作成した頻出単語表を比較して，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選択される実装となっている．

4.2.3　解析アルゴリズムの各解析知能ハブへの追加

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追加実装する構成について説明する．各解析知能ハブに対してアルゴリズムを追加実装するために用いる，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が記述されている．アルゴリズムを新規に追加して試したい場合，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 に実装した主要な要素を示す．

表 4.3　Abstract Mode の主要構成要素

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	アルゴリズムの話題を記述する場所
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る．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自動で保存される．

表 4.3 のコンストラクタでは，その解析アルゴリズムが解析を行う専門の「話題」を記述する必要性がある．具体的には about という String 型の変数に話題を入れることで，その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て返答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一部抜粋したものを以下の**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の一部 (about.java)

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

[*3] 頻出単語表：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

```
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，コンストラクタを抜粋したものだが，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の一部 (anaAlgo.java)

```
1: public class Mode_Topic_Ryori extends Abstract_Mode_parts {
2:   @Override
3:   public boolean ChatAnalyze(String chat) {
4:     //この例は一番簡単なクラス名と入力内容をつなげて保存を行うだけの物
5:     stringMap.put("話題","料理の話題：" + chat) //アルゴリズムの記述
6:     return true;
7:   }
8: }
```

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．

次にデータを保存する際に用いるメソッドについて解説を行う為，継承元クラスのソースコードを一部抜粋したものを以下の**リスト 4.4** に示す．

リスト 4.4　継承元クラスから一部抜粋した定義済みのメソッド (anaParts.java)

```
1: public class Abstract_Mode_parts{
2:   public boolean saveData() {
3:     database.showData();
4:     String className = this.getClass().getSimpleName();
5:     if (!stringMap.isEmpty()) {
6:       database.setData(className + "_stringMap", stringMap);
7:       database.showData();
8:       return true;
9:     } else if (!longMap.isEmpty()) {
10:      database.setData(className + "_longMap", longMap);
11:      database.showData();
12:      return true;
13:    } else if (!integerMap.isEmpty()) {
14:      database.setData(className + "_integerMap", integerMap);
15:      return true;
16:    } else if (!booleanMap.isEmpty()) {
17:      database.setData(className + "_booleanMap", booleanMap);
18:      return true;
19:    } else if (!objectMap.isEmpty()) {
20:      database.setData(className + "_objectMap", objectMap);
21:      return true;
22:    } else if (!doubleMap.isEmpty()) {
23:      database.setData(className + "_doubleMap", doubleMap);
24:      return true;
25:    } else if (!maps.isEmpty()) {
26:      database.setData(className + "_maps", maps);
```

```
27:         return true;
28:       } else if (!stack.empty()) {
29:         database.setData(className + "_stack", this.stack);
30:         database.showData();
31:         return true;
32:       }
33:
34:       return false;
35:     }
36:   }
```

リスト 4.4 の saveData は，継承元のクラスに定義されているメソッドである．このメソッドは全ての解析が終わった後に親抽象クラスから呼び出され，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て保存する．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や話題解析知能ハブなどの親抽象クラスに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を行うプログラムである．話題の解析に関しては既に多くな枠としては解析が終わっているが，「ゲーム」という話題の時にさらに細かい「どのゲームか」ということや「どんなシーンなのか」などをさらに詳しく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．感情の解析を行うアルゴリズムは，親抽象クラスを実装した感情解析知能ハブに所属する解析知能の 1 つにあたる．現在感情の解析を行うプログラムは 1 つなため，必ずこのアルゴリズムが選択されるようになっている．

具体的なアルゴリズムとしては「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設定されており，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変数型の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 の鍵[*4]を String にし，value[*5]を Object 型に指定した．

[*4] 情報を取得する際に利用する文字列
[*5] 鍵に対応付けられる値

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から一部抜粋したものを示す．

リスト 4.5　データベースの解析結果を保存するメソッドの一部（dbIn.java）

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

リスト 4.5 の setData メソッドは先ほどの 4.2.3 で説明した saveData メソッドから呼び出されて利用される．保存を行う際にはデータの重複が起きないようにクラス名＋データ型をデータに対応するキーとしている．

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 4.6** に getData メソッドのソースコードを示す．

リスト 4.6　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 4.6 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とがわかる．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情報を取得するための鍵を調べ，その鍵を用いて情報を取得する．

4.4　出力を行う知能ハブの実装

Unity へ命令の送信を行う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 形式などを成形する出力コントローラーについて解説する．出力コントローラーに，実装した主要な構成要素の一覧を以下の**表 4.4** に示す．

表 4.4 のコンストラクタの部分では，各出力知能ハブをこのクラスに登録している．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　実装した主要な構成要素

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 getJson メソッドでは，Unity から入力があった際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実装になっている．

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ことで応答を行う．

4.4.2　出力知能ハブ

動作や返答内容などの出力情報別に，それぞれアルゴリズムを保持する機構について解説する．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．その抽象クラスを実装することで返答知能ハブや動作選択知能ハブなどのアルゴリズムを複数所持するクラスを新規に作成することが可能である．

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 の主要構成要素を示す．

表 4.5　実装した主要な構成要素

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処理を行っている．

表 4.5 の getOutput メソッドでは，キャラクターに対して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返答を行う．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，アルゴリズムの選択はユーザーが発言した内容から作成した頻出単語表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 HashMap を比較して，もっとも頻出単語表が似ているものを選ぶという仕組みになっている．その部分のソースコードから一部抜粋したものを**リスト 4.7** に示す．

リスト 4.7　getOutput.java のソースコードの一部 (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 4.7** の 8 行，9 行目で最新の発言情報を取得し，10 行目で発言内容を GoogleAPI に渡すことで頻出単語表を作成する．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表と GoogleAPI を用いて取得した頻出単語表を 11 行目から 25 行目にかけて比較し，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て取得するメソッドである．

4.4.3 出力アルゴリズムの出力知能ハブへの追加

出力アルゴリズ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 という子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話題に対応した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** に示した主要な要素について解説する．

表 4.6　実装した主要な要素

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

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．解析を行う際と同じように変数 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表が自動で生成される．その生成された頻出単語表も解析の時と同じく先ほど説明したアルゴリズムの選定に利用される．以下の**リスト 4.8** に挨拶の分野を指定する場合のプログラムを一部抜粋したものを示す．

リスト 4.8　話題を指定する際のサンプルソースコードの一部 (aisatu.java)

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

リスト 4.8 の 3 行目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ようになるように実装されている．

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追加されており，料理とゲームの話題に特化したアルゴリズムである．

料理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．

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とその 1 つ 1 つの動作に関係する単語，及びその関係する単語を GoogleAPI にかけた結果である頻出単語表を取得する．

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おり，その具体的なアルゴリズムや通信に関しては 4.6 章に記述する．

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 4.9** にソースコードを一部抜粋したものを示す．

リスト 4.9　WebSocket の Java サーバー側実装の一部 (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 4.9 のソースコードではウェブブラウザ（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為に 23 行目でリストにセッションを格納している．この実装により，セッションが確立され，送受信を行う準備が完了する．

Unity からメッセージが来た場合は**リスト 4.9** の 44 行目の processUpload が呼ばれる．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るが，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ものである．

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 型に変換している．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ている．その処理が終わった後に 49 行目にて返答する json 形式の返答情報を作成し，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 4.9** 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 4.9 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では複数のクライアントが接続された際にはクライアントごとに別のキャラクターと会話を行うのではなく，1 つのキャラと全員で会話をう事も目標としている．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 章で説明した，動作選択を行う際に用いるアルゴリズムについて解説する．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 4.10　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 4.10** 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る．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，その中でキャラクターが行うモーションを選択している．**リスト 4.10** 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る．このコントローラーはサーバー上の 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の実装に関しては，共同開発の鈴木さんの論文 [7] 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と情報の取得を行なっていることがわかる．

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を解説する．**リスト 4.10** の 22 行目から 39 行目を見ると，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 つの頻出単語表を比較し，一番似ている頻出単語表を持っている動作が選択され，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，kuromoji を用いて形態素解析を行わせる機能，及び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からどのような分野の単語なのかということをウェブ上から検索を行って調べるにあたり，その検索する際の検索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精度に関わるため，非常に重要である．

そこで Java の形態素解析器である kuromoji[12] を用いて形態素解析を行い，適切な検索ワードで検索を行えるように mwSoft blog[11] を参照，参考にしつつ実装を行った．

具体的には kuromoji には Search モードというモードがあり，それを利用することで「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することが可能である．この形態素解析に加えて不要な文字である「しよ」などの文字列の削除を行う事で，より良い検索結果を取得できると考えられる．

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文字として取得することが可能である．この機構を実現するために mwSoft blog[10] を参考にし，参考にしたソースに加えて並列初期化機能などの複数の機能を独自実装した．

以下の**リスト 4.11** 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 4.11　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 4.11 の 20 行目にて Google の検索結果ページの URL を作成し，通信することで Google 検索の結果の内容を取得している．

リスト 4.11 の 23 行目から 37 行目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 4.11 の 47 行目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」といった検索を行った際に表示される文字列や「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 4.11** の 84 行目から 108 行目である．84 行目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 4.12　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 4.12 の 7 行目に定義されている 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可能である．引数としては border を設定することができ，n 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可能である．

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ため，実装では**リスト 4.12** の 11 行目にて 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 4.12** の 14 行目から 24 行目にて行っている．最後にカウントした結果を HashMap 形式にし，41 行目でその HashMap を返していることがわかる．以下の**リスト 4.13** 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 4.13　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， キッコーマン=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 4.13 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可能な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，調味料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可能である．

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可能である．

第 5 章

実行結果

5.1　Unity の出力画面の図

実際に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可能である.

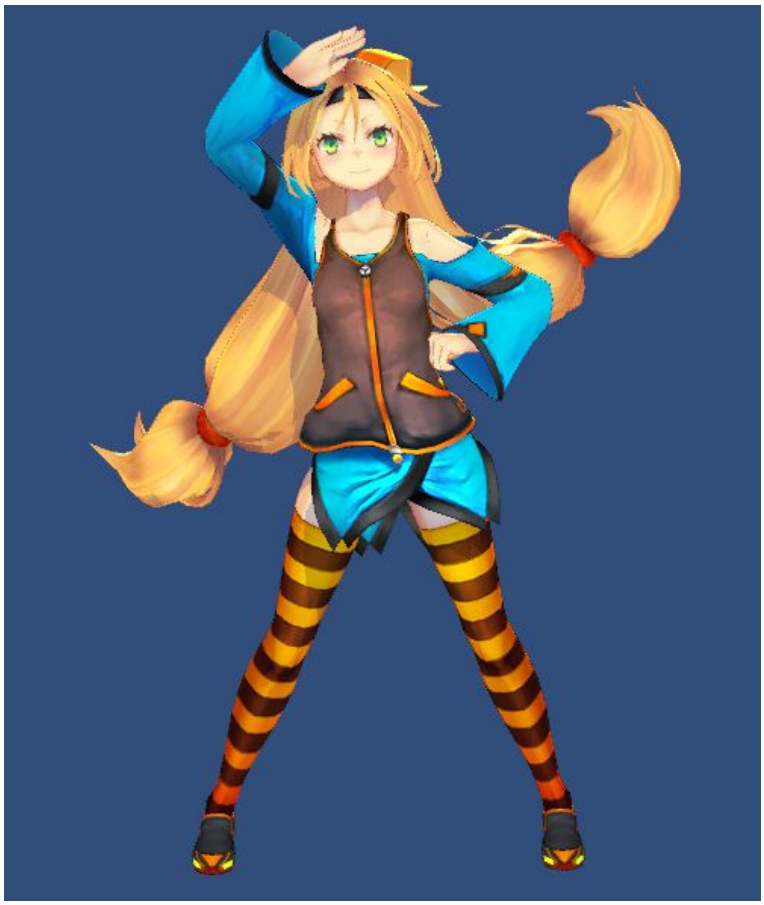

図 5.1　キャラクターとの対話画面

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を確認することが可能である.

5.2 実際の会話

実際に人工知能利用フレームワークを用いて会話を行う例を以下の**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	隠し味にチョコレートいれちゃう？	悪巧みの動作
うわお！クッパ強い，負けたよ	ゲームオーバーだねー	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から，現在の会話の話題を解析することでその話題に対応している解析や出力を行うアルゴリズムが処理を行う仕組みにより，詳しい解析や出力を行うことが可能となっていることが**表 5.1** を見ることでわかる.

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，どの様な変化があるかを検証する.

今回追加を行うアルゴリズムの話題はゲームの「スーパーマリオブラザーズ[*1]」(以下，マリオ）という話題をもつ解析と出力を行うアルゴリズムである．話題を解析するアルゴリズムは，マリオに特化したさらに細かい話題の解析を行うアルゴリズムを実装した.

ユーザーと会話を行う際に用いる，返答アルゴリズムに関しても「マリオ」の話題に対応を行い，マリオの話題に対して特化している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.

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

この「マリオ」という話題は，ゲームという大枠の中の単一ゲームタイトル名である．そのためユーザーが話しかけた内容が，単に幅広くゲームに関連のある単語であれば，ゲームの話題が解析を行う．しかしここで「マリオ」というゲームに登場する敵キャラクター「クッパ[*2]」という単語を含めることにより，**表 5.2** の様にマリオの解析と出力に特化したアルゴリズムが選択される.

返答内容に関しても「負けたよ」という発言から，このゲームのプレイヤーを強化することができるアイテムである「きのこ」を取得して再度，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.

[*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」

この様に料理やゲームなどの会話を行う際に対応する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解析を行うことができるようにするアルゴリズムの追加方法の両方に対応している．

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用いて，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が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，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アルゴリズムも含めて作成したい場合は，他のアルゴリズムを登録する事が出来ないのが現状であり．その場合既存の他のアルゴリズムが利用できなくなるという弊害があるのが現状である．

6.1.1 アルゴリズムの追加による出力の変化

5.3 章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解析や応答をしていることが分かる．また，解析を行うアルゴリズムの追加に関しても，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事がわかる．

この通りアルゴリズムを追加することにより，より高精度な解析や出力を行うことができた．

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，アルゴリズムを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を作ることができた．

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と考えられる．

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た．これにより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

かんしゃかんしゃ

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] Microsoft：“りんな” http://rinna.jp/rinna/ (2015 年 1 月 12 日).

[4] Softbank： “pepper” http://www.softbank.jp/robot/consumer/products/ (2015 年 1 月 12 日).

[5] 東京工科大学：“東京大学松尾 豊” http://ymatsuo.com/japanese/ (2015 年 12 月 27 日).

[6] 藤井克成：“Unity によるキャラクターの制御” (卒業論文　2015 年度).

[7] 鈴木智博：“モーションデータベースの開発” (卒業論文　2015 年度).

[8] 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).

[9] マルチェッロ・マッスィミーニジュリオ・トノーニ 著：“LaTeX 2_ε 意識はいつ生まれるのか” (亜紀書房).

[10] 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).

[11] 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).

[12] github： “形態素解析器 kuromoji” https://github.com/atilika/kuromoji (2015 年 8 月 12 日).