プログラミング初心者のための Blocklyを用いた音声対話シナリオ編集システム

古市 瑞希[1] 山本 大介[1] 高橋 直久[1]

概要：我々の研究室では音声インタラクション構築ツールキット MMDAgent を用いて，一般ユーザによる利用を前提とした音声対話システムの研究を行なっている．MMDAgent は，FST（有限状態遷移の）形式で記述された対話シナリオを編集することで対話の内容を自由に構成することができる．しかし，FST 形式の対話シナリオの編集には一定の知識が必要となり，一般ユーザが編集するのは困難である．また，テキスト形式による記述のため，対話シナリオの把握が困難で，記述ミスが起きやすいという問題点がある．これらを解決するために，Google Blockly を用いた音声対話シナリオ編集システムを提案する．また，提案システムに基づいてプロトタイプシステムを実装し，評価実験を行った．その結果，提案システムを使用することで短い時間で簡単に，見やすく分かりやすい対話シナリオを作成できることが分かった．

Voice Interaction Scenarios Editor using Blockly for Programming Beginners

MIZUKI FURUICHI[1] DAISUKE YAMAMOTO[1] NAOHISA TAKAHASHI[1]

1. はじめに

我々の研究室では音声インタラクション構築ツールキット MMDAgent[1] を用いて，一般ユーザによる利用を前提とした音声対話システムの研究を行っている．MMDAgent は，FST 形式で記述された対話シナリオを編集することで対話の内容を自由に構成することができる．FST 形式の対話シナリオは，状態番号，次状態番号，遷移条件のコマンド，出力のコマンドの４つ組から構成されている．FST 形式の対話シナリオを編集するには，FST 形式の構文を学習する必要がある．また，テキスト形式による記述のため，対話シナリオの把握が困難で，記述ミスが起きやすいという問題点がある．

このような問題を解決するために，我々は Google Blockly[2] を用いた音声対話シナリオ編集システムを提案する．Google Blockly を用いることで，分岐構造やサブルーチン構造などの制御構造を使って対話シナリオを表現できるため，FST 形式の状態遷移の構造になじみのないユーザでも編集が簡単になる．また，Google Blockly のブロック（以下“ブロック”という）を組み合わせることで対話シナリオが表現できるため，プログラミングになじみのないユーザでも編集が簡単になる．

提案システムの実現には，ブロックを用いて作成した対話シナリオを FST 形式の対話シナリオに変換するために Google Blockly と FST 形式の対応付けが必要である．そのため，以下の課題とそれに対するアプローチが考えられる．

課題 1. ブロックと FST 形式の遷移条件のコマンドと出力のコマンドの対応付けが必要である．この課題に対するアプローチとして，FST 形式の遷移条件のコマンドと出力のコマンドに対応したブロックを提案する（以下“命令ブロック”という）．

課題 2. Google Blockly は，ブロックから分岐構造やサブルーチン構造などの制御構造を持つ言語（JavaScript や Python など）のコードへの変換を対象としているが，FST 形式は状態遷移の構造で Google Blockly が対象としている言語の構造とは異なる．そのため，ブロックと FST 形式の構造的な対応付けが必要である．この課題に対するアプ

[1] 名古屋工業大学大学院工学研究科
Graduate School of Engineering，Nagoya Institute of Technology

図 1　MMDAgent 動作画面

ローチとして，ブロックと FST 形式の構造的な対応付けをするブロックを提案する（以下“構造ブロック”という）．

課題 3. ブロックから FST 形式に変換するために状態番号の管理が必要である．この課題に対するアプローチとして，ブロックごとに状態番号を計算し，FST 形式の対話シナリオに変換する機能を実装する．

2.　関連研究

2.1　MMDAgent

MMDAgent は，名古屋工業大学国際音声技術研究所で開発された音声対話システム構築ツールキットである．音声認識，音声合成，3D モデルの描画と物理演算などの機能を統合したシステムで，音声認識では Julius を，音声合成では Open JTalk を，3D モデルでは MikuMikuDance 形式を，物理演算では Bullet Physics を採用している．MMDAgent は対話シナリオの記述形式に有限状態オートマトンの一種である有限状態トランスデューサ (FST) を採用している．FST 形式の対話シナリオが記述されたファイルを FST ファイルと呼ぶ．この FST ファイルを編集することで対話シナリオを自由に構成することができる．MMDAgent の実行画面を図 1 に示す．ユーザが画面中央のキャラクタ“メイちゃん”に話しかけると，認識した文字列に対応した言葉を音声によって返したり，“お辞儀をする”動作や“手を振る”動作をしたりする．

2.2　Google Blockly

Google Blockly とは，ビジュアルプログラミングエディタを構築するための JavaScript ライブラリである．ビジュアルプログラミングとは，プログラムをテキストで記述するのではなく，視覚的なオブジェクトでプログラミングすることである．近年，子どもでも簡単にプログラミングできるとして，Scratch[3] や viscuit[4] などのビジュアルプログラミング言語が普及している．Google Blockly では，パズルのようにブロックを組み合わせることでプログラムを作成することができ，JavaScript や Python のプログラムに自動で変換される．ユーザはブロックを選択し，組み合わせるだけでプログラミングできるため，JavaScript や Python の構文を学習し，覚える必要がない．

2.3　EFDE

EFDE[5][6] は，タブレット端末のタッチ操作で状態遷移図を描くことにより MMDAgent の対話シナリオを作成できるシステムである．作成したい処理に近い FST テンプレートを選択し，データを入力することで手軽に編集できるという点が本研究と同様である．EFDE では，状態遷移図を描くことで対話シナリオを作成編集しているが，本研究では，ユーザは状態の遷移について考えることなく，ブロックを組み合わせることにより対話シナリオを編集する手法を提案している点で EFDE とは異なる．

2.4　MMDAE

MMDAE[7][8] とは，音声対話エージェントのための Web ブラウザを用いたシナリオエディタである．対話シナリオを見やすくし，編集しやすくして，より簡単に扱えるようにするという点では本研究と同様である．しかし，MMDAE は FST 形式で書かれた対話シナリオを編集しやすくするエディタであり，FST 形式より編集しやすい対話シナリオ編集システムを提案している点で本研究とは異なる

3.　FST 形式の対話シナリオ

MMDAgent の対話シナリオは FST 形式で記述される．FST 形式の構文は，状態番号，次状態番号，遷移条件のコマンド，出力のコマンドを [space] を区切り文字として一行に記述し（“文”という），複数の文を系列的に記述することで対話シナリオとする．無条件遷移や無出力の場合には“<eps>”コマンドを記述する．

FST 形式で記述された対話シナリオの例を図 2 に，その対話シナリオに対応した状態遷移図を図 3 に示す．図 2 の状態番号が 1 のような状態を初期状態といい，MMDAgent が何も動作をしていないときは，基本的にこの初期状態にある．図 2 の 1 行目は，MMDAgent が状態番号 1（初期状態）にあるときに音声認識によって「こんにちは」と認識すると，無出力で次の状態である状態番号 10 の状態へ遷移するということを表す．図 2 の対話シナリオは，音声認識によって「こんにちは」と認識すると，MMDAgent のキャラクタ“メイちゃん”が「お辞儀」の動作をし，「こんにちは」と発話するという流れである．

また，複数の対話シナリオを並列して記述することができる．そのときの記述のルールは以下である．

(1) 1 つの対話シナリオは，初期状態から始まり，初期状

```
 1  10  RECOG_EVENT_STOP|こんにちは <eps>
10  11  <eps>                        MOTION_ADD|mei|greet|greet.vmd
11  12  <eps>                        SYNTH_START|mei|normal|こんにちは
12  1   SYNTH_EVENT_STOP|mei         <eps>
```

図 2　FST 形式の対話シナリオの例

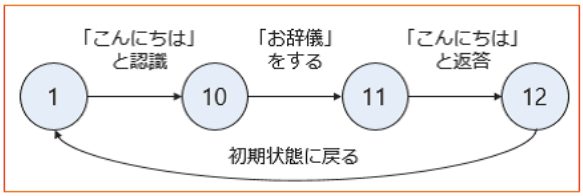

図 3　図 2 の対話シナリオに対応した状態遷移図

態で終わる．

(2) 対話シナリオごとに初期状態から遷移する状態番号が異なる．

4. 提案システム

この章では，提案システムの構成と，命令ブロックと構造ブロック，対話シナリオ変換機能の詳細について述べる．

4.1 提案システムの構成

本稿では，図 4 のように，以下の機能からなる音声対話シナリオ編集システムを提案する．

機能 1. 対話シナリオ作成機能

ユーザがブロックを用いて対話シナリオを作成できる機能である．機能実装のため，命令ブロックと構造ブロックを定義する．ブロックの形や色，そのブロックに対応した FST 形式のコマンドを定義し，ツールボックスにブロックの一覧を表示する．ユーザはツールボックスからブロックを選択し，ワークスペースで組み合わせることで対話シナリオを作成する．命令ブロックと構造ブロックの詳細は，次節以降で述べる．

機能 2. 対話シナリオ変換機能

ブロックの対話シナリオから FST 形式の対話シナリオに変換する機能である．変換の手順は，はじめにブロックから FST 形式の文へ変換する．そして，変換された文の系列に対して，文の状態番号と次状態番号の整合性を保つように，文の状態番号を付け替える．提案システムでは Google Blockly の変換機能を流用できるようにするため，ブロックごとに文を再配置し，FST 形式の対話シナリオを生成する．生成された FST 形式の対話シナリオはテキストエリアに表示される．対話シナリオ変換機能の詳細は，次節以降で述べる．

機能 3. FST ファイル出力機能

システムにより生成された FST 形式の対話シナリオを MMDAgent が受理可能な形式の FST ファイルに出力する機能である．

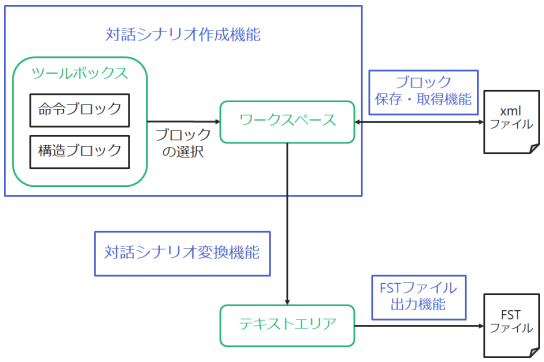

図 4　提案システムの構成図

図 5　UNIT ブロック

機能 4. ブロックの保存・取得機能

ブロックを xml 形式で保存する機能，および，保存済みのブロックを取得してブロックを再利用可能にする機能である．

4.2 命令ブロック

命令ブロックとは，FST 形式の遷移条件のコマンドや出力のコマンドに対応し，FST 形式について知識がない人でもコマンドの内容が分かるように表したブロックである．ツールボックスに表示されるブロックが対話シナリオの編集で使用できるコマンドの一覧となるため，コマンドを覚える必要がなくなる．また，ブロックを用いることでテキスト入力が減るため，記述ミスが減ると考えられる．

4.3 構造ブロック

構造ブロックとは，ブロックと FST 形式の構造的な対応付けをするブロックである．次の 4 つの構造ブロックを提案する．

4.3.1 UNIT ブロック

UNIT ブロックとは，一つの対話シナリオを一つのブロック群“UNIT”として扱うためのブロックである．UNIT ブロックに挟まれたブロックは前述した FST 形式の記述のルールが適用される．複数の“UNIT”を作ることで，複数の対話シナリオを作成することができる．UNIT ブロックを図 5 に示す．

4.3.2 分岐ブロック

分岐ブロックとは，ある状態から異なる 2 つ以上の状態へ遷移する場合に使用するブロックである．

分岐ブロックの使い方を次に示す．例として，ある状態 A から異なる 2 つの状態 B，C に遷移する場合を考える．

(1) ブロック A の次に二股の分岐ブロックを接続する．

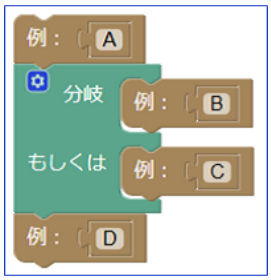

図 6　分岐ブロックの使用例

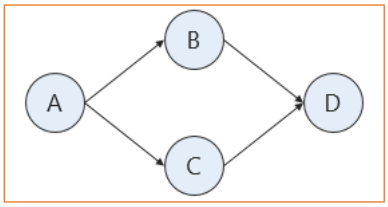

図 7　図 6 のブロックに対応した状態遷移図

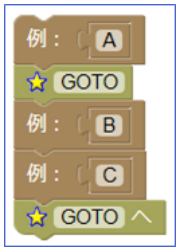

図 8　GOTO ブロックの使用例

(2) 二股のそれぞれにブロック B とブロック C を挿入する．

分岐ブロックの使用例を図 6 に，そのブロックに対応した状態遷移図を図 7 に示す．この例は，状態 A，B，D と遷移する場合と状態 A，C，D と遷移する場合を表す．

4.3.3　GOTO ブロック

GOTO ブロックとは，指定された状態へ無条件で遷移する場合に使用するブロックである．GOTO の遷移を開始するブロックを“GOTO”ブロック，遷移先のブロックを“GOTO_STAR”ブロックという．

GOTO ブロックの使い方を次に示す．

(1) GOTO の遷移を開始場所に“GOTO”ブロックを挿入する．

(2) GOTO の遷移先に“GOTO_STAR”ブロックを挿入する．

GOTO ブロックの使用例を図 8 に，そのブロックに対応した状態遷移図を図 9 に示す．この例は，状態 A，B，C と遷移した後に，再び状態 B へ遷移する場合を表す．

4.3.4　関数ブロック

関数ブロックとは，対話シナリオの中で何度も必要とされる定型的な処理を一つにまとめて，外部から関数マクロとして呼び出す場合に使用するブロックである．関数の本体となるブロックを“FUNCTION”ブロック，関数を呼

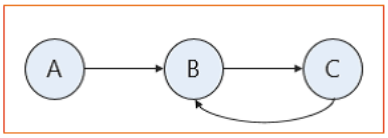

図 9　図 8 のブロックに対応した状態遷移図

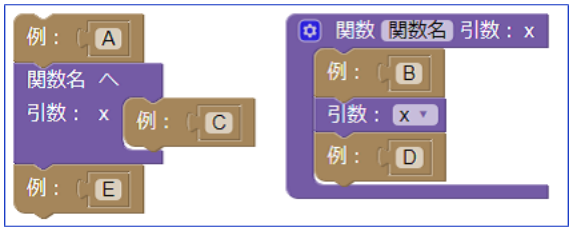

図 10　関数ブロックの使用例

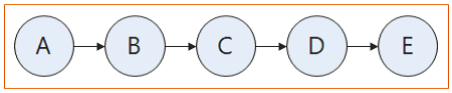

図 11　図 10 のブロックに対応した状態遷移図

び出すブロックを“FUNCTION_CALL”ブロック，引数を呼び出すブロックを“ARGMENT”ブロックという．

関数ブロックの使い方を次に示す．

(1) “FUNCTION”ブロックを作成する．関数の名前，引数の数と名前を決め，一つにまとめたいブロックを“FUNCTION”ブロックの間に挟む．

(2) “FUNCTION_CALL”ブロックを関数を呼び出したい場所に挿入する．引数がある場合には，引数として渡したいブロックを挿入する．

(3) “ARGMENT”ブロックを引数を呼び出したい場所に挿入する．

関数ブロックの使用例を図 10 に，そのブロックに対応した状態遷移図を図 11 に示す．この例は，C のブロックを引数 x として関数“関数名”を呼び出し，状態 A，B，C，D，E と遷移する場合を表す．

4.4　対話シナリオ変換機能

対話シナリオ変換機能とは，ブロックの対話シナリオから FST 形式の対話シナリオに変換する機能である．ブロックの組み合わせに応じて，自動で FST 形式に変換するため，ユーザは FST 形式の構文について考えることなく対話シナリオを作成することができる．

5.　実現法

この章では，提案システムの実現法について述べる．

5.1　命令ブロック

Blockly Developer Tools[9] を使用して，FST 形式の遷移条件のコマンドと出力のコマンドに対応したブロックを作成する．

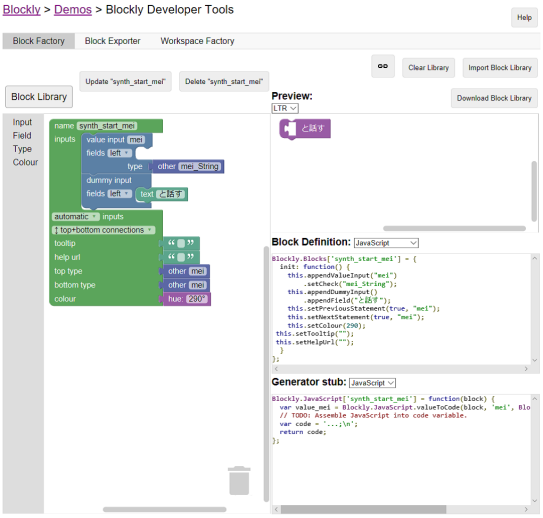

図 12　Blockly Developer Tools

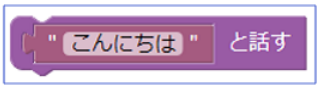

図 13　「こんにちは」と発話するブロック

図 14　お辞儀の動作をするブロック

図 15　分岐ブロック

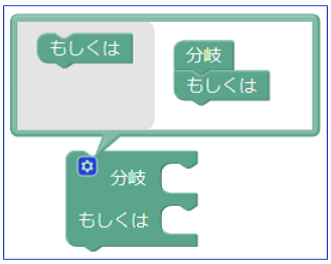

図 16　“Mutator UI”の表示

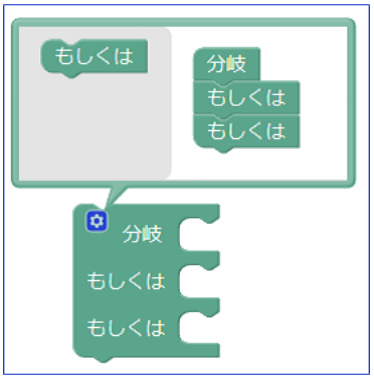

図 17　分岐ブロック変形後

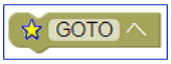

図 18　“GOTO”ブロック　図 19　“GOTO_STAR”ブロック

Blockly Developer Tools を図 12 に示す．Blockly Developer Tools とは，ブロックを簡単に作成できるツールで，元となるブロックにブロックの名前，形，色などを記述すると，ブロックのプレビューと JavaScript 形式のブロックの定義が表示される．

命令ブロックの例として，「メイちゃんが「こんにちは」と発話する」ブロックを図 13 に，「メイちゃんがお辞儀の動作をする」ブロックを図 14 に示す．

5.2　構造ブロック

命令ブロックと同様に，Blockly Developer Tools を使用してブロックを作成する．

5.2.1　UNIT ブロック

UNIT ブロックに数字入力欄を設け，入力された数字によって“UNIT”を識別する．今回は式 1 により各“UNIT”の番号を算出した．複数の対話シナリオを作成する際には“UNIT”ごとに異なる数字を入力することで，それぞれの“UNIT“が初期状態から異なる状態へ遷移するように状態番号が付与される．

$$\text{“UNIT”の番号} = \text{入力された数字} \times 100 \qquad (1)$$

5.2.2　分岐ブロック

分岐ブロックには，Google Blockly の Mutators 機能 [10] を利用する．Mutators 機能とは，ユーザが動的にブロックの形状を変更できる機能である．提案システムでは，ユーザが任意の数の分岐を行うブロックを作成することができる．作成した分岐ブロックの初期形状は図 15 である．分岐ブロックの左上にある“ダイアログボタン”という青色の歯車のマークを押すと，図 16 のように“Mutator UI”が表示される．“Mutator UI”内の“分岐”ブロックに“もしくは”ブロックを接続すると，図 17 のように分岐ブロックの形が変化する．

5.2.3　GOTO ブロック

GOTO の遷移を開始する“GOTO”ブロックと，遷移先の目印となる“GOTO_STAR”ブロックの 2 つのブロックを作成する．それぞれのブロックにはラベルが入力でき，同じラベルの“GOTO”ブロックから“GOTO_STAR“ブロックへ遷移する．“GOTO”ブロックを図 18 に，“GOTO_STAR”ブロックを図 19 に示す．

5.2.4　関数ブロック

関数の本体となる“FUNCTION”ブロックと，関数を呼び出す“FUNCTION_CALL”ブロック，引数を呼び出す“ARGMENT”ブロックを作成する．GOTO ブロックと同様に，“FUNCTION”ブロックと“FUNCTION_CALL”ブロックにはラベルが入力でき，同じラベルの“FUNCTION_CALL”ブロックから“FUNCTION”ブロックが呼び出される．“FUNCTION”ブロックを図 20 に，“FUNCTION_CALL”ブロックを図 21 に示す．

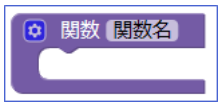

図 20 “FUNCTION” ブロック

図 21 “FUNCTION_CALL” ブロック

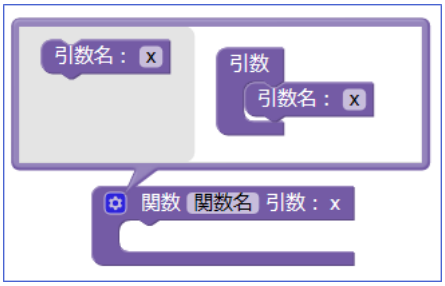

図 22 “FUNCTION” ブロック (引数が 1 つの場合)

図 24 “ARGMENT” ブロック

図 23 “FUNCTION_CALL” ブロック (引数が 1 つの場合)

また，Mutators 機能により関数がとる引数の数をユーザが自由に変更することができる．引数はその引数が使われる関数のラベルとその引数のラベルにより管理される．引数が 1 つの場合の “FUNCTION” ブロックを図 22 に，“FUNCTION_CALL” ブロックを図 23，“ARGMENT” ブロックを図 24 に示す．

提案システムでは，引数として値が渡されるのではなく，ブロックが渡される．このようにすることで，異なる種類のコマンドに対応した複数個のブロックを引数として渡すことができる．関数ブロックの使用例を図 25 に，関数ブロックを使用せず図 25 と同じ内容となる対話シナリオを図 26 に示す．

5.3 対話シナリオ変換機能

UNIT ブロックの間に挟まれたブロックを上から順番に FST 形式の対話シナリオに変換する．システムは基本的に変換するブロックの状態番号と次状態番号のみを記憶する．ブロックを認識すると，そのブロックに対応した FST 形式の文を出力し，次のブロックの変換を行う．これを繰り返すことで対話シナリオを生成する．

今回作成したシステムでは，MMDAgent version 1.7[11] のサンプル FST ファイルに合わせるために，“UNIT” の開始状態番号は式 2 のように，終了状態番号は式 5 のように定義する．また，“UNIT” の最初のブロックの次状態番号は，式 3 のように “UNIT” の番号から算出する．

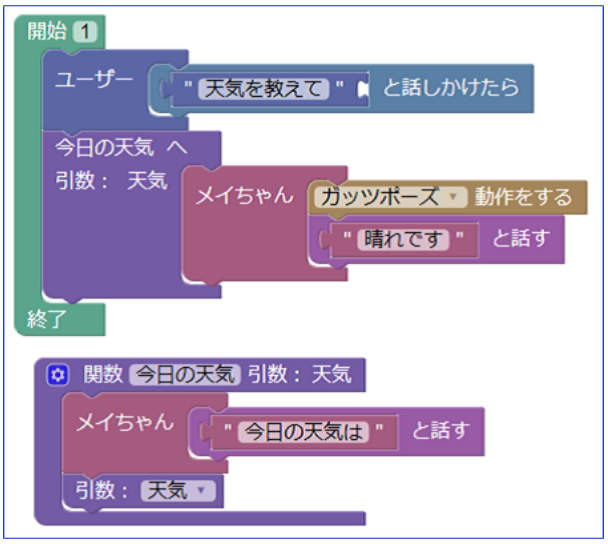

図 25 関数ブロック使用例

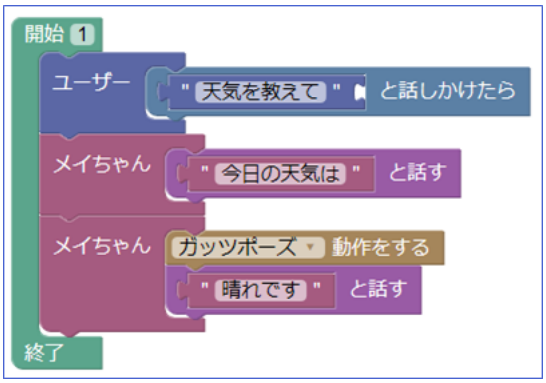

図 26 図 25 と同じ内容となる対話シナリオ

$$\text{状態番号} = 1 \tag{2}$$

$$\text{次状態番号} = \text{“UNIT” の番号} + 1 \tag{3}$$

“UNIT” の最後では，式 4 のように状態番号を付与し，無条件・無出力遷移する．

$$\text{状態番号} = \text{前のブロックの次状態番号} \tag{4}$$

$$\text{次状態番号} = 2 \tag{5}$$

次のように，ブロックの種類によって変換法が異なる．

5.3.1 命令ブロック

命令ブロックでは，そのブロックに対応した FST 形式の文を出力した後，式 6，7 のように状態番号を変更する．命令ブロックの使用例を図 27 に，図 27 から生成される FST 形式の対話シナリオを図 28 に示す．

$$\text{次のブロックの状態番号} = \text{次状態番号} \tag{6}$$

$$\text{次のブロックの次状態番号} = \text{次のブロックの状態番号} + 1 \tag{7}$$

5.3.2 分岐ブロック

分岐ブロックでは，分岐ブロックの前のブロックから分

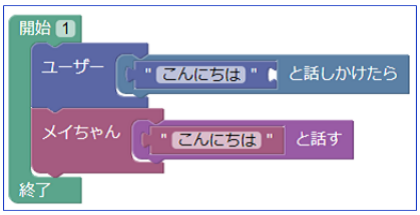

図 27　命令ブロックの使用例

```
1    101 RECOG_EVENT_STOP|こんにちは <eps>
101 102 <eps>                        SYNTH_START|mei|mei_voice_normal|こんにちは
102 2    <eps>                       <eps>
```

図 28　図 27 から生成される FST 形式の対話シナリオ

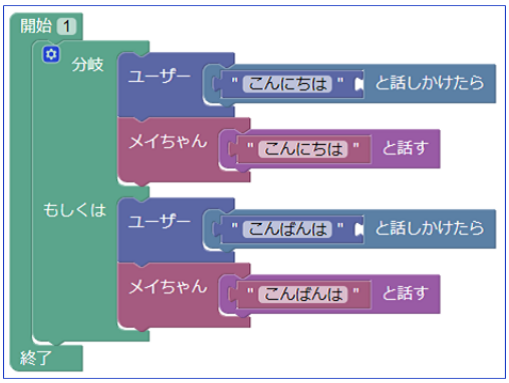

図 29　分岐ブロックの使用例

```
1    101 RECOG_EVENT_STOP|こんにちは <eps>
101 102 <eps>                        SYNTH_START|mei|mei_voice_normal|こんにちは
1    103 RECOG_EVENT_STOP|こんばんは <eps>
103 104 <eps>                        SYNTH_START|mei|mei_voice_normal|こんばんは
102 105 <eps> <eps>
104 105 <eps> <eps>
105 2    <eps> <eps>
```

図 30　図 29 から生成される FST 形式の対話シナリオ

岐先の全ての最初のブロックへ状態番号を繋げる．また，各分岐の最後のブロックから分岐ブロックの次のブロックへ無条件・無出力遷移を行う．分岐ブロックの使用例を図 29 に，図 29 から生成される FST 形式の対話シナリオを図 30 に示す．この例で分岐ブロックから生成される FST 形式の文は図 30 の 1 行目から 6 行目である．変換の具体的な手順を次に示す．

(1) 分岐ブロックの状態番号を記憶する．
(2) 分岐先の最初のブロックの状態番号を（1）で記憶した状態番号とする．
(3) 分岐先のブロックの変換を行う．
(4) 分岐先の最後のブロック次状態番号を記憶する．
(5) （2）から（4）を繰り返す．
(6) 全ての分岐が終了したら，(4) で記憶した全ての次状態番号を状態番号として，次状態番号へ無条件・無出力遷移する．

5.3.3　GOTO ブロック

GOTO ブロックでは，“GOTO”ブロックから“GOTO_STAR”ブロックへ無条件・無出力遷移を行う．

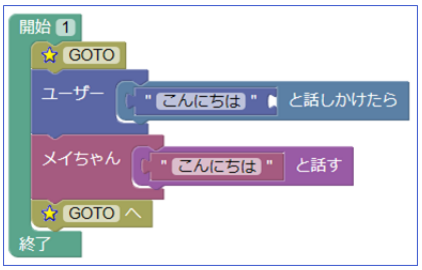

図 31　GOTO ブロックの使用例

```
1    101 RECOG_EVENT_STOP|こんにちは <eps>
101 102 <eps>                        SYNTH_START|mei|mei_voice_normal|こんにちは
102 1    <eps>                       <eps>
102 2    <eps>                       <eps>
```

図 32　図 31 から生成される FST 形式の対話シナリオ

図 33　関数ブロックの使用例

GOTO ブロックの使用例を図 31 に，図 31 から生成される FST 形式の対話シナリオを図 32 に示す．この例で GOTO ブロックから生成される FST 形式の文は図 32 の 3 行目である．変換の具体的な手順を次に示す．

(1) “GOTO_STAR”ブロックの状態番号を記憶する．
(2) “GOTO”ブロックにきたら，(1) で記憶した状態番号を次状態番号として無条件・無出力遷移する．

5.3.4　関数ブロック

関数ブロックでは，“FUNCTION_CALL”ブロックから“FUNCTION”ブロックへ状態番号を繋げる．また，“FUNCTION”ブロックに挟まれた最後のブロックから“FUNCTION_CALL”ブロックの次のブロックへ状態番号を繋げる．関数ブロックの使用例を図 33 に，図 33 から生成される FST 形式の対話シナリオを図 34 に示す．この例で関数ブロックから生成される FST 形式の文は図 34 の 4 行目から 6 行目である．変換の具体的な手順を次に示す．

(1) “FUNCTION_CALL”ブロックの状態番号を記憶する．
(2) “FUNCTION”ブロックに挟まれた最初のブロックの状態番号を（1）で記憶した状態番号とする．
(3) “FUNCTION”ブロックに挟まれたブロックの変換を行う．

```
1    101 RECOG_EVENT_STOP|こんにちは <eps>
104 2    <eps>                          <eps>

101 102 <eps>                          SYNTH_START|mei|mei_voice_normal|こんにちは
102 103 RECOG_EVENT_STOP|さようなら <eps>
103 104 <eps>                          SYNTH_START|mei|mei_voice_normal|さようなら
```

図 34　図 33 から生成される FST 形式の対話シナリオ

(4)“FUNCTION”ブロックに挟まれた最後のブロックの次状態番号を記憶する．
(5)“FUNCTION_CALL”ブロックの次のブロックの状態番号を（4）で記憶した次状態番号とする．
(6)“FUNCTION_CALL”ブロックの次のブロックの次状態番号を次のブロックの状態番号＋ 1 とする．

また，関数に引数がある場合には，“ARGMENT”ブロックからその引数の最初のブロックへ状態番号を繋げる．また，引数の最後のブロックから“ARGMENT”ブロックへ状態番号を繋げる．変換の具体的な手順を次に示す．

(1)“ARGMENT”ブロックの状態番号を記憶する．
(2)引数の最初のブロックの状態番号を（1）で記憶した状態番号とする．
(3)引数のブロックの変換を行う．
(4)引数の最後のブロックの次状態番号を記憶する．
(5)“ARGMENT”ブロックの次のブロックの状態番号を（4）で記憶した次状態番号とする．
(6)“ARGMENT”ブロックの次のブロックの次状態番号を次のブロックの状態番号＋ 1 とする．

6.　プロトタイプシステム

この章では，提案システムに基づいて実装したプロトタイプシステムの概要と使用方法について述べる．

6.1　プロトタイプシステムの概要

システムの開発は，Windows10 上で Eclipse4.7 の環境のもと行い，プログラミング言語として JavaScript を使用した．プロトタイプシステムは Web ブラウザ上で利用可能で，Google Chrome 74.0.3729.131 と Microsoft Edge 44.17763.1.0 で動作を確認した．また，作成された対話シナリオは，MMDAgent version 1.7 で動作を確認した．

6.2　プロトタイプシステムの使用方法

プロトタイプシステムを図 35 に示す．プロトタイプシステムは，次の 3 つのエリアに分かれている．

ツールボックス

システムで使用できるブロックの一覧を表示するエリアである．ブロックのカテゴリ名だけが表示され，カテゴリ名をクリックすると，そのカテゴリに属したブロックの一覧が表示される．

ワークスペース

ブロックを組み合わせて，対話シナリオを作成するエリアである．ツールボックスからブロックを選択し，組み合わせることで対話シナリオを作成する．

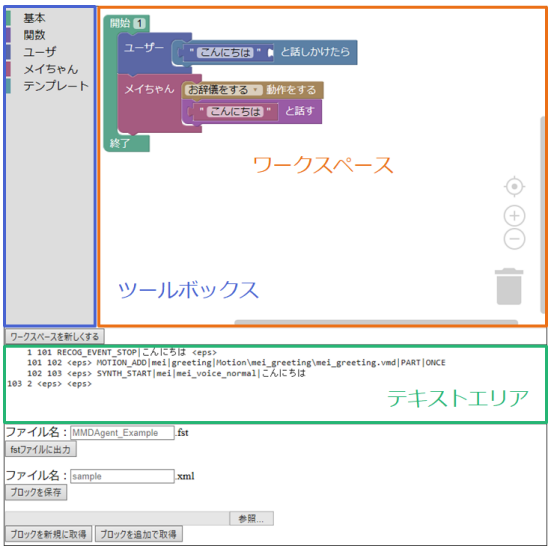

図 35　プロトタイプシステム

テキストエリア

FST 形式の対話シナリオを表示するエリアである．ワークスペースで組み合わせたブロックから生成される FST 形式の対話シナリオが出力される．

7.　評価実験

この章では，第 5 章で述べたプロトタイプシステムを使用して行った評価実験について述べる．

7.1　評価実験の目的

実験の目的は，提案システムを使用することにより対話シナリオの編集が簡単になったか，生成された対話シナリオがみやすくなったかを検証することである．また，FST 形式の対話シナリオの編集と比べて，記述ミスが減ったかを確認することである．

7.2　評価実験の方法

7.2.1　記述問題

被験者（本学学生 6 人）に，指定した内容の対話を再現する対話シナリオを作成する記述問題 3 問を 2 つの方法を用いて解いてもらった．各問題を解くのにかかった時間と正しい対話シナリオを作成した人数，実験後のアンケートから評価を行った．

用いた方法は以下の 2 つである．用いる方法の順番に影響されないように，被験者を 2 つのグループに分け，1 つのグループには方法（1），方法（2）の順で解いてもらい，もう 1 つのグループには方法（2），方法（1）の順で解いてもらった．

(1) FST 形式で対話シナリオを作成する．
(2) 提案システムを使用して対話シナリオを作成する．

表 1　対話シナリオの編集に対する評価項目

番号	質問内容
1	簡単である
2	対話シナリオが見やすい
3	対話シナリオが編集しやすい
4	取っ付きやすい
5	記述ミスの発見が容易

表 2　対話シナリオの読解に対する評価項目

番号	質問内容
1	簡単である
2	対話シナリオが見やすい
3	対話シナリオの内容を推測しやすい
4	取っ付きやすい
5	時間がかからない

表 3　記述問題の実験結果 (FST 形式)

	FST 形式	
問題	かかった時間	正解した人数
1	8 分 50 秒	3 人/6 人
2	4 分 39 秒	2 人/6 人
3	10 分 03 秒	2 人/6 人
正解者の平均時間	8 分 35 秒	

そして，実験後に方法（1）と方法（2）のそれぞれの場合についてアンケートに答えてもらった．アンケート項目として 5 項目を用意し，5 段階評価を行った．評価基準は，「1(当てはまらない)，2(やや当てはまらない)，3(どちらとも言えない)，4(やや当てはまる)，5(当てはまる)」である．アンケート項目を表 1 を示す．

7.2.2　読解問題

被験者（本学学生 6 人）に，2 つの方法を用いて記述された対話シナリオから対話の内容を推測する読解問題 2 問を解いてもらった．実験後のアンケートから評価を行った．

用いた方法は以下の 2 つである．記述問題と同様に，2 つのグループに分けて行った．

(1) FST 形式で記述された対話シナリオ

(2) 提案システムを用いて記述された対話シナリオ

そして，実験後に方法（1）と方法（2）のそれぞれの場合についてアンケートに答えてもらった．アンケート項目として 5 項目を用意し，5 段階評価を行った．評価基準は，記述問題と同様である．アンケート項目を表 2 を示す．

7.3　評価実験の結果

7.3.1　記述問題

各問題の平均解答時間と正しい対話シナリオを作成した人数，正しい対話シナリオを作成した人の全問題の平均解答時間を表 3 と表 4 に示す．また，アンケートの回答結果を平均して棒グラフに表したものを図 36 に示す．

7.3.2　読解問題

アンケートの回答結果を平均して棒グラフに表したものを図 37 に示す．

表 4　記述問題の実験結果 (提案システム)

	提案システム	
問題	かかった時間	正解した人数
1	3 分 27 秒	6 人/6 人
2	4 分 26 秒	3 人/6 人
3	6 分 01 秒	2 人/6 人
正解者の平均時間	3 分 58 秒	

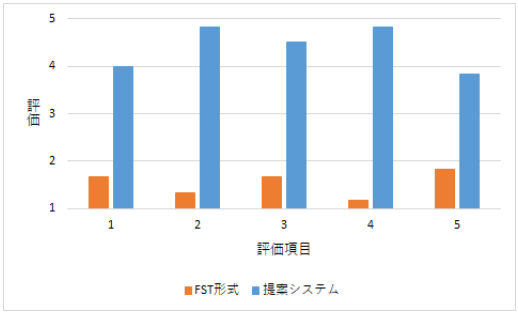

図 36　対話シナリオの記述に対するアンケート結果

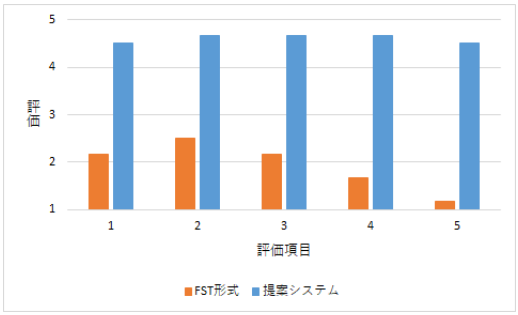

図 37　対話シナリオの読解に対するアンケート結果

7.4　評価実験の考察

各問題の平均解答時間を比較すると，全ての問題で提案システムを使用した方法の方がかかった時間が短くなっている．正しい対話シナリオを作成した人の全問題の平均解答時間を見ても，FST 形式の対話シナリオを作成する方法では 8 分 35 秒かかったが，提案システムを使用して対話シナリオを作成する方法では 3 分 58 秒で 4 分 37 秒の短縮となった．また，正しい対話シナリオを作成した人数を比較すると，全ての問題で提案システムを使用した方法の方が人数が多くなっている．これらのことから，提案システムを使用した方が，短い時間で正しい対話シナリオを作成できることが分かる．

また，被験者により作成された間違った対話シナリオを見てみると，FST 形式の対話シナリオを作成する方法では FST 形式のコマンドの誤字や状態番号が正しく付与されていない，FST 形式の記述のルールに反しているというものが多かった．一方，提案システムを使用した方法ではそのような間違いはなく，ブロックの使い方を間違えているものがあった．これらのことから，提案システムを使用する

ことで編集時のミスを減らすことができ，FST 形式の構文を学習し，覚えていなくても簡単に対話シナリオを作成できることが分かる．また，ブロックの使い方を間違えていたことに対しては，ブロックの仕様の改良が必要だと考える．

対話シナリオの記述に対するアンケートの結果を見ると，どの評価項目についても提案システムを使用した方法の方が良い評価を得ていることが分かる．提案システムを使用することで，FST 形式の対話シナリオの見にくい，編集しにくい，取っ付きにくいという問題が解決すると考えられる．

対話シナリオの読解に対するアンケートの結果を見ると，どの評価項目についても提案システムを使用した方法の方が良い評価を得ていることが分かる．また，読解問題の解答を見ると，提案システムを使用した方法の方が対話シナリオを具体的に推測しているものが多かった．このことから，提案システムを使用することで対話シナリオの細かい点まで注目でき，詳しく対話の内容を把握できることが分かる．

8. おわりに

本論文では，Google Blockly の技術を利用した，音声対話シナリオ編集システムを提案し，その実現法について述べた．また，提案システムに基づいたプロトタイプシステムを実装し，プロトタイプシステムを用いて評価実験を行った．

評価実験では，FST 形式での対話シナリオの記述と提案システムを用いた対話シナリオの記述を比較した．その結果，対話シナリオを作成する実験では提案システムを使用することで，FST 形式の記述と比べて作成時間が 4 分 37 秒の短縮となり，5 段階評価のアンケートで「対話シナリオが見やすい」，「取っ付きやすい」という項目において 4.83 と高評価が得られた．また，対話シナリオの意味を推測する実験では，5 段階評価のアンケートで「対話シナリオが見やすい」，「対話シナリオの内容を推測しやすい」，「取っ付きやすい」という項目において 4.67 と高評価が得られた．これらのことから，提案システムを使用することで短い時間で簡単に，見やすく分かりやすい対話シナリオを作成できることが分かった．

今後の課題として，提案システムによって生成された FST 形式の対話シナリオの可読性を挙げることが挙げられる．現在，提案システムによって生成される FST 形式の対話シナリオは各行の開始位置がずれており，見にくくなっている．また，無条件・無出力遷移があるため，生成される FST の行が多くなっている．これらの問題に対して，簡素化した FST をユーザに出力できるように改善が必要であると考える．また，ブロックの使用方法の誤りを防ぐために，ブロックの形などの仕様の改良も必要であると考える．

参考文献

[1] A.Lee, K.Oura, K.Tokuda, MMDAgent - A fully open-source toolkit for voice interaction systems, Proceedings of the ICASSP 2013, pp. 8382-8385, 2013.5.

[2] Google Blockly，入手先 ⟨https://developers.google.com/blockly/⟩（参照 2019-05-13）.

[3] Scratch，入手先 ⟨https://scratch.mit.edu/⟩（参照 2019-05-13）.

[4] viscuit，入手先 ⟨http://www.viscuit.com/⟩（参照 2019-05-13）.

[5] Keitaro Wakabayashi, Daisuke Yamamoto, Naohisa Takahashi, A Voice Dialog Editor Based on Finite State Transducer Using Composite State for Tablet Devices, Proc. of IEEE/ACS ICIS 2015, Las vegas, 2015.6.29.

[6] Keitaro Wakabayashi, Daisuke Yamamoto, Naohisa Takahashi, A Voice Dialog Editor Based on Finite State Transducer Using Composite State for Tablet Devices, Computer and Information Science 2015, Studies in Computational Intelligence, Vol. 614, pp.125-139, 2016.

[7] Ryota Nishimura, Daisuke Yamamoto, Takahiro Uchiya, Ichi Takumi, Development of a Dialogue Scenario Editor on a Web Browser for a Spoken Dialogue System, Proc. of the 2nd International Conference on Human Agent Interaction, ACM digital library, pp. 129-132, 2014.

[8] Ryota Nishimura, Daisuke Yamamoto, Takahiro Uchiya, Ichi Takumi, MMDAE: Dialog scenario editor for MMDAgent on the web browser, ICT Express, Volume 5, Issue 1, pp.47-51, 2019.

[9] Blockly Developer tools，入手先 ⟨https://blockly-demo.appspot.com/static/demos/blockfactory/index.html⟩（参照 2019-05-13）.

[10] Google Blockly Mutators，入手先 ⟨https://developers.google.com/blockly/guides/create-custom-blocks/web/mutators⟩（参照 2019-05-13）.

[11] MMDAgent version 1.7，入手先 ⟨http://www.mmdagent.jp/⟩（参照 2019-05-13）.