明治大学大学院　理工学研究科

2023 年度修士学位請求論文

遅延聴覚フィードバックがもたらす影響の客観的な評価方法の検討と年齢による影響の変化の分析

2024 年 2 月

指導教員　村上隆啓

専攻　電気工学専攻

研究室名　知能信号処理研究室

学位請求者　山下　一樹

目次

目次

第 1 章

序論

1.1 背景

本節では，現代における補聴器技術の主流であるディジタル補聴器に着目し，その進化と課題について述べる．ディジタル補聴器はディジタル信号処理技術を活用し，従来のアナログ補聴器に比べて高度な機能を実現していることが指摘されている [1][2]．具体的には，音声の増幅率を音圧レベルに応じて調整するノンリニア増幅機能や，雑音を低減しつつ目的の音声を強調するノイズリダクション機能などが挙げられる．しかしながら，補聴器の利用者からは，これらの機能にも関わらず十分な満足度が得られていないという問題が報告されている [3]．不満の一因として，ノンリニア増幅が不要な音まで増幅することに起因するという意見が存在する．このことは，ディジタル補聴器のさらなる性能改善への要求を示唆している．ディジタル補聴器の性能を向上させるためには，精緻なディジタル信号処理が必要であり，これには音声信号の周波数帯域を細分化することが求められる．しかしながら，周波数帯域の細分化は，処理に使用する音声信号の長さを増加させるという課題を伴う．ディジタル補聴器では，音声信号を数ミリ秒単位のフレームに分割して処理を行うため，入力から出力までにフレーム長に相当する時間の遅延が発生する．さらに，アナログ信号をディジタル信号に変換し，その後再びアナログに戻す AD/DA 変

換プロセスによっても遅延が生じる．この AD/DA 変換とフレーム長に起因する遅延は，音声の入出力間で最低数ミリ秒のタイムラグを発生させる．したがって，ディジタル信号処理に用いる音声信号の長さが長くなると，補聴器における音声の入出力信号間の遅延時間も増大することになる．人間は能動的な活動を行う際，活動とそれに伴う感覚フィードバックを対応付けることで行動の調整を行っている．この中で，聴覚に関するフィードバックを聴覚フィードバックと呼んでいる [4]．聴覚フィードバックは，発話や身体運動において重要な役割を果たしている．例として，発話時に自身の声を聴くことにより，音声の高さや強さを調整する行為が挙げられる．遅延聴覚フィードバックは，発話者が自身の声を聴く際のタイミングが遅延することによって生じ，ディジタル補聴器における入出力信号間の遅延がこれに該当する．一般に，この遅延時間が 10[ms] を超えると発話に違和感を覚えることが知られている [5]．そのため，ディジタル補聴器はこの遅延時間を超えないよう設計されている [1]．しかし，この設計制約がディジタル補聴器の性能向上における課題となっている．ディジタル信号処理の精度を高めるためには音声信号の長さを長くする必要があり，これは必然的に入出力間の遅延時間を増加させる．そのため，ディジタル信号処理における高度な処理と遅延時間の短縮という二つの要求を両立させることが困難である．一方で，高齢者においては，遅延時間が 10[ms] を超えても発話に違和感を覚えにくいことが示唆されている．この知見を活用すれば，ディジタル補聴器の入出力信号間の遅延時間を増大させ，それに伴い音声信号の長さを長くすることが可能となる．これにより，周波数帯域の細分化を進め，より高度なディジタル信号処理を実装することが期待される．このアプローチは，特に高齢者における補聴器の性能向上に寄与する可能性がある．

1.2　目的

本研究では，若年者と高齢者における聴覚フィードバックの遅延時間の許容範囲の差について検討する．文献 [6] において，著者らが行った調査のシステムについて改良を行い，

遅延聴覚フィードバックの身体運動への影響を調査する．ここでは，身体運動への影響を幅広い年代の被験者間で比較することを想定して，高齢者でも簡単に実験を行うことのできるボタン押し課題を採用する．本研究で行うボタン押し課題は，一定の時間間隔ごとにコントローラーのボタンを押すという動作を一定の回数行う課題である．被験者がボタンを押下する一定の時間間隔は，ボタン押し課題を行っている間，電子メトロノームの合図音によって提示する．そして，被験者は聴覚フィードバックに遅延が発生している状態で本研究のボタン押し課題を行う．遅延聴覚フィードバックが身体運動に何らかの影響を与えていれば，被験者がボタンを押下する時間間隔にばらつきが発生すると考えられる．そこで，被験者がボタンを押下する時間間隔の全体の分散，およびボタンの押下回数が 4 の倍数に到達する直前の押下間隔と 4 の倍数に到達した直後の押下間隔の二乗誤差を評価指標として用いる．また，遅延によるボタン押し課題への影響が大きく現れる最適な実験条件を探るために，ボタン押し課題における最適なボタンの押下間隔，ボタン押し課題の長さおよび最適な遅延のタイミングについて検討する．そして，決定した実験条件のもとで，聴力が正常な若年者と高齢者を対象に遅延聴覚フィードバックが身体運動に与える影響について調査を行う．本研究は，聴覚フィードバックの遅延が人の身体運動にどのような影響を与えるか，特に年齢差がその影響にどのようにして関わってくるかを明らかにすることに寄与することが期待される．この結果は，高齢者向けの補聴器の設計において，重要な示唆を提供するものであると考えられる．

第 2 章

先行研究

本章では，過去に行われた遅延聴覚フィードバックの影響を調査する研究について紹介する．

2.1　遅延聴覚フィードバックが発話に及ぼす影響の調査

先行研究 [7] では，主観調査による遅延聴覚フィードバックの影響の調査が行われた．主観調査とは，耳介付近に伝達された音に一定の遅延を発生させて外耳道に出力する装置（以後，音響測定装置と呼ぶ）を装着した被験者が，発話時の違和感を主観的に評価するという内容の調査である．人は，自身の発話音声を聴取するタイミングに遅延が生じると，違和感を覚えて発話が阻害されると考えられるため，このときの違和感を遅延時間ごとに主観的に評価することにより，音声における入出力信号間の遅延時間の許容範囲を検討することが可能になる．この調査では，被験者が原稿を読み上げているとき，発話音声が音響測定装置を通して耳に戻るまでの時間を変化させ，被験者が感じる違和感の程度を調査する．被験者は，「読み上げるときにしゃべりにくくないか」および「遅れが気にならないか」の 2 つについて「優（4 点）」「良（3 点）」「可（2 点）」「不可（1 点）」の 4 段階で評価する．評価基準は，表 2.1 および表 2.2 のとおりである．得られた評価結果は被験者

表 2.1: 読み上げるときにしゃべりにくくないかの評価基準

評価	評価基準	評点
優	しゃべりにくくない	4
良	しゃべりにくいが気にならない	3
可	しゃべりにくい	2
不可	とてもしゃべりにくい	1

表 2.2: 遅れが気にならないかの評価基準

評価	評価基準	評点
優	遅れがまったくわからない	4
良	遅れが分かるが気にならない	3
可	遅れが気になる	2
不可	遅れがはっきり分かる	1

間で平均され，評点は，しゃべりやすさと遅延の気にならなさに比例する．若年者と高齢者の間で評点を比較することで，発話音声に発生させる遅延時間の大きさと違和感の程度が，若年者と高齢者の間でどのように異なるかを観察することが可能となる． また，文献 [8] での調査結果によると，若年者と高齢者の間で聴覚フィードバックの遅延時間の許容量には差異があったものの，両者で提示した遅延時間に違いがあったため，その差が統計的に有意であるかの分析が困難であった．そのため，著者らは両者の遅延時間を揃えるために高齢者に提示した遅延時間と同一の条件で若年者に対して主観評価実験を行った．遅延時間ごとに若年者と高齢者の結果の分布をコルモゴロフ・スミノルフ検定で比較したところ，90[ms] 以上の遅延で若年者と高齢者の遅延の感じやすさに有意な差が存在することが示され，若年者は高齢者と比較して遅延の影響をより敏感に受けやすいという結果が得られた．今後は，主観調査実験について，評価方法を改良して調査を行い，主観評価による個人差が減少するかどうかを検証することが必要であるとされている．

2.2　遅延聴覚フィードバックが身体運動に及ぼす影響の調査

重松氏らによる研究 [6] では，遅延聴覚フィードバックの影響を客観的に評価するために，テンポの画面提示アプリケーション [9] を活用したボタン押し課題を通じて，遅延聴覚フィードバックが身体運動に及ぼす影響の調査が行われた．ボタン押し課題は，遅延聴覚フィードバックの下で，一定のテンポでボタンを押すことを被験者に要求する課題である．テンポの画面表示アプリケーションは，画面の上部に表示される短いバーが画面下部の長いバーへ向けて一定速度で移動し，両バーが重なるタイミングで被験者がボタンを押下するタイミングが示される仕組みである．また，長いバーの点滅もタイミングの指示に利用される．この課題では，指定された遅延時間を用いて若年者 8 名を対象に実験が実施された．結果として，この課題による遅延聴覚フィードバックが身体運動に与える影響の観察が可能であることが分かった．その一方で，被験者間で遅延時間への反応に差異が認められ，特定の被験者では遅延時間の増加がボタン押下間隔に及ぼす影響が限定的であることが確認された．これは，被験者が遅延時間に関わらず聴覚フィードバックとしてボタン押下時の音を認識せずに画面のみに従ってボタン押し課題を行ってしまった可能性を示唆している．したがって，被験者が一貫してヘッドホンからの音を聴覚フィードバックとして認識できるようにテンポの提示方法の改良が必要であるとされている．本研究は，このボタン押し課題のシステムを改良し，遅延聴覚フィードバックが身体運動に与える影響をより明確に観察することを目指している．

第 3 章

主観評価実験におけるアプリケーション開発

本章では，2.1 節で述べた主観評価実験で利用することを想定したアプリケーションの概要と機能について説明する．これまでの主観評価実験では、遅延聴覚フィードバック下での違和感を被験者に実験者が用意した用紙に入力させ，その後 PC 上にデータを移行して結果を保存していた．このデータ移行には入力ミスのリスクがあり，注意深い作業が必要で研究にとって非効率であった．このアプリケーションを作成することで，被験者がアプリケーション上に直接評価結果を入力し，自動で外部ファイルに結果を出力できるようになる．これにより，実験者の負担が軽減され，効率的な実験および評価結果の分析が可能となり，短期間で多くの実験を実施することが期待できる．また，本節で述べる関数は，文献 [10] に基づいて利用する．

3.1　アプリケーションの概要

本研究では, Microsoft 社が提供する統合開発環境である Microsoft Visual Studio 2022 を使用し，C++ で開発する．アプリケーションの外観を図 3.1 および図 3.2 に示す．図

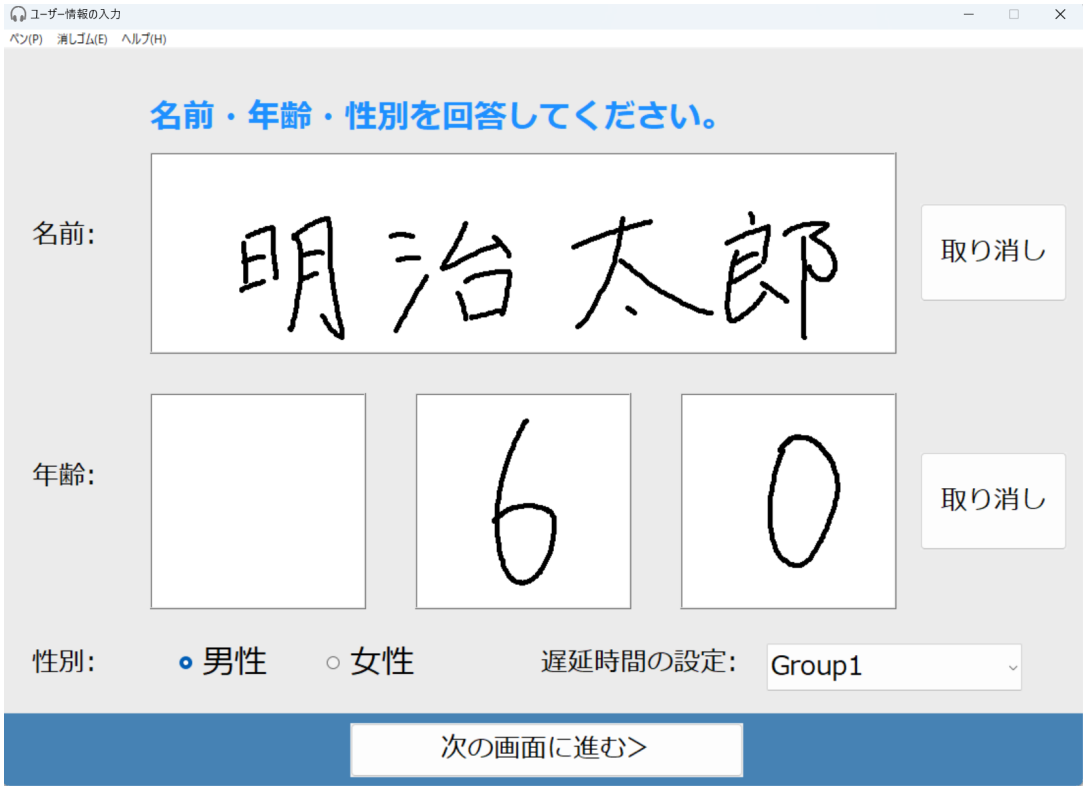

図 3.1: アプリケーション起動直後の画面

3.1 は，実験開始後に最初に表示されるアプリケーションの画面である．図 3.2 は，図 3.1 の画面で「次の画面に進む」ボタンを押下した後に表示される画面である．

また，開発したアプリケーションのプログラムを付録 B に掲載する．以下に，開発するアプリケーションの機能を示す．

(1) 高齢者が使用することを想定し，タッチパネルのように名前と年齢を描画できる機能（3.1.1 項参照）

(2) 被験者の名前と年齢，性別，遅延時間の設定が書かれている画面をキャプチャすると同時にそれらを外部ファイルに書き込む機能（3.1.2 項参照）

(3) 読む文章の番号の順番をランダムに定義し，画面上に表示する機能

(4) 2 つの質問に対するそれぞれ 4 つの回答項目をプッシュボタンとして表示し，押下された結果を (1) と同じ外部ファイルに書き込む機能（3.1.3 項参照）

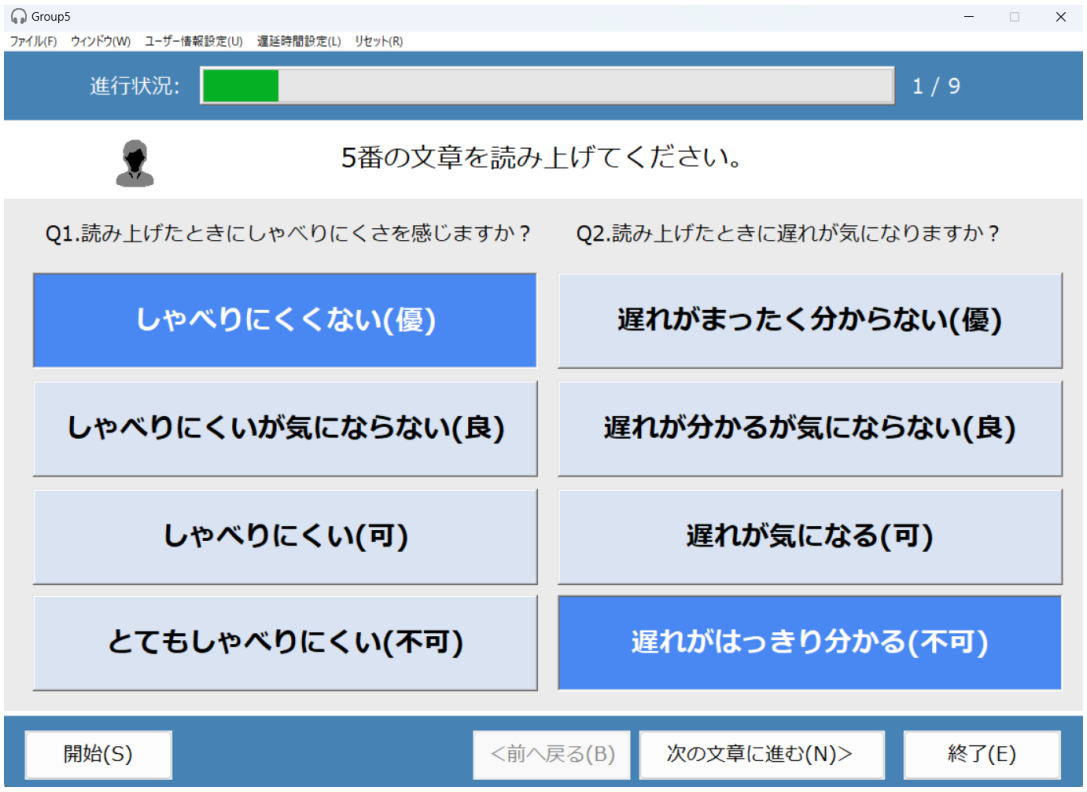

図 3.2: 調査中の画面

主観評価で被験者に読んでもらう文章の番号は，文献 [7] で使用された文章を使用することを想定し，10 通り用意する．そのため，(3) では，1 番から 10 番までの番号をランダムに並び替え，画面上に表示することで，被験者に提示する．

3.1.1　タッチパネルによるユーザー情報の取得

指やペンなどの 1 つ以上のタッチポイントがタッチに依存するデジタイザーサーフェスに触れたときに，Windows API の「WM_TOUCH」メッセージがウィンドウに通知される [10]．「WM_TOUCH」イベントは，タッチ入力に関する情報を含んでおり，アプリケーションはこのイベントを処理して，タッチ操作に応じたアクションを実行することができる．このアクションには，例えばタッチするスクリーン上の位置，タッチの圧力，動きなどの情報が含まれる．このイベントを取り扱うためにまず，ウィンドウ作成時に

RegisterTouchWindow() 関数を使用してアプリケーションがタッチイベントを受けとることができるようにする．その後，ウィンドウプロージャで「WM_TOUCH」メッセージ内の処理を行うことにより，タッチパネルとしての機能を実装する．「WM_TOUCH」イベントが通知されたら以下の処理を行うように設定する．

(1) ウィンドウのデバイスコンテキストのハンドルを取得し，デバイスコンテキストに新しいペンのハンドルを割り当てる．
(2) GetTouchInputInfo() 関数を使用して，各タッチイベントの情報を取得する．
(3) 取得したタッチイベントの情報を元に，タッチポイントの座標を画面上の座標に変換し，タッチの位置がタッチパネル内にあるかどうかを確認する．タッチが続いている場合，以前のタッチポイントから現在のタッチポイントまで線を描画する．
(4) 各タッチポイントについて，前回のタッチポイントの位置とタッチパネルの内か外かを記録する．これにより，タッチの移動を追跡し，描画を連続的に行うことができるようにする．
(5) 描画が終わった後，使用したペンを削除し，デバイスコンテキストを解放する．

3.1.2　画像の取り込みおよび保存

図 3.1 に示したユーザーが手書きで入力した名前と年齢は，画面下部の「次の画面に進む」というプッシュボタンをユーザーが押下したことを合図にウィンドウの画像をキャプチャし，画像ファイルとして保存することによって記録する．画像をキャプチャする方法は，以下の手順で実装する．

(1) GetDC() 関数を使用して，ウィンドウのデバイスコンテキストを取得する．
(2) GetClientRect() 関数を使用して，ウィンドウのクライアント領域の寸法を取得する．クライアント領域は，アプリケーションが描画できるウィンドウの部分であり，

タイトルバーと境界線を除いた部分である．

(3) クライアント領域の幅と高さ，および年齢の 100,10,1 桁を表す 3 つの領域の幅と高さを計算する．

(4) CreateCompatibleDC() 関数と CreateCompatibleBitmap() 関数を使用して，ウィンドウ全体と 3 つの年齢を示す領域のメモリデバイスコンテキストおよびビットマップを作成する．

(5) BitBlt() 関数を使用して，ウィンドウ全体と 3 つの年齢を示す領域のビットマップをメモリデバイスコンテキストにコピーする．

(6) c++ の CImage クラスを使用して，ビットマップを読み込み．画像を指定したフォルダに JPEG として保存される．そのフォルダが存在しない場合，新しく画像を保存するためのフォルダが作成される．

(7) 最後に CImage オブジェクトからビットマップを切り離し，ウィンドウのデバイスコンテキストを解放し，元のグラフィックオブジェクトをメモリデバイスコンテキストに再選択してから，ビットマップとメモリデバイスコンテキストを削除する．

3.1.3　回答の入力と出力

2.1.1 項において記述した主観評価実験では，「文章の読み上げ時のしゃべりにくさ」と「文章の読み上げ時の遅れの感じ方」に関する 2 つの質問が提示される．被験者は，これらの質問に対して，4 つの選択肢の中から Windows API により実装したオーナー描画ボタン [10] を通じて回答する．ボタンが選択されると，背景色は青色に，文字色が白色になる設計となっており，これにより高齢者を含む操作に不慣れなユーザーでも，選択状態を直感的に把握することが可能である．また，背景色と文字色の変更は，ボタン押下時に InvalidRect() 関数によって明示的にウィンドウの再描画を要求することによって実現している．ウィンドウの再描画が必要な場合，「WM_ERASEBKGND」メッセージ

が受信され，ウィンドウの背景がクリアされた後，「WM_PAINT」メッセージによってウィンドウの内容が再描画される．この 2 つの処理ステップが画面のちらつきを引き起こすことがあるため，「WM_ERASEBKGND」メッセージの処理を明示的にスキップし，InvalidRect() で更新する領域を 8 つのボタンを含む領域の中で最小限に設定する．そうすることで，背景の再描画処理を行わずに直接「WM_PAINT」メッセージでの再描画に移行する．これにより，背景と前景の描画が一度に行われるため，ちらつきを減少させることができ，特に高齢者にとって快適な操作体験が実現できる．

また，全てのボタンの選択状況を常に監視し，両方のボタンが選択されていない場合，「次の文章に進む」ボタンを無効化する．これにより，結果の記録における誤りを防止する．さらに，調査の進行状況を示すプログレスバーを画面上部に設置し，調査の進捗状況をユーザーに視覚的にフィードバックする．「次の文章に進む」ボタン押下時には音声信号が本プリケーションを起動しているデバイスから出力され，実験者はこの音声信号による合図を受けて，ユーザーの合図を待たずに次の調査へと移行できる．アンケート終了後，ユーザーが「次の文章に進む」ボタンを押下すると，アプリケーション起動時に選択した CSV ファイルに，キャプチャした画面が保存されているファイルのパス，遅延時間の提示順，読まれた文章の番号および被験者の回答結果が自動的に記録される．既存のファイルであれば，結果はファイルの末尾に追記される．このシステムにより，多数の被験者を対象とする実験でも，アプリケーションの再起動なしに迅速に実験を進行できるという利点がある．

第 4 章

ボタン押し課題のシステム

本章では，遅延聴覚フィードバックが身体運動に与える影響を客観的に評価するための調査で行うボタン押し課題，この調査を行うために構築した調査システム及び Windows アプリケーションについて述べる．聴覚フィードバックは遅延時間が大きくなると，発話だけでなく身体運動に影響を与えることが知られている [11][12]．身体運動を遅延聴覚フィードバックが与える影響の調査に使用することが可能となれば，客観的なデータを計測しやすくなることが期待される．

4.1　ボタン押し課題

本研究で行う客観評価による調査では，被験者が行う課題にボタン押し課題を採用する．この調査で採用するボタン押し課題は，ボタンを押下するときの音に遅延を発生させて被験者に聞かせながら，被験者がメトロノームの合図音に合わせて一定の時間間隔でボタンを押下する課題を行うというものである．このボタン押し課題は，楽器演奏のような特別な技能を必要としないため，遅延聴覚フィードバックが身体運動に与える影響を様々な年代の被験者について調査することが可能になると考えられる．ボタン押し課題を行っているとき，被験者がボタンを押下する時間間隔を記録すると，被験者に提示する

ボタン押下の時間間隔が毎分 60 回であれば，理想的に全てのボタンを押下する時間間隔が 1000[ms] となるが，人間の操作には誤差が生じる．また，遅延聴覚フィードバックが身体運動に影響を与えていれば，このばらつきを提示するボタンを押下する音の遅延時間によって変化するものであると考えられる．そのため，遅延聴覚フィードバックを与えている状態で被験者がボタン押し課題を行うと，ボタンを押下する時間間隔に変化が現れることが期待される．したがって，遅延聴覚フィードバックの下で被験者がボタン押し課題を行うときのボタンを押下する時間間隔を，様々な遅延時間で観察することで遅延聴覚フィードバックが身体運動に与える影響を客観的に評価することができると考えられる．この調査で使用するシステムの図を図 4.1 に示し，使用機器を表 4.1 に示す．このシステムは表 4.1 の使用機器と遅延聴覚フィードバックを生成する音響信号への遅延生成アプリケーション，メトロノーム，研究室で作成したパルスジェネレーターで構成されている．このパルスジェネレーターは，スーパーファミコンのボタンが押下されると，オペアンプを通じて音響信号がアナログ出力され，ディジタル入力された後，オーディオインターフェース上で信号の入力を検知したら，クリック音がヘッドホンから出力されるという仕組みである．音響信号への遅延生成アプリケーションは，4.2 節で述べられるものと同様である．以下に，本研究で用いるボタン押し課題の手順と図 4.1 のシステムの動作について説明する．

(1) 被験者はヘッドホンを装着し，コントローラーを手に持つ．コントローラーの A, B, C, D いずれかのボタンを押すと，任意の遅延時間が経過した後にヘッドホンに「ピッ」というクリック音が出力される．

(2) 電子メトロノームの合図音によって，一定間隔でボタンを押下するための合図を提示し，その合図に合わせて，被験者はボタン押し課題を実施する．ボタン押し課題で被験者がボタンを押下する回数は，アプリケーション上で実験者が設定する．課題中，PC に保存された WAV ファイルがオーディオインターフェースを介して，実験者が指定した時間だけ遅延してヘッドホンから出力される．この時，実験者が指定した遅

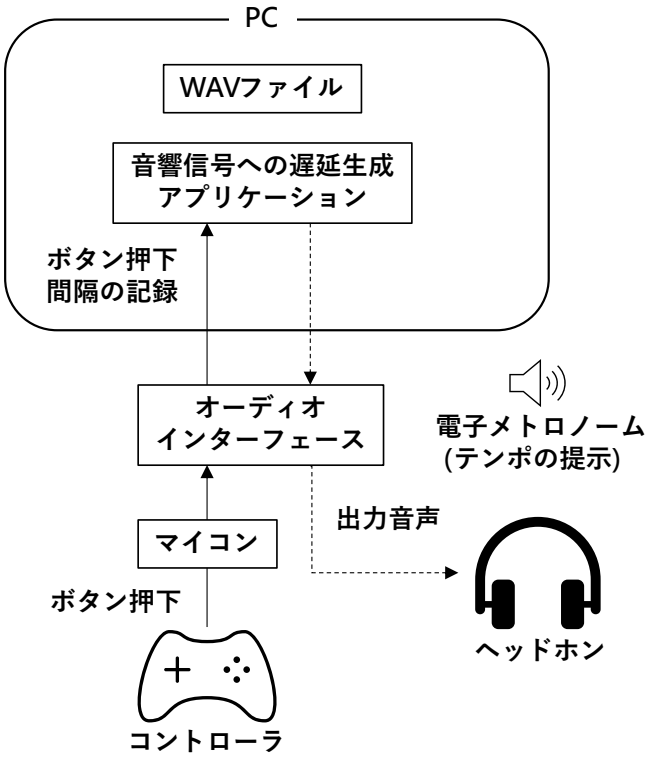

図 4.1: 調査システムの構成

表 4.1: 使用機器

使用機器	製造会社	製品名
オーディオインターフェース	Focusrite	Scalett-Solo 3rd Generation
コントローラー	Nintendo	Super Famicom Controller
開放型ヘッドホン	beyerdynamic	DT 990 PRO
電子メトロノーム	SEIKO	DM71 Digital Metronome
PC	HP Inc.	5

延時間は被験者には非公開とする．また，音響信号への遅延生成アプリケーションは，被験者がボタンを押下する時間間隔の記録も行う．

手順 1 の後，被験者が装着しているヘッドホンから出力される音声の音量調節を行う．このとき，大きな遅延時間を提示すると調査で提示する全ての遅延時間について遅れてい

ないと判断してしまう可能性が発話に関する調査で指摘されている [13]．そのため，音量調節時に音響信号への遅延生成アプリケーションで指定する遅延時間は，一般的に遅延を感じないとされている 10[ms] 以下のものとする．また，音量調節時にヘッドホンから音声を出力している間，被験者には遅延のない音声を出力していると説明する．音量調節が完了したら，練習としてメトロノームの合図音に合わせて 20 回から 40 回ボタンを押下してもらい，実験の内容を理解させる．上記の手順によって記録されるボタンを押下する時間間隔及び，5 章で説明する評価指標を用いて評価を行う．

4.2　音響信号への遅延生成アプリケーション

本研究で使用する音響信号への遅延生成アプリケーションは，Microsoft 社が提供する統合開発環境である Visual Studio 2022 を用いて C++ で開発する．このアプリケーションの表示例を図 4.2 に示す．図 4.2 は，ボタン押し課題を開始し，アプリケーションのスタート直後の状態である．被験者がボタンを押すと，押した時刻と直前に押した時刻からの経過時間 [ms] が画面左側のエディットボックス内に書き込まれる．実験が終了し，実験者が「ファイルへの出力」というプッシュボタンをクリックすると，実験者が指定した CSV ファイルに結果が書き込まれる．そして，開発したアプリケーションのソースコードを付録 B に掲載する．以下にアプリケーションの主な機能を示す．

(1) 任意の外部ファイルから複数の遅延時間を設定する機能
(2) 実験者が画面上のコンボボックスで指定した時間だけ遅延させる機能（4.2.2 項参照）
(3) 被験者がボタンを押下する時間間隔を記録する機能（4.2.3 項参照）
(4) 被験者が押下するボタンの押下回数が実験者がアプリケーション上で指定した回数に到達したら合図音の出力を一時的に停止する機能
(5) (3) で述べた記録とアプリケーション上での設定内容，被験者情報を外部ファイルに書き込む機能

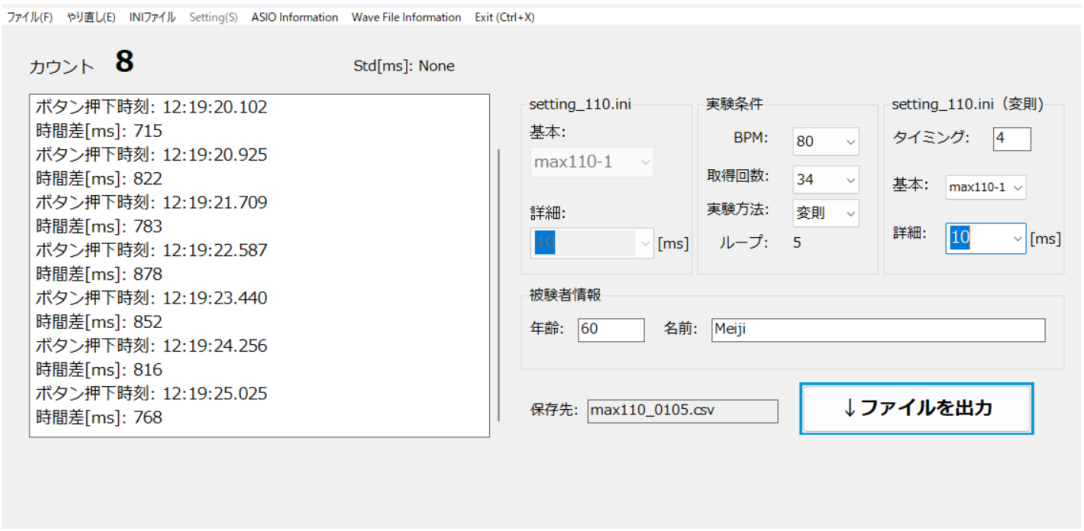

図 4.2: 実験開始直後の音響信号への遅延生成アプリケーションの画面

(1) は Windows で主に使用される INI（Initialization）ファイル形式を採用している．遅延時間の一覧を INI ファイルに予め設定し，アプリケーションから任意の INI ファイルを選択し読み込むことで，コンボボックスから遅延時間を選択する機能を実現している．また，コンボボックスで選択された遅延時間をエディットボックスで指定したタイミングで発生させることも可能である．さらに，選択された遅延時間に基づき遅延を発生させ，ボタンの押下回数が指定した回数に達し結果がファイルに書き込まれると，コンボボックスの選択項目は自動的に次に移行する．この機能により，ボタン押し課題は INI ファイルの選択と結果を出力するプッシュボタンの押下のみで実施可能である．

4.2.1　ASIO における音声の入出力

本研究で開発するアプリケーションは，オーディオドライバに ASIO を用いている．そこで，ASIO における音声の入出力方法について説明する．ASIO ではマルチバッファリングの切り替えを独自のコールバック関数で行う．独自のコールバック関数を用いることで，バッファの切り替えはオーディオインターフェースによって行われるため，OS の影響を受けないという利点がある．音声の入出力のシステムの動きを図 4.3 に示す．入力で

2 つのバッファ，出力で 2 つのバッファを利用しそれぞれでダブルバッファリングを行う．最初の入力バッファを入力バッファ 1，次の入力バッファを入力バッファ 2 とし，最初の出力バッファを出力バッファ 1，次の出力バッファを出力バッファ 2 とする．それぞれ，音声の同時入出力が行われる前に 0 に初期化しておく．入力バッファ 1 に入力信号の格納が開始した時点から音声が出力される時点までの仕組みを以下に示す．

(1) はじめに入力バッファ 1 に入力信号が格納され，それと同時に出力バッファ 1 に格納されたデータの再生が始まる．しかし，この時点で出力バッファ 1 には録音データが格納されていないため，無音になる．
(2) 入力バッファ 1 の格納可能な最大の許容量に達したとき，ASIO でコールバック関数が呼び出され，入力バッファ 1 と出力バッファ 1 がアプリケーションに受け渡される．それと同時に，入力バッファ 2 への録音データの格納が始まり，出力バッファ 2 の再生が始まる．ここでも，はじめは出力バッファ 2 にはまだ録音データが格納されていないため，無音となる．これと同時に，アプリケーション側ではコールバックにより入力バッファ 1 のデータを出力バッファ 1 にコピーする．
(3) 入力バッファ 2 が格納可能な最大の許容量に達したとき，再びコールバック関数が呼び出され，出力バッファ 1 に格納された録音データの再生と入力バッファ 1 への入力信号の格納が開始する．
(4) 手順 2 に戻る．

以上を繰り返すことにより，音声の入出力を可能としている．しかし，このダブルバッファリングを用いた方法では，2 バッファ分の遅延時間が常に発生する．

4.2.2　任意の遅延時間後にボタン押下の合図音を再生させる機能

任意の遅延時間が経過した後に WAV データを再生させる機能は，ASIO における音声の同時入出力の方法に基づいて実装する．ボタン押下の合図音の再生は，ボタン押下を検

知した後に呼び出されるコールバック関数内で，保存されている WAV データを入力バッファの代わりに出力バッファに転送することで行われる．この WAV データの出力バッファへの転送のタイミングを遅延時間ごとに調整することで，任意の遅延時間後にボタン押下の合図音を再生させる機能を実現させる．また，図 4.3 のようにボタンの押下検知から WAV データの再生までに少なくとも 2 つのバッファ分の遅延が生成されることになる．バッファサイズが小さいほど，より高精度な遅延時間の設定が可能となる．オーディオインターフェースのバッファサイズを n[points]，サンプリング周波数を f_s[Hz]，ASIO におけるインターフェイスに固有の入力遅延を i[ms]，ASIO におけるインターフェイスの固有の出力遅延を o[ms]，所望の遅延時間を d[ms] とすると，遅延時間の生成は以下のような手順で実現される．

(1) 初めに式 4.1 に基づき，WAV データのコピー時刻 T を定義する．この時刻 T は，ボタン押下の検出後，何回目のコールバック関数の呼び出し時に WAV データをコピーするかを示す指標である．生成したい遅延時間 d[ms] は，実験者がアプリケーション上で指定する．

$$T = \{d - (i + o)\} \times n \times \left(\frac{f_s}{1000}\right) \tag{4.1}$$

(2) ボタンの押下を検知してから最初のコールバック関数呼び出し時を 1 回目として，$T - 1$ 回目までは，入力バッファに格納されているデータを出力バッファにコピーする．このとき，入力バッファには 0 が格納されているため無音となる．

(3) ボタンの押下を検知してから T 回目のコールバック関数呼び出し時になったら，入力バッファに格納されているデータの代わりに WAV データを出力バッファにコピーしていく．

上記の手順を踏むことにより，任意の遅延時間の生成を実現する．

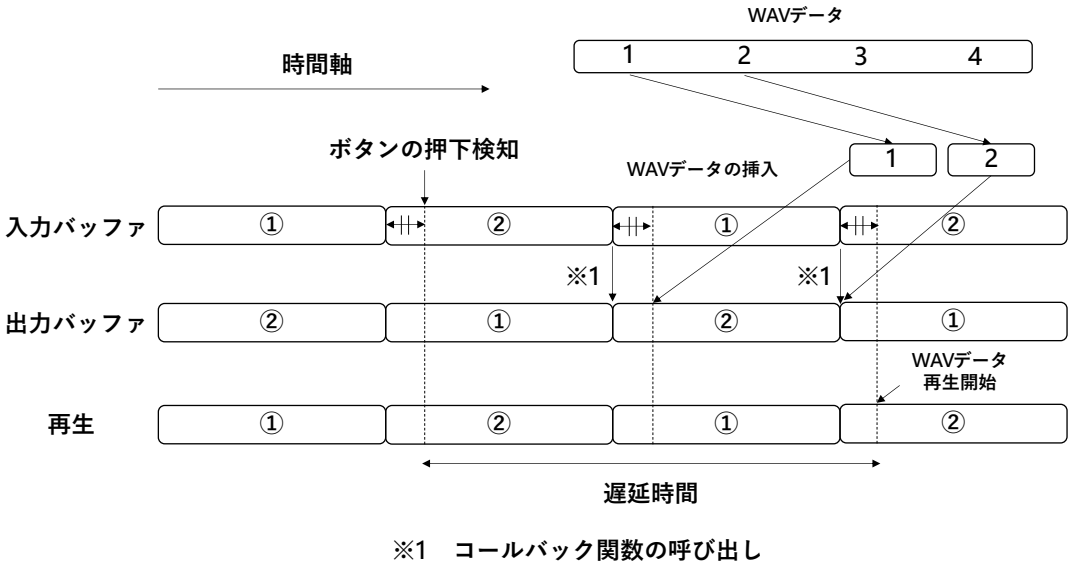

図 4.3: 遅延時間の生成原理

4.2.3　ボタンの押下時間間隔を記録する機能

ボタンの押下時間間隔を記録する機能は，ボタンを押下するごとに，ボタンの押下と押下の間の時間を計測する．例外として，1 回目のボタンの押下時間間隔は記録しないように設定する．押下時間間隔の取得には，C++ の標準ライブラリである std::chrono を使用する．std::chrono は，C++11 以降で使用可能な時間に関する操作を提供するライブラリである．以下に押下と押下の間の時間を計測するための手順を示す．

(1) ボタン押下を検知したら，関数 std::chrono::system_clock::now() を使用してエポック（1970 年 1 月 1 日 0 時 0 分 0 秒 UTC）からの経過時間を取得し，用意した変数 A に代入する．

(2) 再びボタンの押下を検知したら，変数 A を別の変数 B に代入し，変数 A に手順 1 と同様の方法でエポックからの経過時間を取得し，代入する．

(3) ボタンの押下が 2 回目以降であれば，手順 2 の後に変数 B と変数 A の差を計算し，

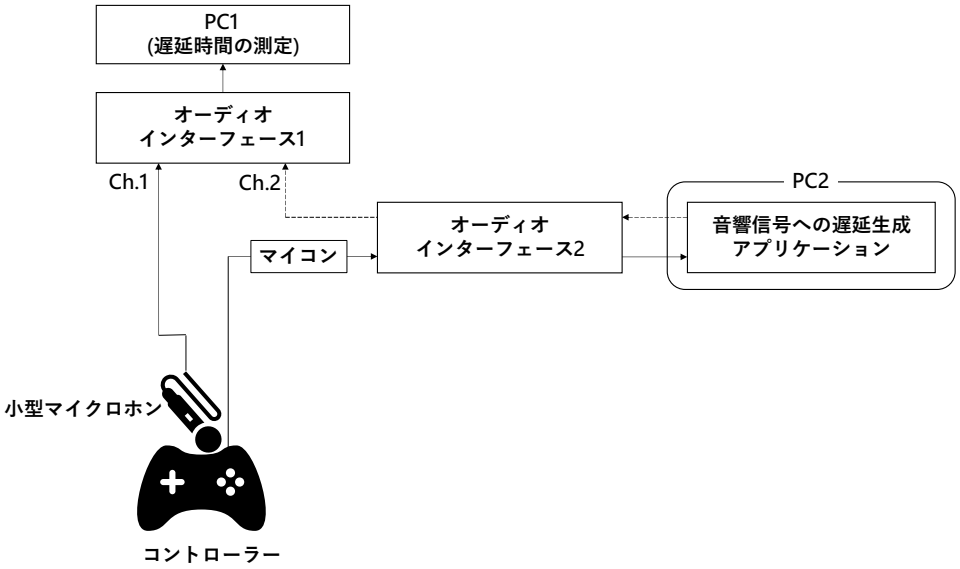

図 4.4: 遅延時間測定のための接続図

変数 C に代入する．

(4) 関数 std::chrono::duration_cast<std::chrono::milliseconds>() によって変数 C をミリ秒単位の時間に変換する．

(5) 手順 3 で変換した変数 C を随時，動的な配列に追加していくことで全てのボタンの押下時間間隔を記録する．

4.3　生成する遅延時間の正確性の調査

4.3.1　遅延時間の測定方法

アプリケーションによって生成された遅延時間の正確性を検証する目的で，遅延時間の測定を行う．図 4.4 に遅延時間の測定の接続図を示す．測定手順は以下の通りである．

(1) 測定システムを図 4.4 に示された構成で用意する．

(2) 遅延時間測定のためのプログラムと遅延時間生成アプリケーションが起動してから十

図 4.5: 小型マイクロホンを装着しているコントローラー

分に時間が経過した状態で，実験者はボタンを押下する．

(3) 図 4.5 のような小型マイクロホンを取り付けたコントローラーから取得した音をオーディオインターフェース 1 のチャンネル 1 へと入力する．

(4) コントローラーのボタンを押下することで，遅延生成アプリケーションを介して出力される音声をオーディオインターフェース 1 のチャンネル 2 へ入力する．

(5) チャンネル 1 に入力された信号の開始点から 1000 点目の点から 6000 点目の点の信号の振幅の平均と標準偏差を求める．チャンネル 1 において，平均 + 2 × 標準偏差を超えた点を検出し，チャンネル 1 に入力された信号（小型マイクロホンから取得した音響信号）の開始点とする．

(6) チャンネル 2 においても同様に，平均 + 2 × 標準偏差を超えた点を検出し，これをチャンネル 2 における信号（アプリケーションによって生成された音響信号）の開始点とする．

(7) それぞれの開始点の時間差を計算し，これを遅延時間 [ms] として算出する．

システムの動作確認は，次の方法で実施した．アプリケーションの遅延時間の設定値を

表 4.2: 測定に使用した機器

実験装置	製造会社	製品名
オーディオインターフェース 1	Roland	Duo-Capture EX
オーディオインターフェース 2	Forcusrite	Scarett-solo 3rd Generation
遅延時間生成用の PC	HP	HP ProBook 450 G7
遅延時間測定用の PC	HP	pavilion
マイクロホン	DPA Microphones	IMK-SC4060

10[ms] に設定し，生成される遅延時間を計測した．このプロセスでは，PC 上で他アプリケーションによる CPU 占有が生じてもシステムが正常に機能するかを検証するため，遅延時間生成用 PC の全 CPU コアを利用して重い計算を実行し，CPU 使用率を 100 %に保ち，この状態と通常の CPU 負荷がない状態の両方で遅延時間を測定した．CPU 使用率を 100 %にするプログラムと遅延時間測定プログラムは，付録 B に掲載している．この測定は合計 10 回行い，遅延時間の理論値と実測値の差の絶対値の平均および実測値の標準偏差を算出した．使用した機器の詳細は表 4.2 に示す．また，設定するバッファサイズをオーディオインターフェースで設定可能な最小のバッファサイズである 16 とし，遅延時間を 10-40[ms] の範囲では 5[ms] ずつ，50-150[ms] の範囲では 20[ms] ずつ変化させ，それぞれの場合において生成された遅延時間を 4.3.1 項で説明した方法で測定する．

4.3.2　測定結果

表 4.3 に測定結果を示す．表 4.3 より CPU 利用率が平常時においては，理論値の誤差と実測値の差が小さいことから，アプリケーションの正常な動作が確認された．一方で，CPU 利用率が 100 %の状態では，理論値と実測値の差および実測値の標準偏差が著しく増加することが確認された．これは，本研究で開発されたアプリケーションが，CPU 利用率が 100 %の際には正常に機能しないことを示している．したがって，本研究で開発されたアプリケーションを用いてボタン押し課題を実施する際には，CPU 利用率が 100 %

表 4.3: 測定結果

CPU 利用率	オーディオ入出力のバッファサイズ	理論値と実測値の差の絶対値の平均 [ms]	実測値の標準偏差 [ms]
平常時	16	0.129	1.23
100[%] 時	16	5.08	4.93

に至らないよう注意が必要であり，実験を行う研究者は，他のアプリケーションを終了させる等の措置を講じ，CPU 利用率が 100 %にならないように慎重に実験を進める必要がある．

第 5 章

遅延聴覚フィードバックが身体運動に与える影響の評価方法

本章では，遅延聴覚フィードバックが身体運動に与える影響の評価方法を述べる．4 章で説明した方法で被験者がボタンを押下する時間を記録すると，被験者に提示するボタンの押下の時間間隔が毎分 60 回であれば，理想的には 1000[ms] の間隔でボタンが押下される．しかし，人がボタンを押下すると理想的なボタン押下の時間間隔にばらつきが生じると考えられる．また，遅延聴覚フィードバックが身体運動に影響を与えて入れば，このばらつきは聴覚フィードバックの遅延時間の大きさによって変化するものであると考えらえる．そこで，このボタン押下の時間間隔のばらつきを各遅延時間で評価することで，遅延聴覚フィードバックが身体運動に与える影響を評価する．ばらつきの評価には，分散と二乗誤差を用いる．これらの評価指標は，ばらつきが大きくなるほど値が大きくなるため，評価指標の計算結果の大小でばらつきの大きさを観察することが可能になると考えられる．そして，これらの評価指標を算出するとき，評価指標の値の時間変化や評価に使用するデータの個数によって値が変化を比較することを想定して，これらをパラメータとして用いる．

5.1　分散

まず、ボタンを押下する時間間隔の不偏分散 s_a^2 は，被験者が行うボタン押下の時間間隔を用いて算出する．s_a^2 は，以下の式により示される．

$$s_a^2 = \frac{1}{l-1} \sum_{i=k}^{k+l-1} (x_i - \bar{x}_{kl})^2 \tag{5.1}$$

ここで l[回] は分散を算出するために使用するデータの個数，k は分析するデータの最初のインデックス，x_i[ms] は，取得した i 番目のボタンの押下時間間隔のデータ，$\bar{x}_{kl}$[ms] は，k 番目のデータから n 個のデータを用いて算出するボタンの押下時間間隔のデータの平均値を指す．s_a^2 において，任意の i で理想的な時刻にボタンが押下されなかった場合，x_i と x_{i+1} の両方に理想的なボタンの押下時間間隔との差異が生じる．データの平均値との差異を分析する際，大きな誤差が発生した場合，その影響で分散が過大になり，適切な評価が困難になる可能性がある．したがって，各被験者のボタンの押下時間間隔のデータの中央値を真値とする分散を検討することが有効である．中央値を真値として用いることにより，データに極端な誤差が生じた場合でも、ボタンの押下時間間隔のばらつきを適切に評価することが可能になると考えられる．各被験者のボタンの押下時間間隔のデータの中央値を真値とする分散 s_2^2 は，以下の式により示される．

$$s_b^2 = \frac{1}{l} \sum_{i=k}^{k+l-1} (x_i - M_{kl})^2 \tag{5.2}$$

ここで，M_{kl} は，k 番目のデータから n 個のデータを用いて算出するボタンの押下時間間隔のデータの中央値を指す．次に，真値を理想的なボタンの押下時間間隔とする場合を考える．例えば，ボタン押下の回数が毎分 60 回であれば，理想的には 1000[ms] の間隔でボタンが押下される．しかし，実際の実験では，この理想的な間隔でボタンが押下されるとは考えにくく，その理想的な間隔との誤差の分散は，ばらつきが増加するとともに大きく

なると推測される．理想的なボタンの押下時間間隔との誤差の分散 s_c^2 は，以下の式により示される．

$$s_c^2 = \frac{1}{l}\sum_{i=k}^{k+l-1}(x_i - T)^2 \tag{5.3}$$

5.2　二乗誤差

本節では，聴覚フィードバックの遅延が変則的に発生する場合のばらつき評価について検討する．遅延が t の倍数に到達したときのみ発生する状況を想定し, t の倍数に到達する直前のボタン押下間隔と直後のボタン押下間隔のデータの差の二乗平均（Mean Squared Error, MSE）が，遅延聴覚フィードバックがボタン押下間隔に与える影響を反映すると仮定する．遅延時間が増加するにつれて，MSE も増加すると予測される．MSE を算出する際に用いる誤差の総数を表す関数 $f(n)$ と，使用するデータの最後のインデックスを示す関数 $s(n)$ は，ボタンの押下回数 n を用いて以下の式で表される．

$$f(n) = \left\lfloor \frac{n-t-1}{t} \right\rfloor + 1 \tag{5.4}$$

$$s(n) = \left\lfloor \frac{n-t-2}{t-1} \right\rfloor \tag{5.5}$$

ここで，t は 2 以上の自然数，$\lfloor x \rfloor$ は x を超えない最大の整数を表す．これらの関数を用いて，MSE は次の式で定義される．

$$MSE = \frac{1}{f(n)}\sum_{i=0}^{s(n)}(d_{t-1+ti} - d_{t+ti})^2 \tag{5.6}$$

ここで，d_{t-1+ti} は t の倍数に達する直前のボタン押下間隔，d_{t+ti} は t の倍数に達した直後のボタン押下間隔のデータを示す．これにより，聴覚フィードバックの遅延による影響を定量的に評価することが可能となると考えられる．さらに，t の倍数に到達する直前のボタン押下時間間隔と到達した直後の間隔との誤差の中央値（Median Squared Error,

MedSE）での評価を検討する．この計算法により，誤差の中に極端な値が存在しても適切なばらつきの評価が行える可能性がある．誤差の中央値 $MedSE$ は以下の式で定義される．

$$MedSE = Med((d_3 - d_4)^2, (d_7 - d_8)^2, \ldots, (d_{t-1+ts(n)} - d_{t+ts(n)})^2) \tag{5.7}$$

この式における $Med()$ は中央値を計算する関数であり，括弧内の各項はボタンの押下回数が t の倍数に到達する直前のボタン押下間隔と t の倍数に到達した直後のボタン押下間隔の差の二乗を表す．このように，$MedSE$ を用いることで，データの極端なばらつきによる影響を抑えつつ，遅延聴覚フィードバックの効果をより適切に評価することが期待される．

第 6 章

身体運動のばらつきを評価するための最適な実験条件

4 章で述べたボタン押し課題でボタンの押下時間間隔を記録すると，聴覚フィードバックに遅延がない場合でも理想的なボタンの押下間隔にはならずいくらかのばらつきが発生する．これは，遅延聴覚フィードバックが身体運動に与える影響の調査において発生する本質的なばらつきであると考えられる．そのため，聴覚フィードバックに遅延がない場合でのばらつきが小さくなるような実験条件を用いれば，遅延による影響がより顕著に現れることが期待される．そこで，本章ではこの調査における最適な実験条件を明らかにするために，以下の実験から実験条件によるばらつきの変化を調査する．

実験 (1) メトロノームの合図音の BPM を変化させたときのボタンの押下時間間隔のばらつきの変化を調べる．

実験 (2) 分散の計算に用いるデータの個数の違いによるばらつきの変化を調べる．

実験 (1) と実験 (2) では，聴覚フィードバックの遅延時間を一般的に影響のない遅延時間とされている 10[ms] に設定する．

6.1　ボタンを押下する間隔の最適な条件

この実験では，1 分間にボタンを押下する回数を 70 回から 110 回までの 5 種類に変化させる．20 代の被験者 8 人を対象に調査を行った．図 6.1 にボタンを押下する間隔と評価指標の関係を示す．この図から 1 分間に 80 回の間隔でボタンを押下するときが最も標準偏差が小さいことがわかる．このため，これ以降の実験では，ボタンを押下する時間間隔を 1 分間に 80 回とする．

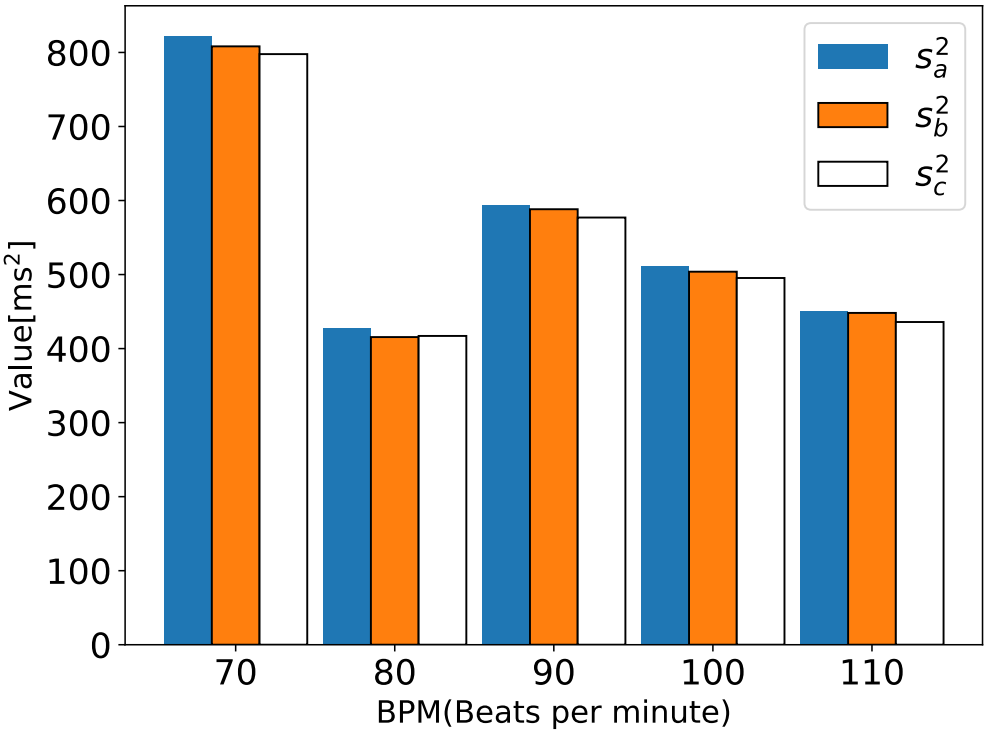

図 6.1: ボタンを押下する間隔と評価指標の関係

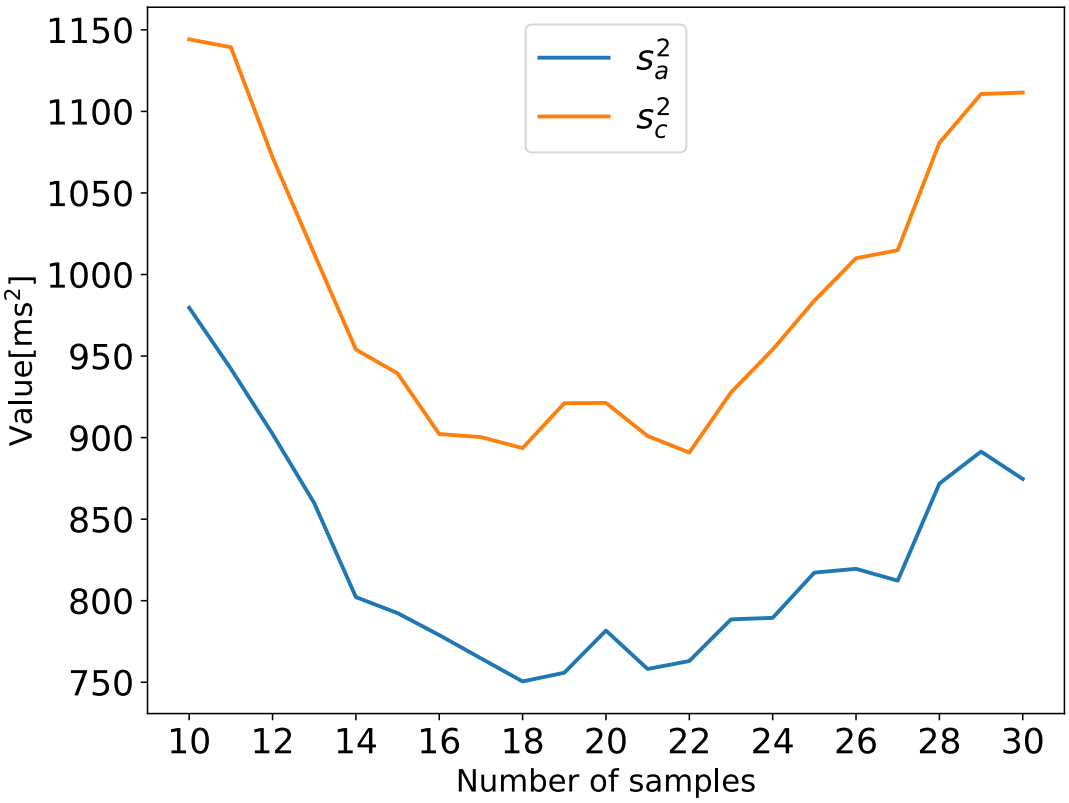

図 6.2: 使用するボタンの押下時間間隔のデータ個数と評価指標の関係

6.2　ボタンを押下する回数の最適な条件

本節では，遅延時間を 10[ms]，k を 9 に固定する．図 6.2l を 10 から 30 まで 1 ずつ変化させながら s_a^2 および s_c^2 を算出し，被験者数で平均している．図 6.3 では同様に s_b^2 を算出し，被験者数で平均している．これらの図から，評価指標の値の算出に使用するデータの個数が 20 を超えると，値がデータの個数の増加に伴い，大きくなっていくことがわかる．そのため，実験条件によって発生するばらつきを最小限に抑えるためには分析するデータの個数を 20 個以下とすればよいと考えられる．以降の実験では，観測値全体のばらつきを評価するときのデータのサイズは，20 個とする．

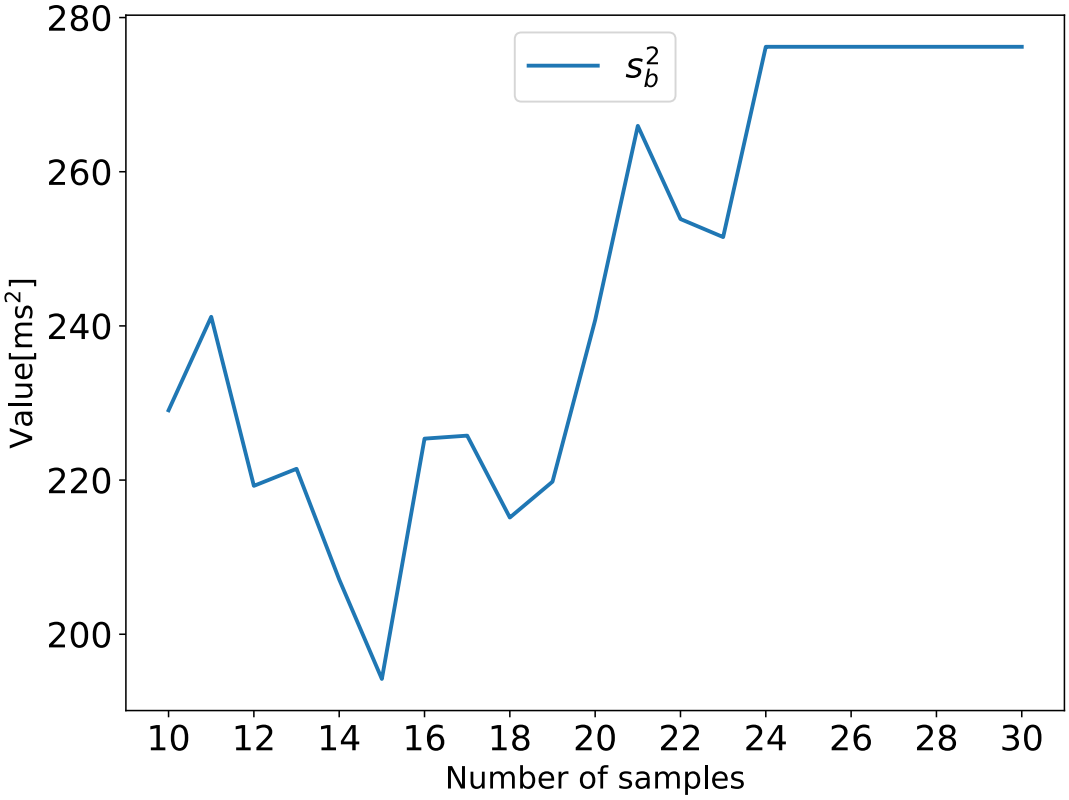

図 6.3: 使用するボタンの押下時間間隔のデータ個数と評価指標の関係

6.3　遅延を発生させる最適なタイミング

本節では，6.1 節で規定された適切なボタン押下間隔を用いて，遅延を発生させる最適なタイミングを検証する．実験 (1) では，被験者による予備調査後の口頭アンケート結果から，「遅延を感じなかった」あるいは「遅延を感じたが操作感に影響はなかった」との意見が寄せられた．この結果を踏まえ，「ボタン押下時の合図音の遅延が一定であれば，被験者は遅延に適応し，ボタン押下の時間間隔に及ぼす影響が減少する」との仮説を立て，「合図音の遅延が毎回でない場合，慣れの影響が軽減され，遅延聴覚フィードバックの影響が明確に観察される」と考えた．この仮説を検証するために，実験 (2) では，ボタンの押下回数が 4 の倍数に達したときだけ遅延を発生させるシステムを用いて，遅延聴覚

フィードバックが身体運動に及ぼす影響を分析した.

実験 (1) ボタンを押下したときの合図音が常に遅延する場合

実験 (2) ボタンを押下するときの合図音がボタンの押下回数が 4 の倍数に達したときのみ遅延する場合

上記 2 つの実験を行い，聴覚フィードバックの遅延時間とボタンの押下時間隔のばらつきの関係が読み取り可能である実験条件を探る.

6.3.1　調査方法

若年者を対象に行った遅延聴覚フィードバックが身体運動に与える影響の調査方法を以下に示す．実験 (1) と実験 (2) ともに調査の方法は同様である.

(1) 被験者に図 4.1 のシステムを用意する.

(2) 4.1 節の手順 1 から手順 2 まで述べた方法によって，ボタン押し課題を実施する.

(3) ヘッドホンから出力されるボタン押下の合図音の遅延時間をランダムに変更して実験者が指定した回数だけ手順 2 を繰り返す．このとき，実験者が提示する回数は，6.3.2 項で提示する遅延時間の種類である．また，遅延時間を変更し，次の実験に移るときには 20 秒間の休憩を挟む.

6.3.2　調査条件及び調査対象

先行研究 [7] においては，遅延聴覚フィードバックが発話に与える影響を若年者と高齢者に対して検証した．その結果，90[ms] を超える遅延時間において，若年者と高齢者の間で遅延聴覚フィードバックに対する違和感に有意な差が見られた．文献 [11] では，遅延なしの状態と比較して 100[ms] 以上の遅延があるとリズムを刻む作業が困難になることが示されている．これを踏まえ，本研究では 100[ms] 未満の遅延時間においても身体運

表 6.1: 実験 (1) における遅延時間の提示順

提示順	遅延時間 [ms]	被験者数	年齢 [平均 ± 標準偏差]
提示順 1	10, 30, 110, 10, 70, 90, 50	3	22.0 ± 0.84
提示順 2	10, 70, 30, 110, 50, 90, 10	5	22.5 ± 0.68
提示順 3	10, 110, 90, 50, 10, 30, 70	4	23.0 ± 0.72
提示順 4	10, 50, 90, 10, 30, 70, 110	4	22.5 ± 0.88
提示順 5	10, 30, 10, 50, 110, 70, 90	5	21.8 ± 0.41

表 6.2: 実験 (2) における遅延時間の提示順

提示順	遅延時間 [ms]	被験者数	年齢 [平均 ± 標準偏差]
提示順 1	10, 30, 110, 10, 70, 90, 50	4	22.5 ± 0.88
提示順 2	10, 70, 30, 110, 50, 90, 10	3	22.0 ± 0.0
提示順 3	10, 110, 90, 50, 10, 30, 70	3	23.7 ± 1.3
提示順 4	10, 50, 90, 10, 30, 70, 110	3	21.7 ± 0.49
提示順 5	10, 30, 10, 50, 110, 70, 90	3	22.7 ± 1.8

動に与える遅延聴覚フィードバックの影響を検証するため，遅延時間を 10, 30, 50, 70, 90, 110[ms] の 6 つに設定した．遅延時間の提示順序は，初めに 10[ms] を提示し，次に 10[ms] 以外の中からランダムに選択し提示する．その後，残る遅延時間に 10[ms] を加えたものをランダムに提示する．10[ms] の条件については，最初に提示したものではなくランダムに提示したものの結果を用いる．表 6.1 に実験 (1) における遅延時間の提示順序，被験者数，被験者の年齢の平均及び標準偏差を示し，表 6.2 に実験 (2) の同様の情報を示す．調査結果は，提示する遅延時間ごとに 5.1 節の s_a^2，s_b^2 及び s_c^2 を算出する．

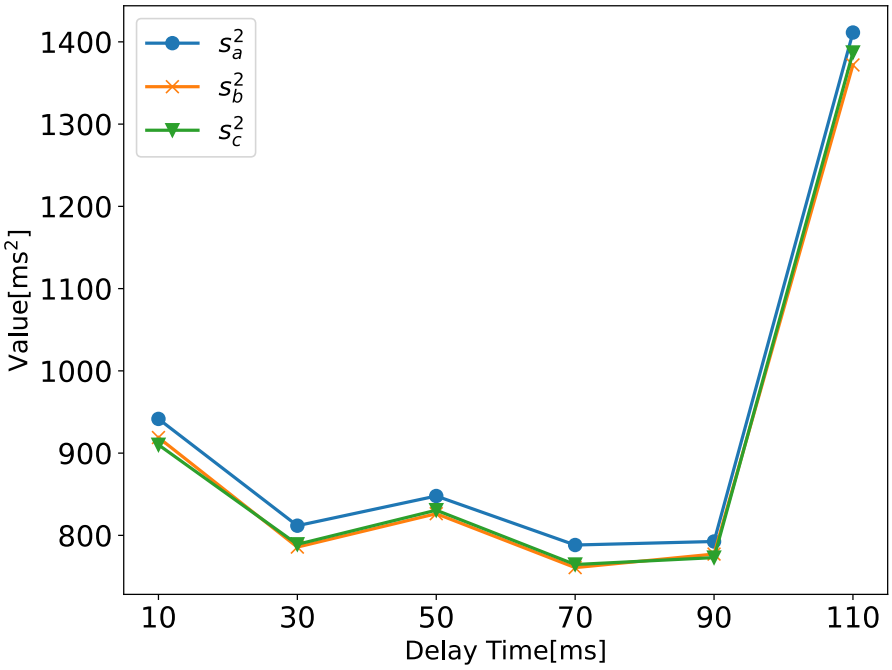

図 6.4: 実験 (1) における遅延時間と評価指標の関係

6.3.3　調査結果

図 6.4 における実験 A による遅延聴覚フィードバックの身体運動への影響の調査結果と，図 6.5 における提示順 1 による同様の影響調査結果を提示する．図 6.4 に示される結果から，10[ms] から 90[ms] の間で評価指標が減少傾向を示し，90[ms] から 110[ms] の範囲では評価指標が急激に増加していることが観察される．10[ms] から 90[ms] の間で評価指標が減少した原因としては，被験者の馴化が挙げられる．合図音の遅延が一貫しているシステム設計においては，ボタン押し課題の繰り返しにより，被験者は課題に慣れ，大きな遅延にも順応してしまうことが示唆される．対照的に，図 6.5 では，10[ms] の評価値が最小であり，遅延時間の増加に伴い評価値が上昇する傾向が見られる．これは，4 回に 1 度の遅延が身体運動に与える影響を示唆する結果である．したがって，本研究では「ボタン押下時の合図音の遅延が一定であれば，被験者は遅延に馴染み，ボタン押下時間間隔に

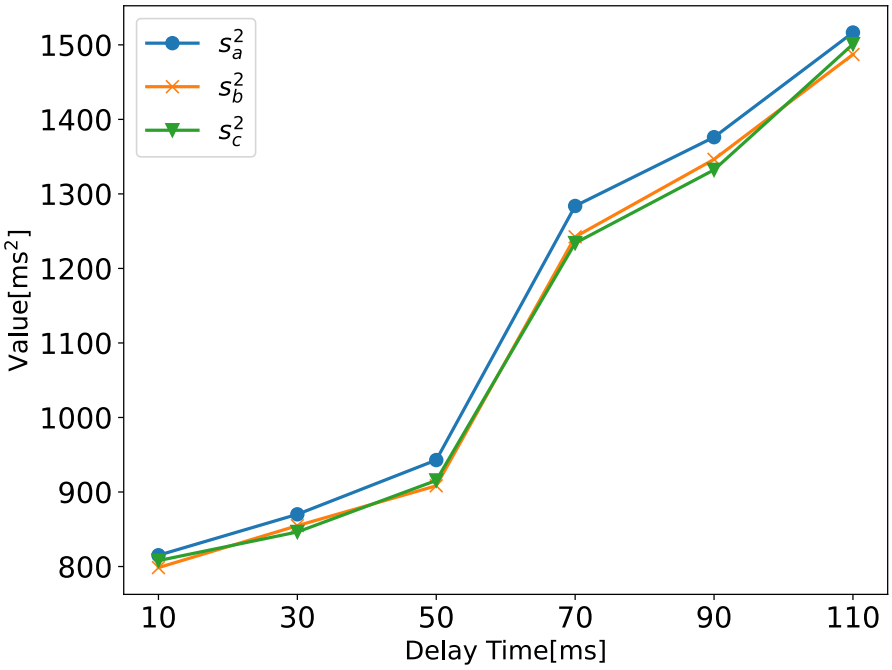

図 6.5: 実験 (2) における遅延時間と評価指標の関係

及ぼす影響が減少する」という仮説を支持し，次回以降の実験では，実験 (2) の条件下での調査を実施することとする．

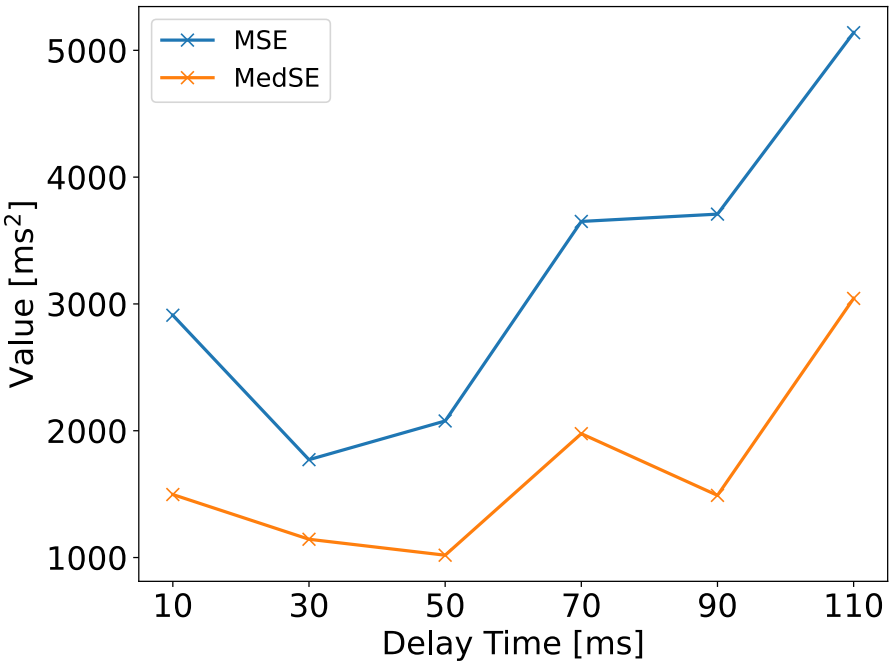

図 6.6: 実験 (2) における遅延時間と MSE および MedSE の関係

第 7 章

遅延聴覚フィードバックが身体運動に与える影響の調査

本章では，第 4 章で開発されたシステムと第 6 章で決定された実験条件を用いて，遅延聴覚フィードバックが身体運動に及ぼす影響を聴力が正常な若年者及び 60 代以上の高齢者に対して調査した結果について述べる．「高齢者は若年者に比べて聴覚フィードバックの遅延に対する許容度が高い」という仮説を，4 の倍数に達したときにのみ遅延を発生させるシステムを用いて検証する．第 6 章の予備実験により，4 の倍数に達する直前の押下間隔と 4 の倍数に達した直後の押下間隔の差の二乗の平均値及び中央値が遅延時間の増加に比例して増大すること，そして 4 の倍数に達したときの遅延が全体のボタン押下間隔のばらつきを拡大させるという可能性が示唆された．これらの予備実験で得られた考察を基に，仮説の検証を行う．

7.1　調査方法

聴力の正常な若年者及び高齢者を対象に行ったボタン押し課題による遅延聴覚フィードバックが身体運動に与える影響の調査の方法を以下に示す．

（1）被験者に図 4.1 のシステムを用意する．

（2）4 章の手順 1 から手順 2 までに述べた方法によって，ボタン押し課題を実施する．

（3）ヘッドホンから出力される音声の遅延時間をランダムに変更して実験者がアプリケーション上で指定した回数だけ手順 2 を繰り返す．このとき，実験者が提示する回数は，7.2 節で提示する遅延時間の種類である．

表 7.1: 若年者の実験 A における遅延時間の提示順

提示順	遅延時間 [ms]	被験者数	年齢 [平均 ± 標準偏差]
提示順 1	10, 30, 110, 10, 70, 90, 50	8	22.1 ± 0.78
提示順 2	10, 70, 30, 110, 50, 90, 10	8	22.4 ± 0.99
提示順 3	10, 110, 90, 50, 10, 30, 70	8	22.3 ± 1.09
提示順 4	10, 50, 90, 10, 30, 70, 110	7	22.7 ± 0.88
提示順 5	10, 30, 10, 50, 110, 70, 90	7	23.1 ± 0.64

表 7.2: 若年者の実験 B における遅延時間の提示順

提示順	遅延時間 [ms]	被験者数	年齢 [平均 ± 標準偏差]
提示順 1	10, 25, 35, 30, 40, 20, 10, 15	7	22.6 ± 1.29
提示順 2	10, 15, 40, 10, 35, 25, 30, 20	7	22.9 ± 1.25
提示順 3	10, 30, 25, 10, 40, 20, 35, 15	7	23.6 ± 1.18
提示順 4	10, 30, 20, 10, 15, 35, 25, 40	7	22.1 ± 1.36
提示順 5	10, 40, 10, 25, 30, 20, 15, 35	6	23.5 ± 0.76

7.2　調査条件及び調査対象

遅延時間の提示順序については，最初に 10[ms] を提示し，その後 10[ms] 以外の中からランダムに提示する．提示する遅延時間の種類を表 7.1，表 7.2，表 7.3，表 7.4 に示す．本研究では，10, 30, 50, 70, 90, 110[ms] の 6 種類の遅延時間を用いた実験を実験 A とし，10, 15, 20, 25, 35, 30, 40[ms] の 7 種類の遅延時間を用いた実験を実験 B と称する．

表 7.3: 高齢者の実験 A における遅延時間の提示順

提示順	遅延時間 [ms]	被験者数	年齢 [平均 ± 標準偏差]
提示順 1	10, 30, 110, 10, 70, 90, 50	8	63.5 ± 3.94
提示順 2	10, 70, 30, 110, 50, 90, 10	8	70.1 ± 4.96
提示順 3	10, 110, 90, 50, 10, 30, 70	8	68.4 ± 4.50
提示順 4	10, 50, 90, 10, 30, 70, 110	9	70.2 ± 6.48
提示順 5	10, 30, 10, 50, 110, 70, 90	8	74.1 ± 5.09

表 7.4: 高齢者の実験 B における遅延時間の提示順

提示順	遅延時間 [ms]	被験者数	年齢 [平均 ± 標準偏差]
提示順 1	10, 25, 35, 30, 40, 20, 10, 15	8	70.0 ± 7.28
提示順 2	10, 15, 40, 10, 35, 25, 30, 20	8	72.1 ± 10.7
提示順 3	10, 30, 25, 10, 40, 20, 35, 15	8	67.9 ± 8.75
提示順 4	10, 30, 20, 10, 15, 35, 25, 40	9	75.2 ± 7.87
提示順 5	10, 40, 10, 25, 30, 20, 15, 35	8	73.1 ± 4.54

7.3　調査結果

7.3.1　観測値の分布

本節では，遅延時間ごとに得られたボタンの押下時間間隔のデータの分布を箱ひげ図を用いて示す．箱ひげ図は，観測値の中央値，第 1 四分位数，第 3 四分位数を示すことで，データの分布状況および中心的傾向を効果的に視覚化する．加えて，箱ひげ図に外れ値を示すことで，データのばらつきを直接的に評価することが可能となる．外れ値の算出は，以下の手順により行う．

（1）各遅延時間におけるボタンの押下時間間隔のデータを，被験者ごとに集計する．

（2）各遅延時間でのボタンの押下時間間隔のデータの中央値，第 1 四分位数，第 3 四分位数を算出する．

（3）第 3 四分位数から第 1 四分位数を引いた値である四分位範囲 (Interquartile Range, IQR) を計算する．

（4）(3) で求めた IQR を 1.5 倍し，この値を第三四分位数に加算および第一四分位数から減算した結果を上下の閾値と定める．

（5）閾値内の最も遠いデータまで箱からひげを引く．

（6）各データと閾値を比較し，4 で求めた閾値の外にある値を全て外れ値とし，点で表す．

これにより，データの基本的な統計的特徴を簡潔に表現し，後続の分析におけるデータの解釈を支援する．図 7.1，図 7.2，図 7.3 および図 7.4 に遅延時間ごとのボタンの押下時間間隔のデータの分布を示す．これらの図より，それぞれの遅延時間でのボタンの押下時間間隔のデータには，一定数の外れ値が存在することがわかる．したがって，本研究では外れ値を除外したデータを用いて分析を行う．この方法により，これから行う分析において，外れ値の影響を排除し，適切な評価を行うためのデータを用いることができる．

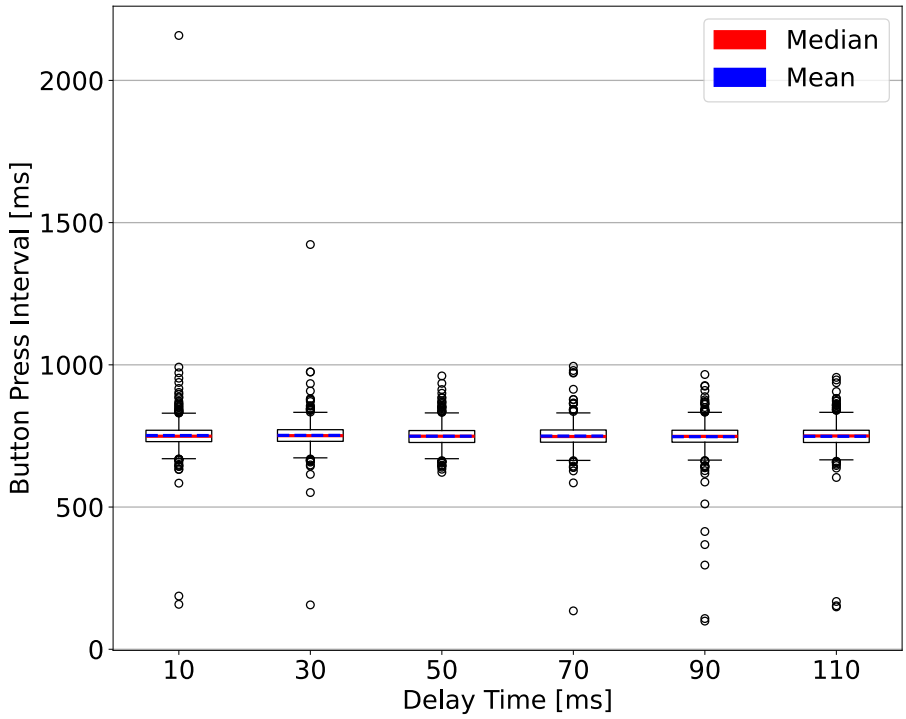

図 7.1: 実験 A における遅延時間ごとの若年者のデータの分布

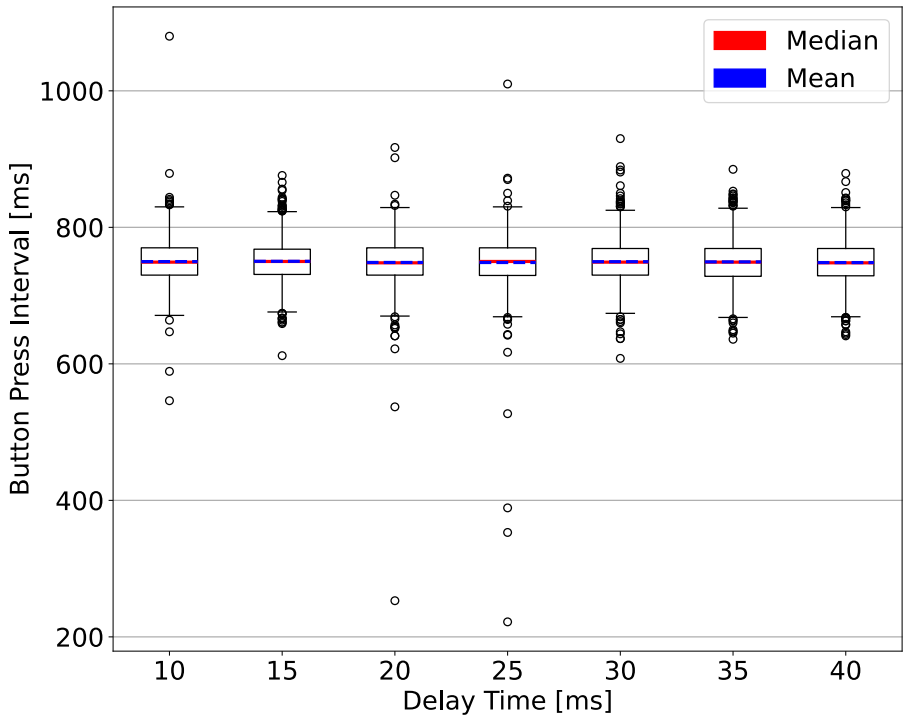

図 7.2: 実験 B における遅延時間ごとの若年者のデータの分布

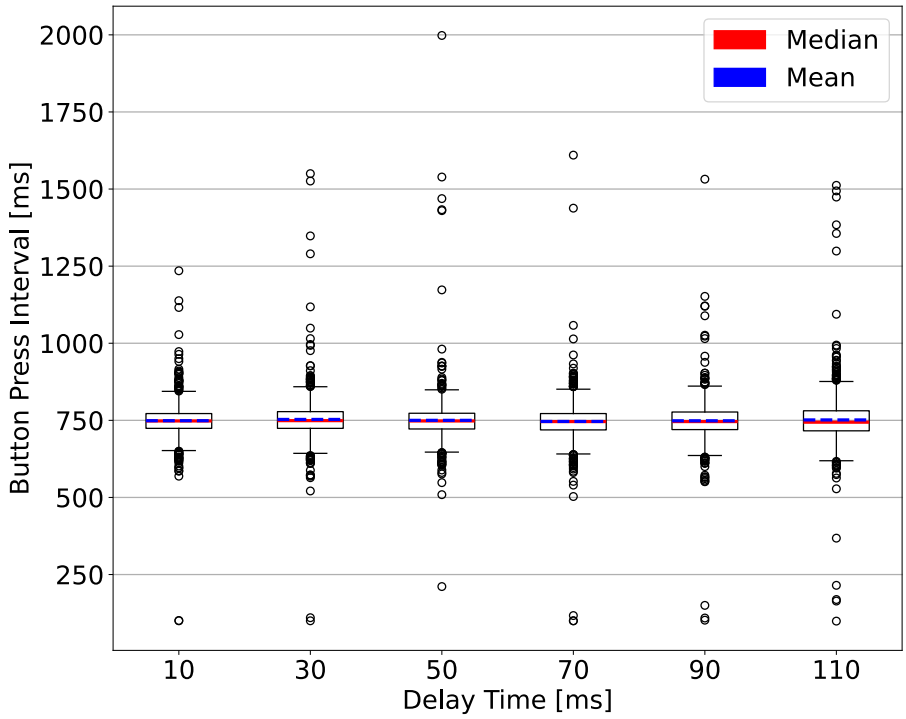

図 7.3: 実験 A における遅延時間ごとの高齢者のデータの分布

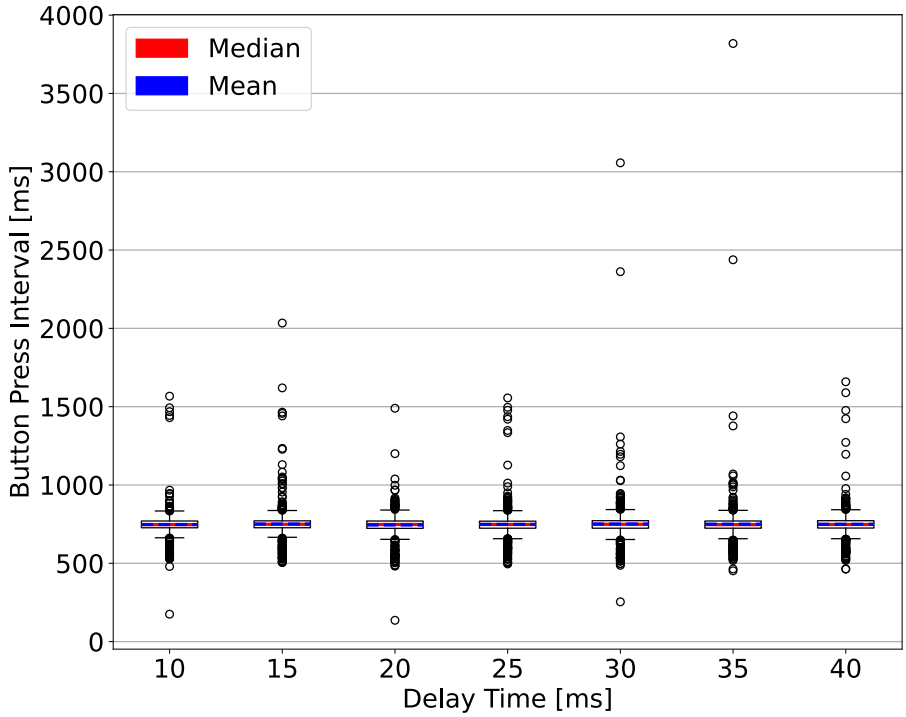

図 7.4: 実験 B における遅延時間ごとの高齢者のデータの分布

7.3.2　遅延時間と評価指標の関係

本節では, 7.3.1 項で説明した方法によりデータの外れ値を除去した後の遅延聴覚フィードバックが身体運動に与える影響の調査結果を標本全体の分散，遅延直前後の押下間隔の差の二乗の平均値（Mean Squared Error, MSE），遅延直前後の押下間隔の差の中央値（Median Squared Error, MedSE）の 3 つの評価指標を用いて示す．遅延時間と評価値の関係が若年者と高齢者で異なるかどうかを検証するために，若年者と高齢者それぞれの結果を同一グラフ上に表示し，比較する．図 7.5，図 7.6 に実験 A における遅延時間と評価指標の関係，図 7.7，図 7.8 に実験 B における遅延時間と評価指標の関係，図 7.9，図 7.10 に実験 A，実験 B における遅延時間と評価指標の関係において，10ms 時の評価値を基準に正規化した場合の結果を示す．図 7.11，図 7.12 に MSE および $MedSE$ の結果を，図 7.13，図 7.14 に MSE および $MedSE$ の結果を 10ms 時の評価値で正規化した場合の結果を示す.

はじめに，実験 A における結果について述べる．実験 A の結果において，ボタンの押下時間間隔全体の分散を評価値とする指標によれば，正規化されたデータでは若年者と高齢者の反応に顕著な差異が見られる．若年者は，一貫して評価値が緩やかに増加するのに対し，高齢者は 110[ms] を境に分散が大幅に増加することが示された．また，実験 A における MSE および $MedSE$ による評価からも，遅延時間の増加に伴い，若年者と高齢者双方の評価値に増加の傾向が確認された．この結果は，ボタン押し課題で提供される聴覚フィードバックの遅延が長くなるにつれて，遅延聴覚フィードバックの影響がより顕著になることを示唆している．若年者は遅延時間の増加に対して比較的均一な反応を示したが，対照的に高齢者は 70[ms] まで比較的緩やかな増加を見せた後，90[ms], 110[ms] の遅延時間において顕著な反応の増加を示した．これは，高齢者がある程度の遅延には許容度を持つことを示唆しており，遅延時間が 110[ms] に達した際には，若年者との差異が特に明確になった．これらの結果から，高齢者が若年者と比較して聴覚フィードバックの遅延

に対する許容度が高い可能性が示され，結果として遅延時間の増加に対する高齢者の感知の鈍さが推測される．さらに，高齢者の評価値が若年者と比較して一貫性に欠けることからも，高齢者が遅延に対して鈍感である可能性も示唆される．

次に，実験 B における結果について述べる．実験 B においてもボタン押し課題の押下時間間隔全体の分散を通じて分析した結果，若年者と高齢者での間で異なる傾向が観察された．正規化されたデータに基づく評価では，10[ms] から 40[ms] の遅延時間帯において，若年者の分散は一貫して低い水準を維持しているのに対し，高齢者は特定の遅延時間帯で分散の顕著な増加を示している．このことは，若年者が遅延時間の増加に対して一貫性を維持しやすく，高齢者が一定の遅延時間を超えると，その影響をより強く感じることを意味していると解釈できる．非正規化されたデータに基づく分析でも，高齢者は遅延時間に対して比較的一貫した分散の値を示しているが，若年者に比べて全体的に高い分散値を記録している．これは，高齢者が遅延時間の増加に対して若年者よりも鈍感であることを示唆し，逆に若年者が遅延時間の変化に対してより敏感である可能性を示している．また，MSE および $MedSE$ による評価では，10[ms] から 40[ms] の比較的短い遅延時間帯において，若年者における評価値が緩やかに増加する傾向が認められ，若年者が遅延時間の増加に対して比較的敏感であること，そして遅延聴覚フィードバックによる影響をより感じやすいことを示唆している．一方で，高齢者においては，遅延時間と評価値の間に一貫した関係が見出されず，遅延に対する感受性の低さが示唆される．

以上の結果をまとめると，10[ms] から 40[ms] の短い遅延時間帯における観察結果から，若年者の反応は遅延時間の増加に伴い緩やかに増加する傾向にあるが，高齢者の反応には一貫した関係が認められないことが示された．このことは，若年者が遅延に対して敏感であり，一方で高齢者が遅延時間に対してある程度の許容度を持っていることを示唆している．一方，遅延時間を 10[ms] から 110[ms] に拡大した場合，特に 90[ms] を超える長い遅延時間帯における高齢者の反応に大幅な増加が観察され，高齢者が遅延時間の増加に対して比較的鈍感であるものの，90[ms] を超えるとタスクの一貫性を保つことが困難に

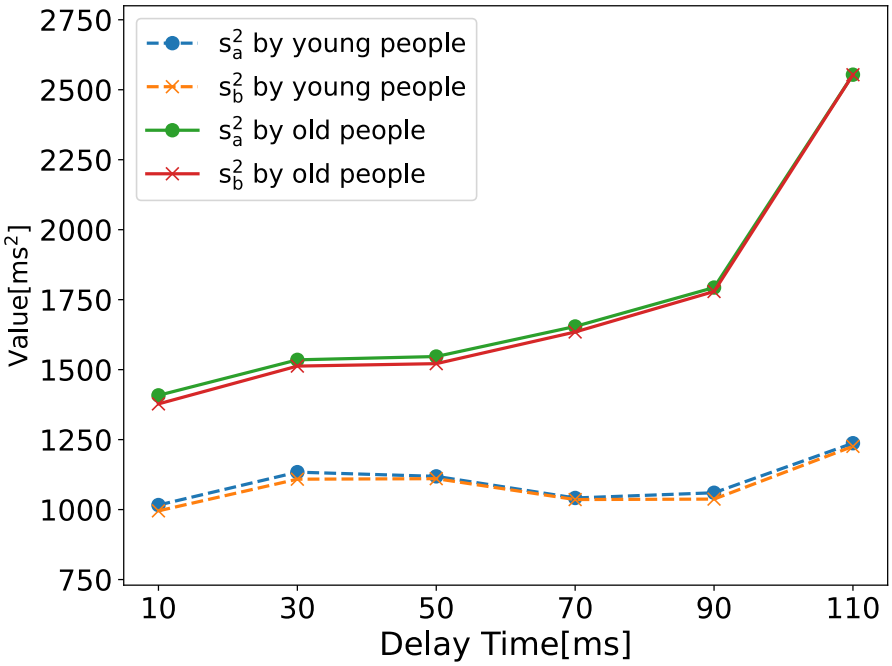

図 7.5: 実験 A における若年者と高齢者の評価値の比較

なることが示された．若年者も長い遅延時間において，反応の増加を示したが，この増加は高齢者ほど急激ではなかった．これらの結果は，遅延時間が増加するにつれて若年者と高齢者の反応の差異が顕著になることを示し，若年者は短い遅延時間帯でも遅延を感じやすく，高齢者は長い遅延時間において特に顕著な反応を示したことを明らかにする．さらに，遅延時間に対する年齢別の感受性の差異は，遅延聴覚フィードバックの適用範囲および効果を最大化するための異なるアプローチの重要性を協調している．

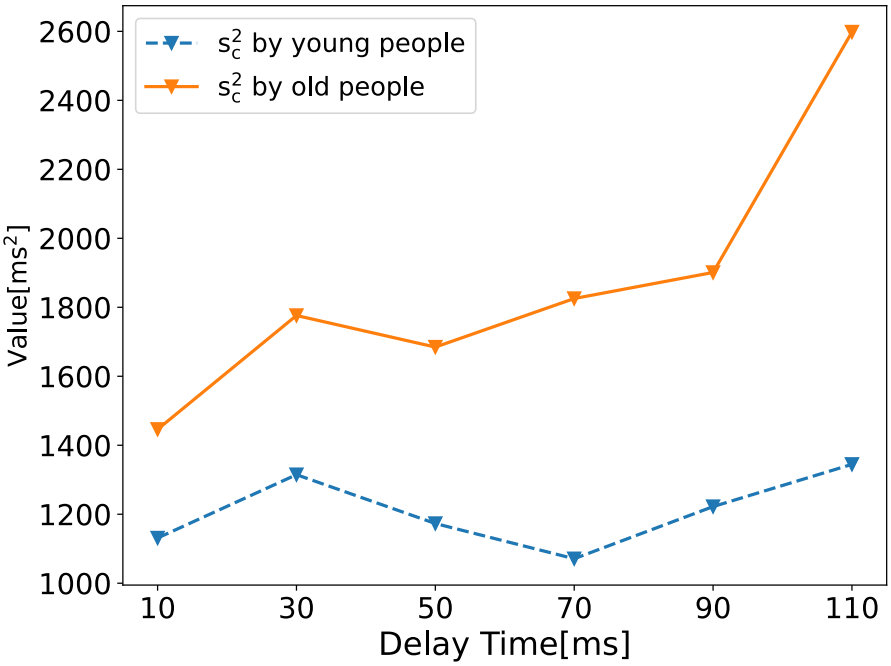

図 7.6: 実験 A における若年者と高齢者の評価値の比較

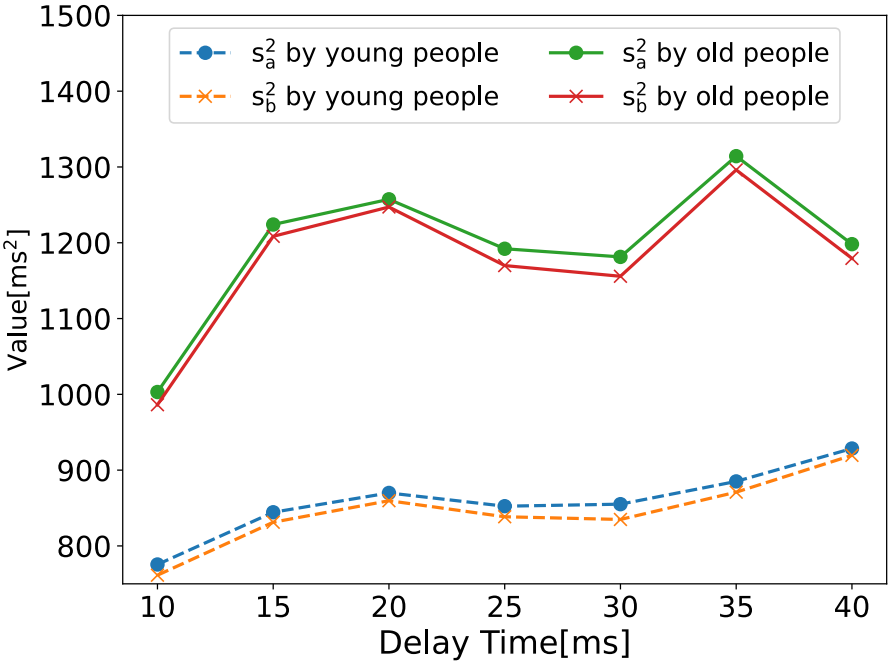

図 7.7: 実験 B における若年者と高齢者の分散の比較

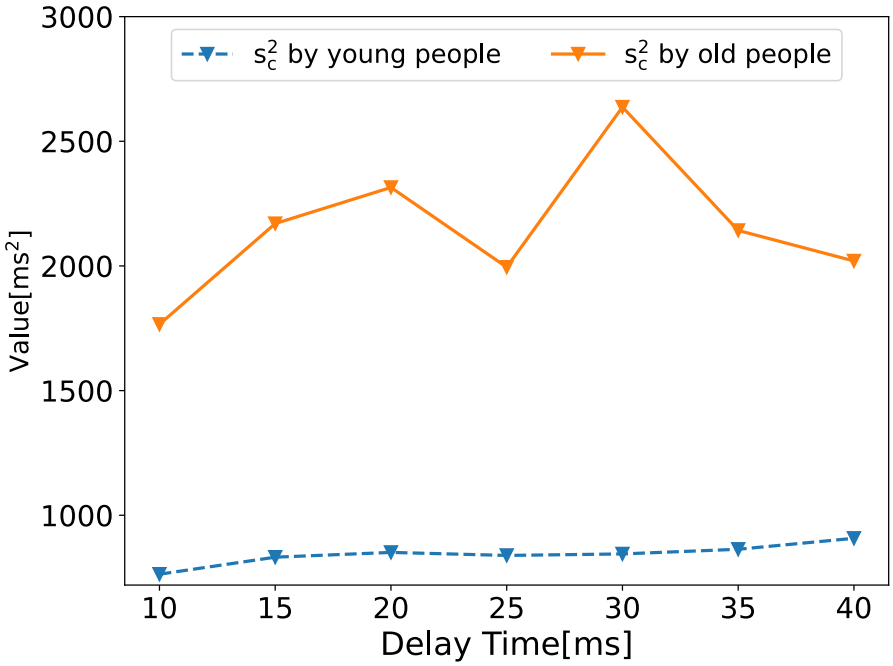

図 7.8: 実験 B における若年者と高齢者の評価値の比較

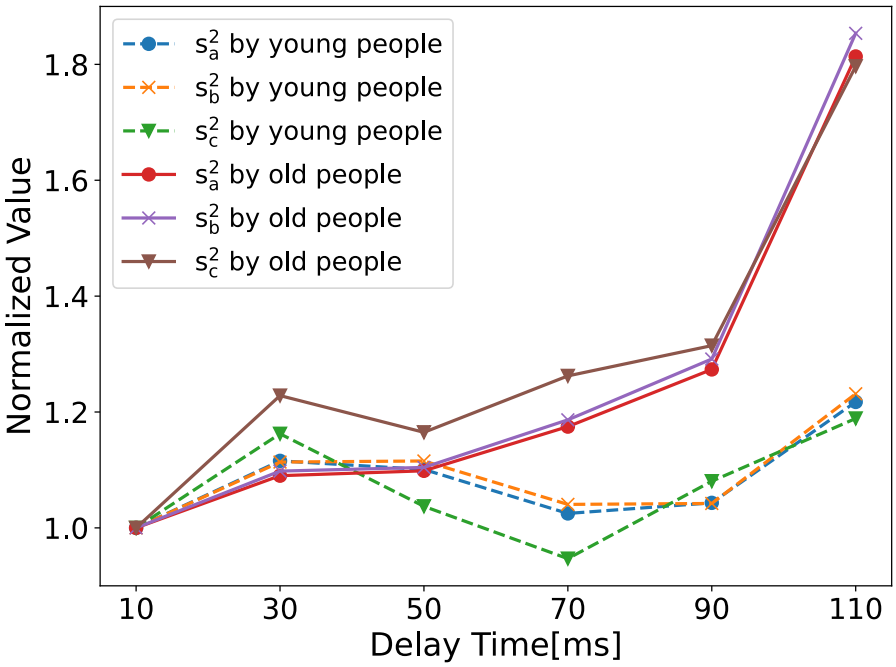

図 7.9: 実験 A における若年者と高齢者の正規化した評価値の比較

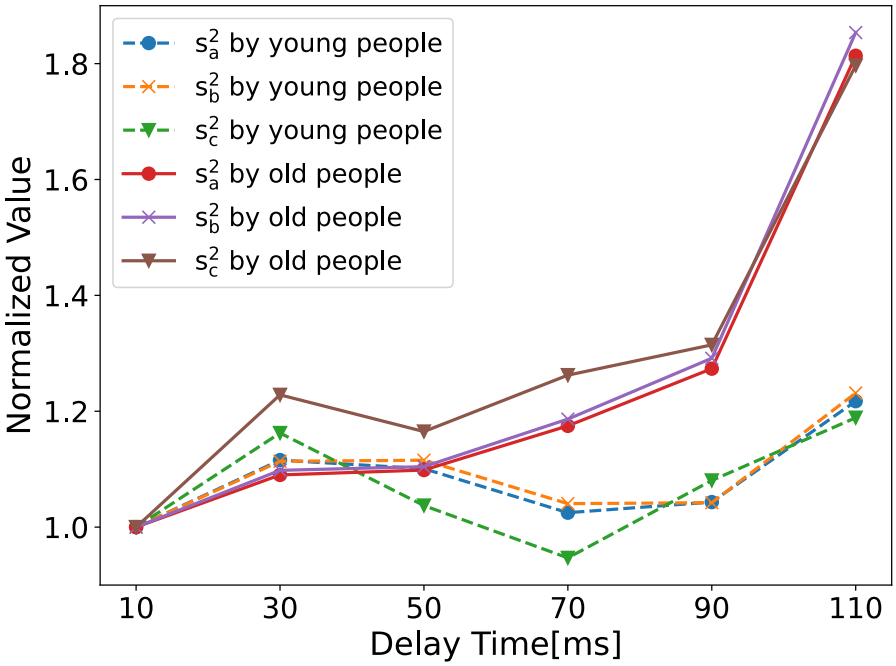

図 7.10: 実験 B における若年者と高齢者の正規化した評価値の比較

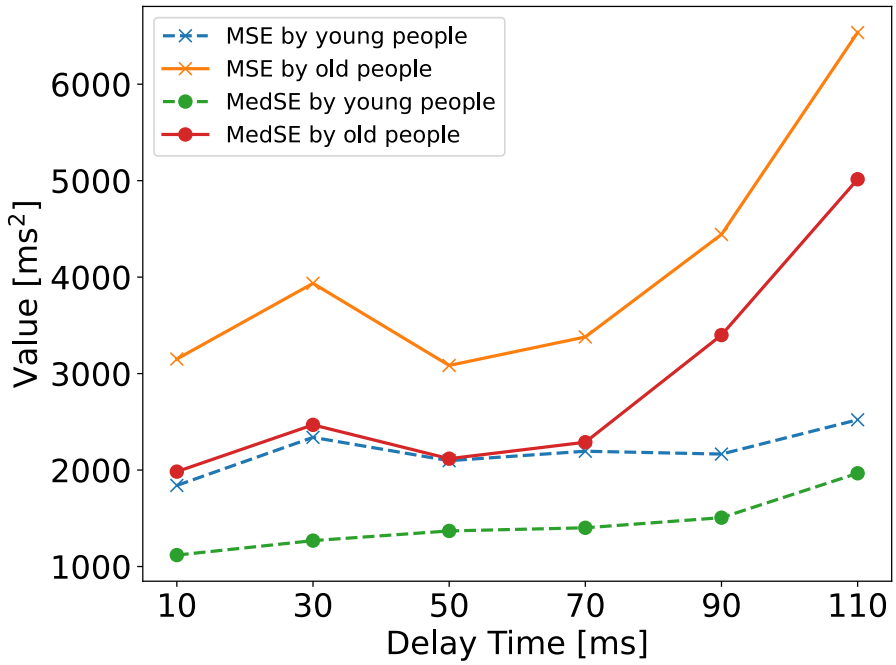

図 7.11: 実験 A における若年者と高齢者の MSE と MedSE の比較

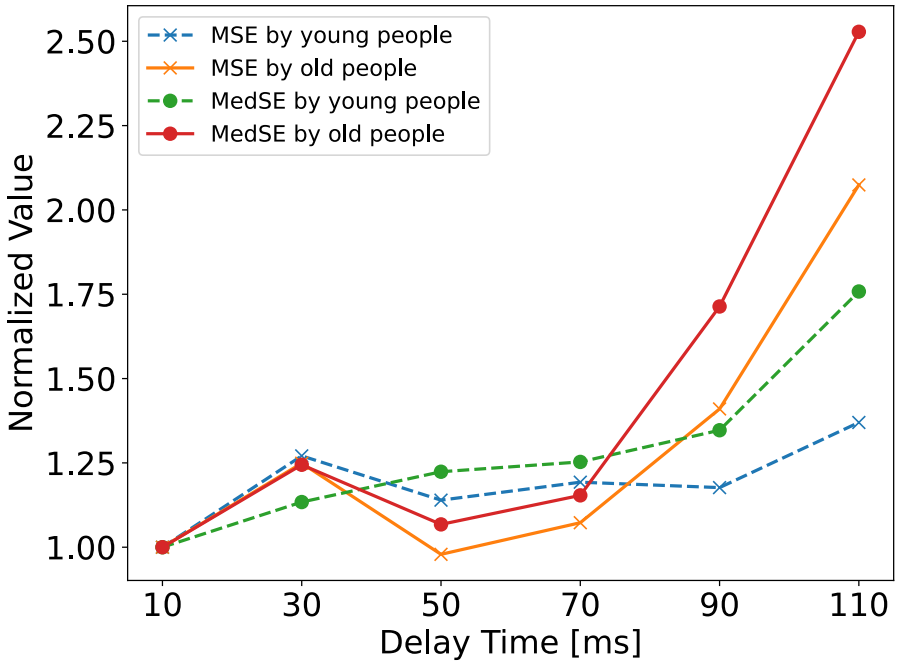

図 7.12: 実験 A における若年者と高齢者の正規化した MSE と MedSE の比較

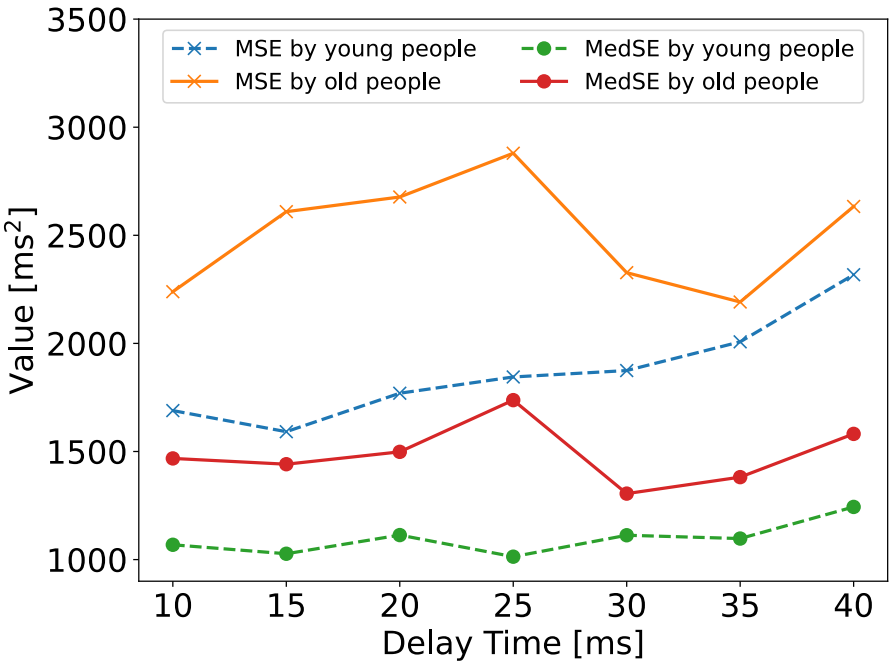

図 7.13: 実験 B における若年者と高齢者の MSE と MedSE の比較

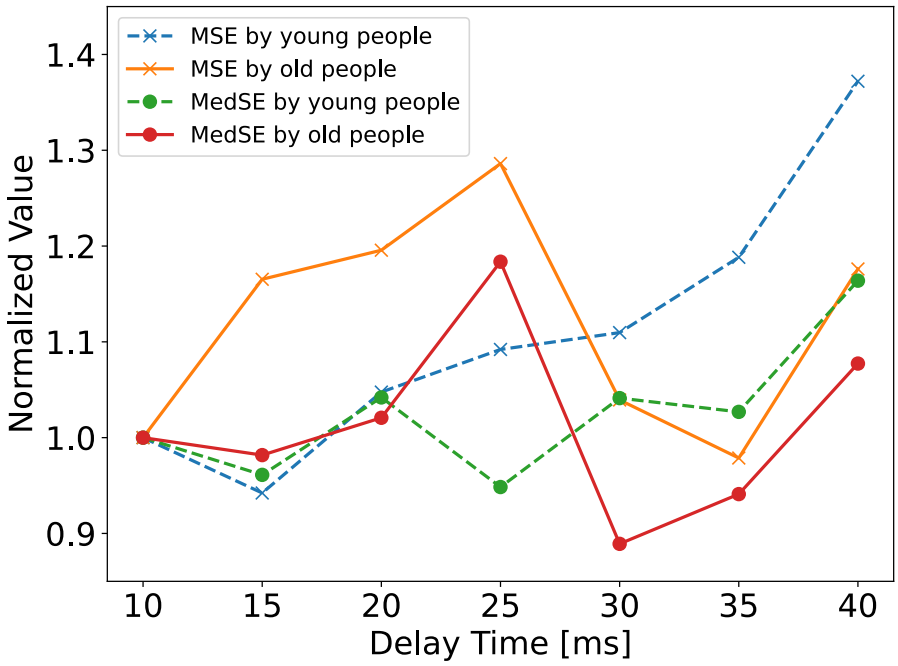

図 7.14: 実験 B における若年者と高齢者の正規化した MSE と MedSE の比較

第 8 章

結論

8.1 まとめ

本研究では，将来的に実施予定の遅延聴覚フィードバックが発話に及ぼす影響の調査に向け，アプリケーションを開発した．このアプリケーションにより，被験者がアプリケーション上に直接評価結果を入力し，その結果を外部ファイルに出力する機能を備えている．この開発により，実験の効率化および評価結果の分析が容易になることが期待される．

また，聴覚フィードバックによる違和感を客観的に評価するために，先行研究 [6] で開発されたシステムを改良し，遅延聴覚フィードバックの身体運動への影響をボタン押し課題を通じて調査した．被験者は一定の時間間隔でボタンを押下し，ヘッドホンからの聴覚フィードバックの遅延がボタンの押下回数が 4 の倍数に達した際のみ発生する条件で課題を実施した．この方法により，遅延聴覚フィードバックの影響によるボタンの押下間隔のばらつきを観察しやすくなるよう，客観的な評価指標および調査条件を定めた．その結果，ボタン押し課題では，ボタンを押下する間隔を 1 分間に 80 回とし，客観的な評価指標としてボタン押下間隔の平均及び中央値を真値とする分散，およびボタンの押下回数が 4 の倍数に到達する直前の押下間隔と 4 の倍数に到達した直後の押下間隔の平均二乗誤差

及び中央値二乗誤差を用いることが適切であると判断した．

そして，聴力の正常な若年者と高齢者を対象に，ボタン押し課題を用いて遅延聴覚フィードバックの身体運動への影響を調査した．その結果，10[ms] から 40[ms] の短い遅延時間では，若年者の評価値は遅延時間の増加に伴って，緩やかに増加する傾向が見られたが，高齢者の評価値には一貫した関係が見出されなかった．これは，若年者が遅延に敏感である一方で，高齢者が遅延時間に対して一定の許容度を持っていることを示している．遅延時間が 10[ms] から 110[ms] に及ぶ実験では，高齢者が遅延時間の増加に対して比較的鈍感であるものの，一定の遅延時間を超えるとタスクの一貫性を保つことが困難になることが明らかになった．これらの結果は，聴覚フィードバックの遅延に対する年齢別の感受性の違いを浮き彫りにし，高齢者向けの補聴器設計における重要な知見を提供するものであると考えられる．さらに，高齢者が若年者に比べてすべての遅延時間で高い評価値を示したという結果は，高齢者と若年者間の潜在的な運動能力の差異がこれらの反応に影響している可能性を示唆している．

8.2　今後の展望

今後は，高齢者と若年者の運動能力の差異を調整し，遅延聴覚フィードバックの影響をより公平に評価するために，被験者の運動能力を考慮した課題を検討する必要がある．運動能力の基本的な側面を評価するためのタスクを導入し，参加者の反応速度，精度，運動の一貫性を測定することで，運動能力の基準値を設定し，遅延時間に対する反応の差異を適切に解釈することが可能になる．また，課題の難易度や認知負荷を調整することで，運動能力以外の要因が遅延聴覚フィードバックの影響を受けるかどうかを評価することで，運動能力以外の能力が影響を与える可能性を探ることができる．

参考文献

[1] 細井裕司，“補聴器この 20 年間の進歩” 日本耳鼻咽喉科科学会会報, 114 巻, 12 号, pp.905-911, Jan.2015.

[2] 神田幸彦，“補聴器の進歩と聴覚医学「補聴器の歴史と変遷-最新補聴器の紹介-」” Audiology Japan, 60 巻, 2 号, pp. 121-128, Apr.2017.

[3] 西山崇経, 新田清一, 鈴木大介, 岡崎宏, 坂本耕二, 中村伸太郎, 上野恵, 小川郁, “補聴器装用者の満足度に関わる要因の検討” Audiology Japan, 57 巻，3 号，pp.189-194, Jun.2014.

[4] 河原英紀，“聴覚フィードバックの発話への影響: ヒトは自分の話声を聞いているのか？” 日本音響学会誌, 59 巻, 11 号, pp.670-675, Nov.2003.

[5] 硲田猛真，中村陽裕，福本儀智，長谷川賢作，北野博也，“ディレイタイムの認知閾値” Ausiology Japan, 46 巻, 5 号, pp.465-467, Sep.2007.

[6] 重松颯人，丹治寛樹，村上隆啓，松本直樹，“遅延聴覚フィードバックが身体運動に与える影響の客観的な評価方法の検討” 日本音響学会聴覚研究会資料, pp.499-504, Nov.2019.

[7] 香山実結花，山下一樹，丹治寛樹，村上隆啓，“若年者と高齢者の聴覚フィードバックにおける遅延時間の許容量の統計的分析による比較” 2022 年度電子情報通信学会東京支部学生会研究発表会, pp.113, Mar.2023.

[8] 重松颯人，村上隆啓，“高齢者の聴覚における遅延時間の許容量の評価” 平成 29 年度

電子情報通信学会東京支部学生会研究発表会講演論文集，pp.143, Mar.2018.

[9] 重松颯人，“補聴器を想定した音声おける入出力の遅延時間の許容範囲についての検討”，明治大学理工学部電気電子生命学科知能信号処理研究室，2019 年度修士学位請求論文.

[10] Microsoft，“Win32 API のプログラミング リファレンス” https://docs.microsoft.com/ja-jp/windows/win32/api/, 参照 2024 年 1 月 24 日.

[11] P. Q. Pfordresher and C. Palmer，“Effects of delayed auditory feedback on timing of music performance” Psychological Research, Vol. 16, pp.71-79, Jan.2002.

[12] 樋田浩一，上野佳奈子，嶋田総太郎，“身体運動に伴う遅延聴覚フィードバックの知覚順応”，日本認知科学会大会，pp.493-497，Dec.2013.

[13] 高橋浩貴，村上隆啓，“補聴器における遅延時間の許容量の評価” 平成 26 年度電子情報通信学会東京支部学生会研究発表会講演論文集, p168, pp.71–79, Sep.2015.

発表論文

[P1] 山下一樹，安田和生，丹治寛樹，村上隆啓，“若年者と高齢者の遅延聴覚フィードバックの身体運動への影響の比較” 2023 年度電子情報通信学会東京支部学生会, Mar.2024.

謝辞

本研究を進めるにあたり，多大なるご指導，ご助言を頂きました明治大学理工学部電気電子生命学科所属の村上隆啓専任講師に心から感謝いたします．また，日頃の研究活動及び本論文に多大なご助言を頂きました，明治大学理工学部電気電子生命学科所属の丹治寛樹助教に心から感謝いたします．遅延時間の許容調査における被験者募集にご協力いただいた情報科学科の井口幸洋教授，井口道子様，宮口祥子様，明大サポートの山本浩志様にも感謝の意を表します．また，本研究にご協力いただいた多くの被験者の方々に厚く御礼申し上げます．共同研究者として多くのご協力を頂いた知能信号処理研究室所属の安田和生氏と研究に関しまして多数のご助言をくださいました知能信号処理研究室の同期と後輩の皆様に心から感謝いたします．

2024 年 2 月 9 日
明治大学大学院　理工学研究科
電気工学専攻　博士前期課程
知能信号処理研究室
山下　一樹

付録 A　遅延聴覚フィードバックが身体運動に与える影響の調査の資料

遅延聴覚フィードバックが身体運動に与える影響の調査を行うときに用いた, 研究参加同意書（明治大学生用・明治大学生以外の 60 歳未満の方用・明治大学生以外の 60 歳以上の方用）, 実験概要説明ボード（60 歳の未満の方用・60 歳以上の方用），聴力検査手順説明ボード，実験手順説明ボード，音量調整ボードを掲載する.

（明治大学生用）

研究参加同意書（聴覚に与える遅延によって身体運動に現れる影響の調査）

【研究責任者の所属・職・氏名】
明治大学理工学部電気電子生命学科・専任講師・村上隆啓

【研究課題名】
補聴器における入出力時間差の許容量の年齢による変化と老人性難聴用補聴器への応用

＜被験者の方へ＞
本研究では、老人性難聴用補聴器の開発に向けて、音に遅延が発生した場合に身体運動に現れる影響の調査を行っています。今回ご参加いただく研究では、市販の音響機器およびゲームコントローラを用いた測定装置を使用します。以下の項目についてご同意いただける場合は、本同意書にご署名ください。

1. 本研究への参加は、**被験者の自由意思**によるものです。
2. 本研究への被験者としての参加を決定した後でも、また、研究が進行中であっても、自由意思により参加を中止することができます。
3. 本研究では、説明を含めて約 30 分間（1 名あたり）の実験時間を予定しています。
4. 本研究で使用する実験機器は市販の音響機器およびゲームコントローラを組み合わせたものであり、安全性が確認されています。
5. 研究の内容や安全性について疑問がある場合は、**研究者に質問**して答えを求めることができます。
6. 予め被験者へ説明することなくテストや測定を行うことはありません。もしそのような事柄が実施されようとしたときは、直ちに**研究への参加を拒否**することができます。
7. 研究データは、研究者によって厳重に保管されます。**被験者のプライバシーに関する情報および個人名を公開することはありません**。
8. 本研究への参加の拒否、または参加の途中中止による**成績や単位など学業への影響はありません**。

私は、上記の研究内容について適切かつ十分な説明を受け、研究目的、被験者の人権保護および安全確保への対策をよく理解しましたので、この研究への被験者としての参加に同意いたします。

調査実施日　西暦　　年　　月　　日

ご署名　　（　　歳）（男・女）

図 A.1: 研究参加同意書（明治大学生用）

（明治大学生以外の 60 歳未満の方用）

研究参加同意書（聴覚に与える遅延によって身体運動に現れる影響の調査）

【研究責任者の所属・職・氏名】
明治大学理工学部電気電子生命学科・専任講師・村上隆啓

【研究課題名】
補聴器における入出力時間差の許容量の年齢による変化と老人性難聴用補聴器への応用

＜被験者の方へ＞
本研究では、老人性難聴用補聴器の開発に向けて、音に遅延が発生した場合に身体運動に現れる影響の調査を行っています。今回ご参加いただく研究では、市販の音響機器およびゲームコントローラを用いた測定装置を使用します。以下の項目についてご同意いただける場合は、本同意書にご署名ください。

1. 本研究への参加は、**被験者の自由意思**によるものです。
2. 本研究への被験者としての参加を決定した後でも、また、研究が進行中であっても、自由意思により参加を中止することができます。
3. 本研究では、説明を含めて約 30 分間（1 名あたり）の実験時間を予定しています。
4. 本研究で使用する実験機器は市販の音響機器およびゲームコントローラを組み合わせたものであり、安全性が確認されています。
5. 研究の内容や安全性について疑問がある場合は、**研究者に質問**して答えを求めることができます。
6. 予め被験者へ説明することなくテストや測定を行うことはありません。もしそのような事柄が実施されようとしたときは、直ちに**研究への参加を拒否**することができます。
7. 研究データは、研究者によって厳重に保管されます。**被験者のプライバシーに関する情報および個人名を公開することはありません**。

私は、上記の研究内容について適切かつ十分な説明を受け、研究目的、被験者の人権保護および安全確保への対策をよく理解しましたので、この研究への被験者としての参加に同意いたします。

調査実施日　西暦　　年　　月　　日

ご署名　　（　　歳）（男・女）

図 A.2: 研究参加同意書（明治大学生以外の 60 歳未満の方用）

（明治大学生以外の 60 歳以上の方用）

研究参加同意書（聴覚に与える遅延によって身体運動に現れる影響の調査）

【研究責任者の所属・職・氏名】
明治大学理工学部電気電子生命学科・専任講師・村上隆啓

【研究課題名】
補聴器における入出力時間差の許容量の年齢による変化と老人性難聴用補聴器への応用

＜被験者の方へ＞
本研究では、老人性難聴用補聴器の開発に向けて、音に遅延が発生した場合に身体運動に現れる影響の調査を行っています。今回ご参加いただく研究では、市販の音響機器とゲームコントローラを用いた測定装置、および市販の聴力検査装置を使用します。以下の項目についてご同意いただける場合は、本同意書にご署名ください。

1. 本研究への参加は、**被験者の自由意思**によるものです。
2. 本研究への被験者としての参加を決定した後でも、また、研究が進行中であっても、自由意思により参加を中止することができます。
3. 本研究では、説明、聴力検査を含めて約 30 分間（1 名あたり）の実験時間を予定しています。
4. 本研究で使用する実験機器は市販の音響機器およびゲームコントローラを組み合わせたものであり、安全性が確認されています。
5. 研究の内容や安全性について疑問がある場合は、**研究者に質問**して答えを求めることができます。
6. 予め被験者へ説明することなくテストや測定を行うことはありません。もしそのような事柄が実施されようとしたときは、直ちに**研究への参加を拒否**することができます。
7. 研究データは、研究者によって厳重に保管されます。**被験者のプライバシーに関する情報および個人名を公開することはありません**。
8. 聴力検査（閾値検査）の結果は、データ分析のみに使用します。**得られた聴力検査結果に基づいて医学的な診断を行うことは一切ありません**。

私は、上記の研究内容について適切かつ十分な説明を受け、研究目的、被験者の人権保護および安全確保への対策をよく理解しましたので、この研究への被験者としての参加に同意いたします。

調査実施日　西暦　　年　　月　　日

ご署名　　（　　歳）（男・女）

図 A.3: 研究参加同意書（明治大学生以外の 60 歳以上の方用）

（60 歳未満の方用）

＜実験の概要（聴覚に与える遅延によって身体運動に現れる影響の調査）＞

私たちの研究では、よりよい補聴器を開発するためのデータを集めています。

もしも、補聴器から聞こえてくる音が、**実際の音よりも遅れて聞こえたら**、人によっては、**身体の動作も音につられて遅れる、などの影響が現れるかも**しれません。
本日は、その影響を測定する実験を行います。

この実験では、一定のリズムで鳴る**メトロノームの合図音に合わせて**、**ゲームコントローラのボタンを何回か押して**いただきます。
ゲームコントローラのボタンを押すと、ヘッドホンから「ピッ」という音が**ボタンを押した瞬間に聞こえたり**、**ボタンを押してから少し遅れて聞こえたり**します。

実験の開始と終了は、こちらが合図します。

ボタンを**押し間違えたり、押すタイミングがずれたりしても構いません**ので、リラックスしてボタンを押してください。

図 A.4: 実験概要説明ボード（60 歳未満の方用）

（60 歳以上の方用）

＜実験の概要（聴覚に与える遅延によって身体運動に現れる影響の調査）＞

私たちの研究では、よりよい補聴器を開発するためのデータを集めています。

もしも、補聴器から聞こえてくる音が、**実際の音よりも遅れて聞こえたら**、人によっては、**身体の動作も音につられて遅れる、などの影響が現れるかも**しれません。
本日は、その**影響を測定する実験**と、**聴力検査（閾値検査）**を行います。

影響を測定する実験では、一定のリズムで鳴る**メトロノームの合図音に合わせて**、**ゲームコントローラのボタンを何回か押して**いただきます。
ゲームコントローラのボタンを押すと、ヘッドホンから「ピッ」という音が**ボタンを押した瞬間に聞こえたり**、**ボタンを押してから少し遅れて聞こえたり**します。

実験の開始と終了は、こちらが合図します。

ボタンを**押し間違えたり、押すタイミングがずれたりしても構いません**ので、リラックスしてボタンを押してください。

図 A.5: 実験概要説明ボード（60 歳以上の方用）

（60 歳以上の方用）

＜聴力検査（閾値検査）の概要＞

（１）ヘッドホンを、耳全体を覆うように装着してください。
「Ｌ（青）」が左耳、「Ｒ（赤）」が右耳です。

（２）応答ボタンスティックを、黒いボタンを押せるように持ってください。
持つ手は片手でも両手でも大丈夫ですが、片手で持って親指でボタンを押すと押しやすいです。

（３）試しに、黒いボタンを何回か押してください。

（４）こちらが合図したら、聴力検査が始まります。
検査時間は５分程度です。

（５）ヘッドホンから「ピーッピーッ」という音が、いろいろな高さで鳴ります。
「ピーッピーッ」が聞こえたら、聞こえている間ずっとボタンを押し続けてください。
「ピーッピーッ」が聞こえなくなったら、ボタンをはなしてください。

（６）こちらが合図したら、聴力検査が終了します。

図 A.6: 聴力検査手順説明ボード

＜実験の手順（聴覚に与える遅延によって身体運動に現れる影響の調査）＞

（１）**ヘッドホンを装着**してください。

（２）**ゲームコントローラを両手で持って**ください。
ゲームコントローラを持った後に、**音量を調整**します。

（３）こちらが合図したら、**メトロノームの合図音に合わせて**
ゲームコントローラの丸いボタンを
右手親指で約２０回押してください。
終了も、こちらが合図します。

丸いボタンは「Ａ（赤）」「Ｂ（黄）」「Ｘ（青）」「Ｙ（緑）」のどれでも構いません。

ボタンを**押し間違えたり、押すタイミングがずれたりしても構いません**ので、リラックスしてボタンを押してください。

（４）上記（３）の実験を、**５～１０回**、実施していただきます。
途中で手順が分からなくなったり、気分が悪くなったりしたら、**気軽に申し出て**ください。

図 A.7: 実験手順説明ボード（60 歳未満の方用）

＜音量の調整（聴覚に与える遅延によって身体運動に現れる影響の調査）＞

・ゲームコントローラの丸いボタンを押すと、**ヘッドホンから「ピッ」という音が聞こえ**ます。

ためしに、**ゲームコントローラの丸いボタンを押して**みてください。

丸いボタンは「Ａ（赤）」「Ｂ（黄）」「Ｘ（青）」「Ｙ（緑）」のどれでも構いません。

・ヘッドホンから**「ピッ」という音は聞こえましたか？**
・ヘッドホンからの**「ピッ」という音は、小さくなかったですか？**
・ヘッドホンからの**「ピッ」という音は、大きすぎなかったですか？**

図 A.8: 音量調整ボード

付録B　主観調査におけるアプリケーションのプログラム

開発した主観調査におけるアプリケーションを掲載する．本プログラムは，Microsoft Visual Studio 2022 でコンパイルできるソースファイル，ヘッダファイルおよびリソースファイルである．関数の目的別にファイルを分けて作成している．

ソースコード B.1: main.cpp

```
//////////////////////////////////////////
// main.cpp: 関数・ウィンドウプロージャの定義 WinMain
//////////////////////////////////////////

#include<windows.h>
#include<stdio.h>
#include<tchar.h>
#include<WinUser.h>
#include<commctrl.h>
#include<random>
#include<string>
#include<iomanip>
#include<stdlib.h>
#include<windowsx.h>
#include<manipulations.h> // Touch 用 Input

//////////////////////////////////
/// 自作ヘッダファイルのインクルード
//////////////////////////////////
#include"main.h"
```

```
#include"window.h"
#include"resource.h"
#include"file.h"
#include"UserInfoWindow.h"

using namespace std;

////////
// pragma
// コモンコントロール 1.
// 音声再生用 2.
// ビジュアルスタイルの有効化 3.
////////
#pragma comment(lib, "comctl32.lib")
#pragma comment(lib, "winmm.lib")
#pragma comment(linker,"/manifestdependency:\"type='win32'␣\
  name='Microsoft.Windows.Common-Controls' \
  version='6.0.0.0' \
  processorArchitecture='*' \
  publicKeyToken='6595b64144ccf1df' \
  language='*'\"")

//////////
// 定数
//////////
#define STRLEN 256   // 文字列の最大長
#define GetMonitorRect(rc)  SystemParametersInfo(SPI_GETWORKAREA
    ,0,rc,0)  // ワークエリア領域の
    取得
#define CONBOMAX9 9  // コンボボックスの項目数（〜Group15)
#define CONBOMAX8 8  // コンボボックスの項目数（〜）Group610
#define LENSNUMBER 128



////////////////////////////
//グローバル変数の宣言・初期化
////////////////////////////
```

```
const char      szWinName[5] = _T("Test");
static const char   szWinName2[] = _T("テスト");
HINSTANCE hInst;
HWND hDlg;
extern char WindowTitleText[32];
extern int intCurrentIndex;

//ButtonEnableFunc()で利用
bool boolstart;  // true:「ウィンドウが生成されてから回以上次へボタンを押
    した」1 false:「ウィンドウが生成されてからまだ次へボタンを押していない」
bool ZeroOrNot;  // true:「変数がである」NumberOfTimes0 false:「変数が以
    外である」NumberOfTimes0

// ファイルに書き込んだか否かの取得
bool FileWriteInfo;

//HWND hChildUserInfo;

// 背景
extern const HBRUSH BackGround_clear = CreateSolidBrush(RGB(235,
    235, 235));
extern const COLORREF TextBackground = RGB(235, 235, 235);
const COLORREF ColorEdgeButton = RGB(100, 149, 237);

// フォントサイズ
extern int fontsize;   // エディットボックスのフォント
extern int fontsize2;

// その他変数
extern unsigned int NumberOfTimes;          // 実験回数を保持する変数
char Grade[STRLEN];                        // 評価結果の保存用変数
extern char GradeSpeak[STRLEN],GradeLate[STRLEN];
HANDLE hfile;
extern unsigned int GradeS[STRLEN], GradeL[STRLEN];

char Result[STRLEN];                // 全評価結果を保持する変数
short int IndexResult;              // Resultのインデックス番号 []
```

```
extern short int i;                    // 評価結果保存用配列のインデックス
    番号
extern char temp1[STRLEN];
char temp3[STRLEN];

// 読み上げる文章の順番を決めるための操作
extern int SNumber[LENSNUMBER];
int* P_SNumber;

// 座標
int key_i[2] = { 30, 170 };
int NumberTest_i[2] = { 40, 215 };  // 表示させる実験回数の座標

// ファイルへの出力用変数
DWORD dwWriteSize;


// コンボボックス内
int intTxtLen;
char* pszBuf;

////////////////////
// 構造体変数 RECT
////////////////////

extern RECT FrameTop, FrameProgTop, FrameCenter;
extern RECT Frame1;
extern RECT Frame2;
extern RECT Frame3;
extern RECT FrameUserInfo;
extern RECT FrameComment;
extern RECT FrameProg;
extern RECT RectButton1, RectButton2, RectButton3, RectButton4,
    RectButton11, RectButton22, RectButton33, RectButton44;
RECT Frame4 = { 30, FrameTop.top + 20 - 2, FrameTop.left + 600 +
    2, FrameTop.top + 20 + fontsize + 3+2 , };
RECT Frame5 = { FrameTop.left - 2, 130 - 2, FrameTop.left + 70+ 2,
```

```
     130 + fontsize + 3+2 };
RECT Frame_ReadNumber = { Winsize.left + 30, Frame1.top, Winsize.
    right - 30 / 2, Frame1.bottom };

//////////////////////
// 子ウィンドウのハンドル
//////////////////////
extern HWND Button1, Button2, Button3, Button4, Button11, Button22
    , Button33, Button44;
extern HWND hStaticProg, hStaticProg2, hNumberStatic,
    latedatacombo, hStaticUserDialog, hDialogUser, hStaticSentence;


HPEN hpen;
HBRUSH hbr;
HFONT hFont1;
HDC hdc;
//////////////////////////////////////
// プログレスバーで使用するグローバル変数
//////////////////////////////////////
int  _MAX, _MIN, _POS, _TEMP;              // プログレスバーで利用する
    変数
char bufferProg[10];                           // ステティックコント
    ロールに設定する文字列を格納
bool TempMAX;
bool TempResult = true;

// メモリデバイスコンテキスト
HDC  hDCMem, hDCMem4;
HGDIOBJ hDCMemOld, hDCMemOld4;

// 用 OwnerDrawButton
int selectedButtonID_1 = -1;
int selectedButtonID_2 = -1;
bool bBtn1 = false;
bool bBtn2 = false;
bool bBtn3 = false;
```

```
bool bBtn4 = false;
bool bBtn5 = false;
bool bBtn6 = false;
bool bBtn7 = false;
bool bBtn8 = false;

COLORREF color_Buttonbackground = RGB(73, 135, 242);
COLORREF color_iniButtonbackground = RGB(218,227,242);
char szText[50];
bool ButtonBool = false;
bool Clicked_1 = false;
bool Clicked_2 = false;
/*
    ----------------------------------------------------------------
    */
////////////////////
//関数の定義 WinMain
////////////////////
int APIENTRY WinMain(_In_ HINSTANCE hThisInst, _In_opt_ HINSTANCE
    hPrevInst, _In_ LPSTR lpszArgs, _In_ int nWinMode)
{
        HWND            hWnd = NULL;
        HWND        hWndNew = NULL;
        MSG                     uMsg;

        // クライアント領域のサイズ調整
        AdjustWindowRectEx(&Winsize, WS_OVERLAPPEDWINDOW, true, 0)
    ;

        // Show First Window
        CreateNewWindow(hWndNew, hThisInst, Winsize.right -
    Winsize.left, Winsize.bottom - Winsize.top, nWinMode);

        //インスタンスハンドルの保存
        hInst = hThisInst;
```

```
        //メッセージループの生成
        while (GetMessage(&uMsg, NULL, 0, 0)) {
                // GetMessage()関数の戻り値がになったら（0を受け取ったら）
    ループを抜ける WM_QUIT
                TranslateMessage(&uMsg);
                DispatchMessage(&uMsg);
        }
        return((int)uMsg.wParam);
}

////////////////////////////////////
// ウィンドウプロージャの定義
////////////////////////////////////
LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam,
    LPARAM lParam)
{
        short int IGrade;
        INITCOMMONCONTROLSEX initctrl;      //
    INITCOMMONCONTROLSEX 構造体変
    数
        PAINTSTRUCT ps;
        //HWND hwndButton;


        switch (uMsg){
                //終了処理
        case WM_DESTROY:
                // 書き込み済みの場合は書き込まない
                if (!FileWriteInfo) {
                        // ファイルのオープン
                        file_open(hWnd);
                        // ファイルへの書き込み
                        WriteFileFunc();
                        // ファイルのクローズ
                        CloseHandle(hfile);
                }
                // 後始末
```

```
            DeleteObject(hFont);
            DeleteObject(hFont1);
            DeleteObject(hFont2);
            DeleteObject(hFontTitle);
            DeleteObject(hpen);
            DeleteObject(hbr);
            SelectObject(hDCMem, hDCMemOld);
            SelectObject(hDCMem4, hDCMemOld4);
            DeleteDC(hDCMem);
            DeleteDC(hDCMem4);

            //DeleteObject(hBrush); // オーナードローボタンで使用

            // をメッセージキューにポスト WM_QUIT
            PostQuitMessage(0);
            return 0;

    case WM_CREATE:
            P_SNumber = SNumber;                                    // 読
み上げる文章の番号を格納する配列

            // INITCOMMONCONTROLSEX 構造体の初期化
            memset(&initctrl, 0, sizeof(initctrl));
            initctrl.dwSize = sizeof(initctrl); // 構造体のサイズ
INITCOMMONCONTROLSEX
            initctrl.dwICC = ICC_WIN95_CLASSES; // コモンコント
ロールクラスの指定（プログレスバー）
            InitCommonControlsEx(&initctrl);    // 使用するコント
ロールクラスの登録

            // 画面中央にウィンドウを移動
            DesktopCenterWindow(hWnd);

            // 初期化処理
            WM_CREATE_Func(hWnd, lParam);

            break;
```

```
        case WM_CTLCOLORSTATIC:
                return (SetCtlColor(wParam, lParam));

        case WM_TOUCH:
                OnTouch(hWnd, wParam, lParam);
                break;


        case WM_PAINT:
        {
                hdc = BeginPaint(hWnd, &ps);
                // メモリからへコピー DCDC
                BitBlt(ps.hdc,
                        ps.rcPaint.left, ps.rcPaint.top,
                        ps.rcPaint.right - ps.rcPaint.left, ps.
    rcPaint.bottom - ps.rcPaint.top,
                        hDCMem,
                        ps.rcPaint.left, ps.rcPaint.top,
                        SRCCOPY);

                // ビットマップ画像
                BitBlt(ps.hdc,
                        110, 90,
                        ps.rcPaint.right - ps.rcPaint.left, ps.
    rcPaint.bottom - ps.rcPaint.top,
                        hDCMem4,
                        0, 0, SRCCOPY);

                RECT rc;
                GetClientRect(hWnd, &rc);
                int x = rc.right - rc.left;
                int y = rc.bottom - rc.top;

                EndPaint(hWnd, &ps);

        }
```

```
            return 0;

    case WM_ERASEBKGND:
            // 何も処理しない画面のちらつき防止）(
            return 1;
            break;

    case WM_COMMAND:
            CommandFunc(hWnd, wParam, lParam, uMsg, hdc);
            break;

    case WM_DRAWITEM:
    {
            LPDRAWITEMSTRUCT pdis = (LPDRAWITEMSTRUCT)lParam;

            switch(pdis->CtlID)
            {
            case ID_BUTTON1:
                    OnDrawItem(bBtn1, pdis->hDC, pdis->rcItem,
pdis->hwndItem);
                    break;

            case ID_BUTTON2:
                    OnDrawItem(bBtn2, pdis->hDC, pdis->rcItem,
pdis->hwndItem);
                    break;

            case ID_BUTTON3:
                    OnDrawItem(bBtn3, pdis->hDC, pdis->rcItem,
pdis->hwndItem);
                    break;

            case ID_BUTTON4:
                    OnDrawItem(bBtn4, pdis->hDC, pdis->rcItem,
pdis->hwndItem);
                    break;
```

```
            case ID_BUTTON11:
                    OnDrawItem(bBtn5, pdis->hDC, pdis->rcItem,
pdis->hwndItem);
                    break;

            case ID_BUTTON22:
                    OnDrawItem(bBtn6, pdis->hDC, pdis->rcItem,
pdis->hwndItem);
                    break;

            case ID_BUTTON33:
                    OnDrawItem(bBtn7, pdis->hDC, pdis->rcItem,
pdis->hwndItem);
                    break;

            case ID_BUTTON44:
                    OnDrawItem(bBtn8, pdis->hDC, pdis->rcItem,
pdis->hwndItem);
                    break;

            default:
                    return(TRUE);
            }
    }
    return(TRUE);

    case WM_CHAR:
            switch (wParam) {
            case VK_ESCAPE:
                    PostQuitMessage(0);
                    break;

            case '1':
                    //PlaySound(TEXT("whistle.wav"), NULL,
SND_FILENAME);
```

```
                    // 入力キーの保存
                    IGrade = 1;
                    Grade[i] = '1';
                    i++;
                    // 実験回数の更新
                    NumberOfTimes++;
                    CountNumberFunc(NumberOfTimes);
                    break;

            case '2':
                    //PlaySound(TEXT("summernight.wav"), NULL,
SND_FILENAME);
                    // 入力キーの保存
                    IGrade = 2;
                    Grade[i] = '2';
                    i++;
                    //実験回数の更新と表示
                    NumberOfTimes++;
                    CountNumberFunc(NumberOfTimes);
                    break;

            case '3':
                    // 入力キーの保存
                    IGrade = 3;
                    Grade[i] = '3';
                    i++;
                    //実験回数の更新と表示
                    NumberOfTimes++;
                    CountNumberFunc(NumberOfTimes);
                    break;

            case '4':
                    // 入力キーの保存
                    IGrade = 4;
                    Grade[i] = '4';
                    i++;
```

```
                        // 実験回数の更新と表示
                        NumberOfTimes++;
                        CountNumberFunc(NumberOfTimes);
                        break;

                case 0x08: // バックスペース
                        NumberOfTimes--;                        //
入力回数の
更新
                        hdc = GetDC(hWnd);
                        CancelFunc(hWnd, hdc, NumberOfTimes);
                        SetNumber(P_SNumber, NumberOfTimes); // 文
章の番号の決定
                        DispNumberSentence(hdc);
  // 文章の番号の
表示
                        ReleaseDC(hWnd, hdc);
                        break;

                default:
                        MessageBox(hWnd, "1,␣2,␣3,␣いずれかのキーを押
してください。4", NULL, MB_OK | MB_ICONWARNING);
                        break;
                }
                return 0;

        case WM_KEYUP:
                // 何らかのキーが押されたときに発生するイベント
                break;

        case WM_KEYDOWN:
                switch (wParam) {
                        // を押すと Enter
                case VK_RETURN:
                        // 出力先ファイルが既に開かれていたとき、注意
                        //WarningFileOpen(hWnd);
                        if (MessageBox(hWnd, "終了しますか？", "終了
確認", MB_YESNO | MB_ICONQUESTION) == IDYES) {
```

```
                                DestroyWindow(hWnd);
                        }
                        return 0;
                }
                break;

        case WM_CLOSE:
                // 出力先ファイルが既に開かれていたとき、注意
                //WarningFileOpen(hWnd);
                if (MessageBox(hWnd, (LPCSTR)"終了しますか？
    ", (LPCSTR)"終了確認", MB_YESNO | MB_ICONQUESTION) == IDYES) {
                        DestroyWindow(hWnd);
                }
                break;

        default:
                // ボタンの有効・無効の切り替え
                ButtonEnableFunc();
                return (DefWindowProc(hWnd, uMsg, wParam, lParam))
    ;

        }
        return(0);
}


////////////////////
// オーナーボタンの描画
////////////////////
bool OnDrawItem(bool bBtn, HDC hDC, RECT rcItem, HWND hwndItem) {

        HBRUSH hBrush_ini = (HBRUSH)CreateSolidBrush(
    color_iniButtonbackground);
        HBRUSH hBrush = CreateSolidBrush(RGB(73, 135, 242));

        FillRect(hDC, &rcItem, hBrush_ini);
        DrawEdge(hDC, &rcItem, EDGE_RAISED, BF_RECT);
```

```
        GetWindowText(hwndItem, szText, 50);
        // Draw Text on Button
        SetTextColor(hDC, RGB(0, 0, 0));
        SetBkColor(hDC, color_iniButtonbackground);
        DrawText(hDC, szText, -1, &rcItem, DT_CENTER | DT_VCENTER
    | DT_SINGLELINE);

        if (bBtn) {
                FillRect(hDC, &rcItem, hBrush);
                DrawEdge(hDC, &rcItem, EDGE_SUNKEN, BF_RECT);
                GetWindowText(hwndItem, szText, 50);
                // Draw Text on Button
                SetTextColor(hDC, RGB(255, 255, 255));
                SetBkColor(hDC, color_Buttonbackground);
                DrawText(hDC, szText, -1, &rcItem, DT_CENTER |
    DT_VCENTER | DT_SINGLELINE);
        }

        // clean object
        DeleteObject(hBrush_ini);
        DeleteObject(hBrush);

        return true;
}

//////////////////////
// ボタンの有効化・無効化
//////////////////////
bool ButtonEnableFunc() {

        // ボタンの有効化
        if (!boolstart) {
                 //一度有効化したら終了まで有効状態
                if (strMenLady) {
                        EnableWindow(hReserch, TRUE);
                        boolstart = true;
```

```
                }
        }
        else if (!ZeroOrNot) {
                if (NumberOfTimes > 0) {
                        EnableWindow(hCancel, TRUE);
                        // 一度有効化したら次に無効状態になるまで有効化
    しない
                        ZeroOrNot = true;
                }
        }
        else if ((NumberOfTimes == 0)) {
                // になったら無効化する 0
                EnableWindow(hCancel, FALSE);
                ZeroOrNot = false;
        }
        else if(NumberOfTimes == _MAX * 2){
                // 終了後にボタンを無効化
                EnableWindow(hCancel, FALSE);
                EnableWindow(Button1, FALSE);
                EnableWindow(Button2, FALSE);
                EnableWindow(Button3, FALSE);
                EnableWindow(Button4, FALSE);
                EnableWindow(Button11, FALSE);
                EnableWindow(Button22, FALSE);
                EnableWindow(Button33, FALSE);
                EnableWindow(Button44, FALSE);
                EnableWindow(hReserch, FALSE);
                EnableWindow(hNext, FALSE);
        }
        return true;
}

//////////////////////////////
// メッセージの定義 WM_CREATE
//////////////////////////////
bool WM_CREATE_Func(HWND hWnd, LPARAM lParam) {
        // グローバル変数の初期化
```

```
        NumberOfTimes = 0;                          // 実験回数を保存する
    変数
        i = 0;                                      // 評価結果を保存する配
    列のインデックス
        // 子ウィンドウ作成
        //ComboboxFunc(hWnd, lParam);              // 遅延時間設定用
        ReserchStartFunc(hWnd, lParam);             // 実験開始ボタン
        CreateOwnerDrawButton(hWnd, lParam);        // 評価結果入力用ボタン
        CreateProgressBar(hWnd, 40, lParam);        // プログレスバーの作成
        //プログレスバーの設定
        ChangeProgBarMAX(intCurrentIndex);
        // 読み上げる文章の順番を決定
        Getarray(P_SNumber, 10);
        // メモリデバイスコンテキスト描画の事前準備
        GetWindowMemDCFunc(hWnd);
        // メモリデバイスコンテキストへの描画
        PaintFunc(hWnd);
        // フォントの適用
        FontFunc();
        return true;

}

////////////////////////////////////////////////
// メモリデバイスコンテキスト描画の事前準備
////////////////////////////////////////////////
bool GetWindowMemDCFunc(HWND hWnd) {

        HDC hDC;
        HBITMAP hBitmap, hBitmap4;
        BITMAP bitmap4{};

        //デバイスコンテキストの取得
        hDC = GetDC(hWnd);
        // メモリデバイスコンテキストの取得
        hDCMem = CreateCompatibleDC(hDC);
        hDCMem4 = CreateCompatibleDC(hDC);
```

```
        // ビットマップハンドルの取得
        hBitmap = CreateCompatibleBitmap(hDC, Winsize.right,
    Winsize.bottom);
        hBitmap4 = LoadBitmap(GetModuleHandle(NULL),//(HINSTANCE)
    GetWindowLongPtr(hWnd, GWL_HINSTANCE),
                MAKEINTRESOURCE(IDB_BITMAP3));
        // ビットマップ画像のサイズ取得
        GetObject(hBitmap4, sizeof(BITMAP), &bitmap4);
        // デバイスコンテキストの解放（メモリデバイスコンテキストを取得するためだけに使うからもう不要）
        ReleaseDC(hWnd, hDC);
        // メモリにビットマップを割りつけ DC
        hDCMemOld = SelectObject(hDCMem, hBitmap);
        hDCMemOld4 = SelectObject(hDCMem4, hBitmap4);
        // ビットマップの削除（ビットマップはメモリデバイスコンテキストの情報を設定するためだけに使うからもう不要）
        DeleteObject(hBitmap);
        DeleteObject(hBitmap4);

        return true;
}

////////////////////////////////////////
// メモリデバイスコンテキストへの描画
////////////////////////////////////////
bool PaintFunc(HWND hWnd) {

        HRGN hRgn;// , hRgn1, hRgn2, hRgn3;

        // フォントの定義
        hFont1 = CreateFont(
                fontsize2, 0, 0, 0,
                FW_MEDIUM,
                false, false, false,
                SHIFTJIS_CHARSET, OUT_DEFAULT_PRECIS,
                CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                (VARIABLE_PITCH | FF_DONTCARE), _T("メイリオ"));
```

```
    //
    // ウィンドウの背景の設定
    hRgn = CreateRectRgn(Winsize.left, Winsize.top, Winsize.
right, Winsize.bottom);
    FillRgn(hDCMem, hRgn, (HBRUSH)CreateSolidBrush(RGB
(255,255,255)));
    FillRect(hDCMem, &FrameProgTop, (HBRUSH)CreateSolidBrush(
RGB(70, 130, 180)));
    FillRect(hDCMem, &FrameBottom, (HBRUSH)CreateSolidBrush(
RGB(70, 130, 180)));
    FillRect(hDCMem, &FrameCenter, BackGround_clear);
    /*hRgn1 = CreateRectRgn(FrameBottom.left − 30, FrameBottom.
bottom − 64, FrameBottom.right + 30, FrameBottom.bottom);
 hRgn2 = CreateRectRgn(0, 0, 30, Winsize.bottom − (FrameBottom.
bottom − FrameBottom.top + 5));
 hRgn3 = CreateRectRgn(Winsize.right − 30, 0, Winsize.right, Winsize.
bottom − (FrameBottom.bottom − FrameBottom.top + 5));*/
    //FillRgn(hDCMem, hRgn1, (HBRUSH)CreateSolidBrush(RGB
(155,169,194)));
    /*FillRgn(hDCMem, hRgn2, (HBRUSH)CreateSolidBrush(RGB(22,
33, 77)));
 FillRgn(hDCMem, hRgn3, (HBRUSH)CreateSolidBrush(RGB(22, 33, 77)
));*/
    DeleteObject(hRgn);
    //DeleteObject(hRgn1);
    /*DeleteObject(hRgn2);
 DeleteObject(hRgn3);*/
    // 文字列の背景の設定
    SetBkColor(hDCMem, TextBackground);
    // フォントの適用
    SelectObject(hDCMem, hFont1);

    // 枠線
    DrawEdge(hdc, &FrameBottom, EDGE_RAISED, BF_TOP);

    return true;
```

```
}
//////////////////////////////
// メッセージの定義 WM_COMMAND
//////////////////////////////
bool CommandFunc(HWND hWnd, WPARAM wParam, LPARAM lParam, UINT
    uMsg, HDC hdc) {

        int MsgResult;

        switch (LOWORD(wParam)){
        case ID_BUTTON1:
                GradeS[i] = 1;
                Clicked_1 = true;
                bBtn1 = true;
                bBtn2 = false;
                bBtn3 = false;
                bBtn4 = false;
                InvalidateRect(hWnd, &FrameCenter, FALSE);
                //SendMessage(hWnd, WM_DRAWITEM, (WPARAM)0, (
    LPARAM)&dis);
                break;

        case ID_BUTTON2:
                GradeS[i] = 2;
                Clicked_1 = true;
                bBtn1 = false;
                bBtn2 = true;
                bBtn3 = false;
                bBtn4 = false;
                InvalidateRect(hWnd, &FrameCenter, TRUE);
                break;

        case ID_BUTTON3:
                GradeS[i] = 3;
                Clicked_1 = true;
                bBtn1 = false;
```

```
            bBtn2 = false;
            bBtn3 = true;
            bBtn4 = false;
            InvalidateRect(hWnd, &FrameCenter, TRUE);
            break;

        case ID_BUTTON4:
            GradeS[i] = 4;
            Clicked_1 = true;
            bBtn1 = false;
            bBtn2 = false;
            bBtn3 = false;
            bBtn4 = true;
            InvalidateRect(hWnd, &FrameCenter, TRUE);
            break;

        case ID_BUTTON11:
            GradeL[i] = 1;
            Clicked_2 = true;
            bBtn5 = true;
            bBtn6 = false;
            bBtn7 = false;
            bBtn8 = false;
            InvalidateRect(hWnd, &FrameCenter, TRUE);
            break;

        case ID_BUTTON22:
            GradeL[i] = 2;
            Clicked_2 = true;
            bBtn5 = false;
            bBtn6 = true;
            bBtn7 = false;
            bBtn8 = false;
            InvalidateRect(hWnd, &FrameCenter, TRUE);
            break;
```

```
        case ID_BUTTON33:
                GradeL[i] = 3;
                Clicked_2 = true;
                bBtn5 = false;
                bBtn6 = false;
                bBtn7 = true;
                bBtn8 = false;
                InvalidateRect(hWnd, &FrameCenter, TRUE);
                break;

        case ID_BUTTON44:
                GradeL[i] = 4;
                Clicked_2 = true;
                bBtn5 = false;
                bBtn6 = false;
                bBtn7 = false;
                bBtn8 = true;
                InvalidateRect(hWnd, &FrameCenter, TRUE);
                break;

        break;

        case ID_LATEINI:
                LateIniFunc(hWnd, wParam);
                break;


        case ID_ReserchStart:
                if (HIWORD(wParam) == BN_CLICKED) {
                        SetFocus(hWnd);    // 親ウィンドウへフォーカス
を移動
                        // ボタンの有効化
                        EnableWindow(hNext, TRUE);
                        EnableWindow(Button1, TRUE);
                        EnableWindow(Button2, TRUE);
                        EnableWindow(Button3, TRUE);
                        EnableWindow(Button4, TRUE);
```

```
                        EnableWindow(Button11, TRUE);
                        EnableWindow(Button22, TRUE);
                        EnableWindow(Button33, TRUE);
                        EnableWindow(Button44, TRUE);
                        //  遅延時間設定用コンボボックスの無効化
                        EnableWindow(latedatacombo, false);
                        MessageBox(hWnd, _T("調査を開始します。
"), _T("開始確認"), MB_OK);
                        // 読み上げる文章の番号を計算し、表示
                        SetNumber(P_SNumber, NumberOfTimes);
                        DispNumberSentence(hDCMem);
                }

                break;

        case ID_NEXTBUTTON:
                if (HIWORD(wParam) == BN_CLICKED) {
                        if (TempMAX) {
                                MsgResult = MessageBox(hWnd, _T("こ
れで調査は終了です。終了ボタンを押しウィンドウを閉じてください。
"), _T("調査終了確認"), MB_OKCANCEL);
                                if ( MsgResult == IDCANCEL) {
                                        break;
                                }
                                else if (MsgResult == IDOK){
                                        hdc = GetDC(hWnd);
                                        NextButtonFunc(hWnd, hdc);
                                        // プログレスバーを更新する
                                        _TEMP = SendMessage(hProg,
 PBM_STEPIT, 0, 0);
                                        //現在値を取得
                                        _POS = SendMessage(hProg,
PBM_GETPOS, 0, 0);
                                        //最大値を超えない場合はステッ
プテキストに現在値を設定 1&
                                        if (_POS <= _MAX) {
                                                wsprintf(
bufferProg, "%d␣/␣%d", _POS, _MAX);
```

```
                                                SendMessage(
hStaticProg, WM_SETTEXT, 0, (LPARAM)bufferProg);
                                                if (_POS == _MAX)
{
                                                        TempMAX =
true;
                                                }
                                        }
                                        const char* bufferSentence
 = _T("ありがとうございました。
");
                                        SendMessage(
hStaticSentence, WM_SETTEXT, 0, (LPARAM)bufferSentence);
                                        ////////////////////////
                                        // ファイルに結果を出力
                                        ////////////////////////
                                        file_open(hWnd);
// テキストファイルのオー
プン
                                        WriteFileFunc();      //
ファイルへの書き
込み
                                        CloseHandle(hfile);   //
ファイルのク
ローズ
                                        FileWriteInfo = true; //
ファイルに書き込み済みであることを知らせる（で使用）
WM_DESTROY
                                        break;

                                }

                        }
                        else {
                                hdc = GetDC(hWnd);
                                NextButtonFunc(hWnd, hdc);
                                if (!TempResult) {
                                        // 正しく回答できていないとき
は何もしない
```

```
                                        break;
                                }
                                // プログレスバーを更新する
                                _TEMP = SendMessage(hProg,
PBM_STEPIT, 0, 0);
                                //現在値を取得
                                _POS = SendMessage(hProg,
PBM_GETPOS, 0, 0);
                                //最大値を超えない場合はステップテキス
トに現在値を設定 1&
                                if (_POS <= _MAX) {
                                        wsprintf(bufferProg, _T("%
d␣/␣%d"), _POS, _MAX);
                                        SendMessage(hStaticProg,
WM_SETTEXT, 0, (LPARAM)bufferProg);
                                        if (_POS == _MAX) {
                                                TempMAX = true;
                                        }
                                }
                                ReleaseDC(hWnd, hdc);
                                SetFocus(hWnd);
                        }
                }
                break;


    case ID_CANCEL:
                if (HIWORD(wParam) == BN_CLICKED) {
                        hdc = GetDC(hWnd);
                        NumberOfTimes = CancelFunc(hWnd, hdc,
NumberOfTimes);
                        SetNumber(P_SNumber, NumberOfTimes);
                        DispNumberSentence(hdc);
                        ReleaseDC(hWnd, hdc);
                        SetFocus(hWnd);
                }
```

```
            break;

// メニューバー
    case ID_MENU_MENU1:
            //Menu1
            SelectFile(hWnd);
            break;

    case ID_MENU_MENU2:
            //Menu2
            MessageBox(hWnd, _T("Menu2"), _T("Message␣Box"),
MB_OK);
            break;

    case ID_ENABLE_ALL_BUTTON:
            // 全ボタンの有効化
            EnableAllButton();
            break;

    case ID_GOINI:
            if (MessageBox(hWnd, _T("アプリケーションを起動時の状
態に戻します。\r\続行しますか？ n"), _T("確認
"), MB_OKCANCEL | MB_ICONQUESTION) == IDOK) {
                    API_GOINI(hWnd);
                    break;
            }
            break;

    case ID_ENABLE_LATEDATACOMBO:
            EnableWindow(latedatacombo, TRUE);
            break;
    /*case ID_BUTTON1:
      if (BST_CHECKED == SendMessage(Button1, BM_GETCHECK,
0, 0)) {
            GradeS[i] = 1;
      }
      break;
```

```
  case ID_BUTTON2:
        if (BST_CHECKED == SendMessage(Button2, BM_GETCHECK,
0, 0)) {
            GradeS[i] = 2;
     }
        break;
  case ID_BUTTON3:
        if (BST_CHECKED == SendMessage(Button3, BM_GETCHECK,
0, 0)) {
              GradeS[i] = 3;
        }
        break;
  case ID_BUTTON4:
        if (BST_CHECKED == SendMessage(Button4, BM_GETCHECK,
0, 0)) {
              GradeS[i] = 4;
        }
        break;
  case ID_BUTTON11:
        if (BST_CHECKED == SendMessage(Button11, BM_GETCHECK
, 0, 0)) {
              GradeL[i] = 1;
        }
        break;
  case ID_BUTTON22:
        if (BST_CHECKED == SendMessage(Button22, BM_GETCHECK
, 0, 0)) {
              GradeL[i] = 2;
        }
        break;
  case ID_BUTTON33:
        if (BST_CHECKED == SendMessage(Button33, BM_GETCHECK
, 0, 0)) {
              GradeL[i] = 3;
        }
        break;
```

```
        case ID_BUTTON44:
            if (BST_CHECKED == SendMessage(Button44, BM_GETCHECK
    , 0, 0)) {
                GradeL[i] = 4;
            }
            break;*/

        case ID_EXIT:
                        if (MessageBox(hWnd, _T("リセットしますか？
    "), _T("リセット確認"), MB_YESNO | MB_ICONQUESTION) == IDYES) {
                            // メッセージを送信し、ウィンドウの初期
    化 WM_CREATE
                            ResetFunc(hWnd, hInst, szWinName,
    Winsize.right - Winsize.left, Winsize.bottom - Winsize.top);
                        }
                break;

        case ID_QUIT:
                if (HIWORD(wParam) == BN_CLICKED) {
                        // 出力先ファイルが既に開かれていたとき、注意喚起
                        //WarningFileOpen(hWnd);
                        if (MessageBox(hWnd, _T("ウィンドウを閉じま
    す。\r\続行しますか？ n"), _T("終了確認
    "), MB_YESNO | MB_ICONQUESTION) == IDYES) {
                            // メッセージの送信 WM_DESTORY
                            DestroyWindow(hWnd);
                        }
                }
                break;

        default:
                break;
        }
        return true;
}

//////////////////////////////////////////
// コンボボックスにフォーカスが来たときの処理
```

```
////////////////////////////////////////
bool LateIniFunc(HWND hWnd, WPARAM wParam) {

    if (HIWORD(wParam) == CBN_SELCHANGE) {
        //コンボボックスで現在選択されている項目のインデックスを
取得
        intCurrentIndex = SendMessage(GetDlgItem(hWnd, (
int)ID_LATEINI), CB_GETCURSEL, 0, 0);

        // コンボボックスの一覧内の文字列の長さを取得
        int intTxtLen = SendMessage(GetDlgItem(hWnd, (int)
ID_LATEINI), CB_GETLBTEXTLEN, intCurrentIndex, 0);

        if (intTxtLen != CB_ERR) {
            char* pszBuf = new char[intTxtLen + 1];
            // コンボボックスの一覧から選択した項目の文字列
を取得
            if (SendMessage(GetDlgItem(hWnd, (int)
ID_LATEINI), CB_GETLBTEXT, intCurrentIndex, (LPARAM)pszBuf) !=
CB_ERR) {
                char Path[MAX_PATH + 1];
                char settingpath[MAX_PATH + 1]{};
                settingpath[0] = '\0';
                if (0 != GetModuleFileName(NULL,
Path, MAX_PATH)) {
                    char drive[MAX_PATH + 1],
dir[MAX_PATH + 1], fname[MAX_PATH + 1], ext[MAX_PATH + 1];
                    _splitpath_s(Path, drive,
sizeof(drive), dir, sizeof(dir),
                        fname, sizeof(
fname), ext, sizeof(ext));
                    _stprintf_s(settingpath,
MAX_PATH + 1, _T("%s%ssetting.ini"), drive, dir);
                }
                // ファイルから選択したキーの遅延時間を
取得しに保存 inilatedata
                GetPrivateProfileString(pszBuf, _T
```

```
    ("data"), _T("error"), latedata, sizeof(latedata), settingpath)
    ;
                        }
                        // 遅延グループ名をウィンドウタイトルを保持する
    型配列にコピー char
                        strcpy_s(WindowTitleText, sizeof(
    WindowTitleText), pszBuf);
                        delete[] pszBuf;
                }

                // 親ウィンドウへフォーカスを移動
                SetFocus(hWnd);
        }

        return true;
}

////////////////////////////////////////////////////////////////
// プログレスバーの最大値を遅延時間設定で選択された項目によって決定
////////////////////////////////////////////////////////////////
bool ChangeProgBarMAX(int intCurrentIndex) {

        char tempProg[10];

        if (intCurrentIndex <= 4) {
                // プログレスバーの最大値をに指定 9
                _MAX = CONBOMAX9;
                SendMessage(hProg, PBM_SETRANGE, (WPARAM)0,
    MAKELPARAM(0, _MAX));
                // プログレスバー横のステティックコントロールを更新
                sprintf_s(tempProg, sizeof(tempProg), "1␣/␣%d",
    _MAX);
                SendMessage(hStaticProg, WM_SETTEXT, 0, (LPARAM)
    tempProg);
        }
        else {
                // プログレスバーの最大値をに指定 8
```

```
                _MAX = CONBOMAX8;
                SendMessage(hProg, PBM_SETRANGE, (WPARAM)0,
    MAKELPARAM(0, _MAX));
                // プログレスバー横のステティックコントロールを更新
                sprintf_s(tempProg, sizeof(tempProg), "1␣/␣%d",
    _MAX);
                SendMessage(hStaticProg, WM_SETTEXT, 0, (LPARAM)
    tempProg);
            }

            return true;
    }

    ////////////////
    // 評価結果を保存
    ////////////////
    bool ResultGrade(char Grade) {

            Result[IndexResult] = Grade;
            IndexResult += 1;

            return true;
    }

    ///////////////
    //実験回数の計算
    ///////////////
    char CountNumberFunc(unsigned int NumberOfTimes) {

            short int temp2;

            /*if (NumberOfTimes % 2 == 0) { temp2 = NumberOfTimes / 2; }
          else                  { temp2 = (NumberOfTimes + 1) / 2; }*/
            temp2 = (NumberOfTimes + 2) / 2;
            sprintf_s(temp1, sizeof(temp1), "%回目 d", temp2);
```

```
        return char(temp1);
}

////////////////////////////////
// 乱数生成　最小値: low, 最大値: high
////////////////////////////////
int GetRandom(unsigned int low, unsigned int high) {
        random_device rd;
        default_random_engine eng(rd());
        uniform_int_distribution<int> distr(low, high);
        return distr(eng);
}
//////////////////
// 配列のシャッフル
//////////////////
int* ShuffleFunc(int* P_SNumber, unsigned int length) {
        for (size_t i = 0; i < length; i++){
                int r = GetRandom(i, length - 1);
                int tmp = P_SNumber[i];
                P_SNumber[i] = P_SNumber[r];
                P_SNumber[r] = tmp;
        }
        return P_SNumber;
}

////////////////////////////////////
// 読み上げる文章の番号を文字列に変換
////////////////////////////////////
char* SetNumber(int* P_SNumber, short int NumberOfTimes) {

        int tempSetNumber;

        // 実験回数の取り込み
        /*if (NumberOfTimes % 2 == 0) { temp1 = NumberOfTimes / 2; }
      else { temp1 = (NumberOfTimes + 1) / 2; }*/
```

```
        tempSetNumber = NumberOfTimes / 2;

        // 型を文字列に変換 int
        sprintf_s(temp3, sizeof(temp3), "%番の文章を読み上げてくださ
    い。d", P_SNumber[tempSetNumber]);

        return temp3;
}

////////////////////////////
// 読み上げる文章の番号を取得
////////////////////////////
int* Getarray(int* P_SNumber, unsigned int length) {
                // 配列要素の初期化
                for (size_t i = 0; i < length; i++) {
                        P_SNumber[i] = i + 1;
                }
        // 配列のシャッフル
        P_SNumber = ShuffleFunc(P_SNumber, length);

        return P_SNumber;
}

////////////////////////////////////
// 「戻る」ボタンが押下された時の処理
////////////////////////////////////
unsigned int CancelFunc(HWND hWnd, HDC hdc, unsigned int
    NumberOfTimes) {

        if (i < 1) {
                MessageBox(hWnd, "取り消す対象がありません。
    ", NULL, MB_OK | MB_ICONWARNING);
        }
        else {
                // ボタンを元に戻す
                bBtn1 = false;
                bBtn2 = false;
```

```
                bBtn3 = false;
                bBtn4 = false;
                bBtn5 = false;
                bBtn6 = false;
                bBtn7 = false;
                bBtn8 = false;
                Clicked_1 = false;
                Clicked_2 = false;

                InvalidateRect(hWnd, &FrameCenter, TRUE);

                i--;                                    // 評価結果保存用配
列のインデックス更新
                NumberOfTimes = NumberOfTimes - 2;
                CountNumberFunc(NumberOfTimes);  // 実験回数を文字列
に変換
                //SendMessage(hNumberStatic, WM_SETTEXT, 0, (
LPARAM)temp1);
                SetBkColor(hdc, TextBackground); //テキストの背景色
                SetTextColor(hdc, RGB(30, 30, 30));

                // プログレスバーとステティックコントロールの更新 //
                //現在位置を取得
                _POS = SendMessage(hProg, PBM_GETPOS, 0, 0);
                if (_POS == _MAX) {
                        TempMAX = false;
                }
                // 現在位置を一つずらす
                SendMessage(hProg, PBM_SETPOS, (WPARAM)_POS - 1,
0);
                // 現在値を取得
                _POS = SendMessage(hProg, PBM_GETPOS, 0, 0);
                // ステティックコントロール
                wsprintf(bufferProg, "%d␣/␣%d", _POS, _MAX);
                SendMessage(hStaticProg, WM_SETTEXT, 0, (LPARAM)
bufferProg);
```

```
        }
        return unsigned int(NumberOfTimes);
}

////////////////////////////////////
// 「次へ」ボタンが押下された時の処理
////////////////////////////////////
bool NextButtonFunc(HWND hWnd, HDC hdc) {

        // ボタンを押していない場合は次に進めない
        if (Clicked_1 == false || Clicked_2 == false) {
                MessageBox(hWnd, _T("正しく回答できていません。
    "), _T("エラー"), MB_OK | MB_ICONWARNING);
                TempResult = false;
                return true;
        }

        // 実験回数の更新と表示
        NumberOfTimes += 2;
        i++;                                    // 配列のインデックス更新
        CountNumberFunc(NumberOfTimes); // 実験回数を文字列に変換
        // 実験回数を表示
        //SendMessage(hNumberStatic, WM_SETTEXT, 0, (LPARAM)
    temp1);
        SetBkColor(hdc, TextBackground); // テキストの背景色
        // 読み上げる文章の番号を表示
        SetNumber(P_SNumber, NumberOfTimes);
        DispNumberSentence(hdc);
        // テキストの色を決定
        SetTextColor(hdc, RGB(30, 30, 30));
        //TextOut(hdc, NumberTest_i[0], NumberTest_i[1], temp1, lstrlen(
    temp1));

        TempResult = true; // ここまできたということは正しく回答できてい
    るということ

    // ボタンを元に戻す
```

```
        bBtn1 = false;
        bBtn2 = false;
        bBtn3 = false;
        bBtn4 = false;
        bBtn5 = false;
        bBtn6 = false;
        bBtn7 = false;
        bBtn8 = false;
        Clicked_1 = false;
        Clicked_2 = false;

        // 再描画要求
        InvalidateRect(hWnd, &FrameCenter, TRUE);

        return true;
}

////////////////////////////////////
// 読み上げる文章の番号を表示
////////////////////////////////////
bool DispNumberSentence(HDC hdc) {
        SendMessage(hStaticSentence, WM_SETTEXT, 0, (LPARAM)temp3)
    ;
        return true;
}

//////////////////////////
// メッセージの定義 WM_CTLCOLORSTATIC
//////////////////////////
long SetCtlColor(WPARAM wParam, LPARAM lParam) {

        int i = GetWindowLong((HWND)lParam, GWL_ID);
        if (i == 0) return -1;
        else {
                if((i == ID_STATICPROG) || (i == ID_STATICPROG2)){
                        SetBkMode((HDC)wParam, TRANSPARENT);
```

```
                        SetTextColor((HDC)wParam, RGB(255, 255,
    255));
                        return (long)CreateSolidBrush(RGB(70, 130,
     180));
                }
                else if (i == ID_STATICSENTENCE) {
                        SetBkMode((HDC)wParam, TRANSPARENT);
                        return (long)CreateSolidBrush(RGB(255,
    255, 255));
                }
                else{
                        SetBkMode((HDC)wParam, TRANSPARENT);
                        return (long)CreateSolidBrush(
    TextBackground);
                }
        }
}

////////////////////////////////////
// 指定した構造体変数の枠線の描画（要改善）RECT
////////////////////////////////////
bool MyDrawEdge(HDC hdc, RECT rc) {

        SelectObject(hdc, hpen);
        SelectObject(hdc, hbr);
        RoundRect(hdc, rc.left, rc.top, rc.right, rc.bottom, 10,
    10);


        return true;
}

////////////////////////////
// リセットボタン押下時の処理
////////////////////////////
bool ResetFunc(HWND hWnd, HINSTANCE hThisInst, const char
```

```
    szWinName[], const int nWidth, const int nHeight) {

        return true;
}

////////////////
// 全ボタンの有効化
////////////////
bool EnableAllButton() {

        //  ボタンの有効化
        EnableWindow(Button1, TRUE);
        EnableWindow(Button2, TRUE);
        EnableWindow(Button3, TRUE);
        EnableWindow(Button4, TRUE);
        EnableWindow(Button11, TRUE);
        EnableWindow(Button22, TRUE);
        EnableWindow(Button33, TRUE);
        EnableWindow(Button44, TRUE);
        EnableWindow(hReserch, TRUE);
        EnableWindow(hNext, TRUE);
        EnableWindow(hCancel, TRUE);

        return true;
}

//////////////////////////
// 再起動
//////////////////////////
bool API_GOINI(HWND hWnd) {

        TCHAR szPath[MAX_PATH];
        GetModuleFileName(NULL, szPath, MAX_PATH);
        STARTUPINFO si;
        PROCESS_INFORMATION pi;
```

```
        memset(&si, 0, sizeof(si));
        si.cb = sizeof(si);
        memset(&pi, 0, sizeof(pi));

        if (CreateProcess(szPath, NULL, NULL, NULL, FALSE, 0, NULL
    , NULL, &si, &pi)) {
                CloseHandle(pi.hProcess);
                CloseHandle(pi.hThread);
                DestroyWindow(hWnd); // メッセージを送信
    WM_DESTROY
        }

        return true;
}

////////////////////////////////////////////////
// メインウィンドウをデスクトップの画面中央に移動
////////////////////////////////////////////////
BOOL DesktopCenterWindow(HWND hWnd)
{
        RECT    rc1{};          // デスクトップ領域
        RECT    rc2;          // ウインドウ領域
        INT     cx, cy;       // ウインドウ位置
        INT     sx, sy;       // ウインドウサイズ

        // サイズの取得
        GetMonitorRect(&rc1);                                  // デスク
    トップのサイズ
        GetWindowRect(hWnd, &rc2);                               // ウ
    インドウのサイズ
        sx = (rc2.right - rc2.left);                              //
    ウインドウの
    横幅
        sy = (rc2.bottom - rc2.top);                              //
    ウインドウの
    高さ
        cx = (((rc1.right - rc1.left) - sx) / 2 + rc1.left);    //
    横方向の中央座
    標軸
```

```
        cy = (((rc1.bottom - rc1.top) - sy) / 2 + rc1.top);     //
    縦方向の中央座
    標軸
        // 画面中央に移動
        return SetWindowPos(hWnd, NULL, cx, cy, 0, 0, (SWP_NOSIZE
    | SWP_NOZORDER | SWP_NOOWNERZORDER));
}

////////////////
// 入力結果の分別
////////////////
bool DivideGradeFunc() {

        size_t i = 0;

        for (size_t j = 0; j < (int)STRLEN / 2; j += 2) {
                GradeSpeak[j] = Grade[i];
                GradeSpeak[j + 1] = ',';
                GradeLate[j] = Grade[i + 1];
                GradeLate[j + 1] = ',';
                i += 2;
        }
        return true;
}
```

ソースコード B.2: window.cpp

```
  ////////////////////////////////////////////
///window.cpp: 親・子ウィンドウ作成用関数の定義
////////////////////////////////////////////

#include<windows.h>
#include<tchar.h>
#include<stdio.h>
#include <commctrl.h>
#include <sstream>
/////////////////////
```

```
// 自作ヘッダファイル
//////////////////////
#include "window.h"
#include "main.h"
#include"resource.h"
#define STRLEN 256

using namespace std;
//////////////////////////
// 構造体変数の初期化 RECT
//////////////////////////
RECT Winsize = {0,0,1100,750};
RECT FrameMain = {30, 30, Winsize.right - 30, Winsize.bottom -
    30};
RECT FrameUserInfo = {0, 30, Winsize.right, 175};
RECT FrameComment = {0, 175, Winsize.right, 280};
RECT FrameProgTop = { Winsize.left, Winsize.top, Winsize.right,
    70};
RECT FrameProg = {0, 280, Winsize.right, 385};
RECT Frame1 = { 30, 310, Winsize.right - 30, 380 };
RECT Frame2 = { 30, 420, Winsize.right / 2  - 10, 670 };    // 読み
    上げるときにしゃべりにくくないか
RECT Frame3 = { Winsize.right / 2 + 10, 420, Winsize.right - 30,
    670 }; // 遅れが気にな
    るか
RECT FrameTop = { 30, 30, Winsize.right - 30, 300 };
RECT FrameS = { (FrameTop.left + 100), 100, (FrameTop.left + 100)
    + 300, 100 + 65 };
RECT FrameBottom = { 0, Winsize.bottom - 74, Winsize.right,
    Winsize.bottom };  // 一番下のボ
    タン
RECT FrameCenter = { 0, 150, Winsize.right, FrameBottom.top - 5};
RECT rectS, FrameNumber;
RECT RectButton1, RectButton2, RectButton3, RectButton4,
    RectButton11, RectButton22, RectButton33, RectButton44;

///////////////
```

```
// 背景色の設定
///////////////
const HBRUSH BackGround_clear = CreateSolidBrush(RGB(235, 235,
    235));
//const HBRUSH BackGround_clear = CreateSolidBrush(LTGRAY_BRUSH);

// ////////////////////////
// 子ウィンドウハンドルの宣言
////////////////////////////
HWND CheckMen, CheckLady;
HWND hEditName, hEditOld;      // 名前入力エディットボックス用ハンドル
      // 年齢入力エディットボックス用ハンドル
HWND hDialogUser, hStaticDialogUser, hStaticTitle;
HWND hQuit, hCancel, hReserch, hNext;
HWND hNumber, hNumberStatic;
HWND hSpeak, hLate;
HWND Button1, Button2, Button3, Button4, Button11, Button22,
    Button33, Button44;
HWND hProg, hStaticProg, hStaticProg2, latedatacombo,
    hStaticSentence;

///////////////
// フォント設定
///////////////
HFONT hFont, hFont2, hFontTitle;
int fontsize = 30;
int fontsize2 = 32;
int fontsize3 = 42;

//////////////////////
// その他グローバル変数
//////////////////////
short int i;                        // 評価結果保存用配列のインデックス
    番号
char temp1[STRLEN];
extern int _MAX;    // プログレスバーで使用
char WindowTitleText[32];  // ウィンドウのテキスト
```

```
int intCurrentIndex; // 遅延時間設定用コンボボックスの項目番号

//////////////////
// 親ウィンドウ作成
//////////////////
HWND CreateParentWindow(HWND hWnd, HINSTANCE hThisInst, int
    nWinMode, const char szWinName[], const int nWidth, const int
    nHeight)
{
        WNDCLASSEX      wcl{};
        //ウィンドウクラスの定義
        wcl.cbSize = sizeof(WNDCLASSEX);
            //構造体のサイズ
    WNDCLASSEX
        wcl.style = 0;
                            //ウィンドウクラススタ
    イル
        wcl.lpfnWndProc = WndProc;
            //ウィンドウ関
    数
        wcl.cbClsExtra = 0;
                            //ウィンドウクラスのエキス
    トラ
        wcl.cbWndExtra = 0;
                            //ウィンドウインスタンスのエキス
    トラ
        wcl.hInstance = hThisInst;
                    //このプログラムのインスタンスへのハン
    ドル
        wcl.hIcon = LoadIcon(NULL, IDI_APPLICATION);    //アイコンへ
    のハンドル
        wcl.hCursor = LoadCursor(NULL, IDC_ARROW);              //
    カーソルへのハン
    ドル
        wcl.hbrBackground = BackGround_clear;           //(HBRUSH
    )GetStockObject(NULL_BRUSH); 背景ブラシへのハンドル
    //
        wcl.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);  //メニュー
        wcl.lpszClassName = szWinName;
            //ウィンドウクラ
```

```
ス名
    wcl.hIconSm = LoadIcon(NULL, IDI_WINLOGO);              //
スモールアイコンへのハン
ドル

    //ウィンドウクラスの登録
    if (!RegisterClassEx(&wcl)) {
            return(0);
    }
    //ウィンドウの生成
    hWnd = CreateWindowEx(
            0,                          //拡張ウィンドウスタイル
            szWinName,                                  //ウィンドウ
クラス名
            szWinName,                                  //ウィンドウ
名
            WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU |
WS_MINIMIZEBOX,       //ウィンドウスタ
イル
            CW_USEDEFAULT,                  //座標 x
            CW_USEDEFAULT,                  //座標 y
            nWidth,                         //幅
            nHeight,                            //高さ
            HWND_DESKTOP,                   //親ウィンドウへのハ
ンドル
            NULL,                                       //メニューへ
のハンドル
            hThisInst,                                  //このプログ
ラムのインスタンスへのハンドル
            NULL                                        //追加引数
    );
    if (!hWnd) {
            return FALSE;
    }

    //ウィンドウの表示
    ShowWindow(hWnd, nWinMode);
    UpdateWindow(hWnd);
```

```
        // アプリのアイコンを変更
        HICON hIcon;
        hIcon = (HICON)LoadImage(hThisInst, MAKEINTRESOURCE(
    IDI_ICON1), IMAGE_ICON, 48, 48, 0);
        SendMessage(hWnd, WM_SETICON, ICON_SMALL, (LPARAM)hIcon);

        // ウィンドウのタイトルを遅延時間のグループ名に設定
        if (WindowTitleText[0] != '\0') {
                SendMessage(hWnd, WM_SETTEXT, NULL, (LPARAM)
    WindowTitleText);
        }

        return hWnd;
}


//////////////////
// フォントの適用
//////////////////
bool FontFunc() {

        hFont = CreateFont(
                fontsize, 0, 0, 0,
                FW_MEDIUM,
                false, false, false,
                SHIFTJIS_CHARSET, OUT_DEFAULT_PRECIS,
                CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                (VARIABLE_PITCH | FF_DONTCARE), _T("メイリオ"));
        hFont1 = CreateFont(
                fontsize2, 0, 0, 0,
                FW_MEDIUM,
                false, false, false,
                SHIFTJIS_CHARSET, OUT_DEFAULT_PRECIS,
                CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                (VARIABLE_PITCH | FF_DONTCARE), _T("メイリオ"));
```

```
    hFont2 = CreateFont(
            fontsize3, 0, 0, 0,
            FW_BOLD,
            false, false, false,
            SHIFTJIS_CHARSET, OUT_DEFAULT_PRECIS,
            CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
            (VARIABLE_PITCH | FF_DONTCARE), _T("メイリオ"));

    hFontTitle = CreateFont(
            40, 0, 0, 0,
            FW_MEDIUM,
            false, false, false,
            SHIFTJIS_CHARSET, OUT_DEFAULT_PRECIS,
            CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
            (VARIABLE_PITCH | FF_DONTCARE), _T("メイリオ"));

    // To ステティックコントロール
    SendMessage(hStaticProg, WM_SETFONT, (WPARAM)hFont1,
MAKELPARAM(false, 0));
    SendMessage(hStaticProg2, WM_SETFONT, (WPARAM)hFont1,
MAKELPARAM(false, 0));
    //SendMessage(hStaticTitle, WM_SETFONT, (WPARAM)hFontTitle
, MAKELPARAM(false, 0));
    SendMessage(hStaticDialogUser, WM_SETFONT, (WPARAM)hFont,
MAKELPARAM(false, 0));
    SendMessage(hStaticSentence, WM_SETFONT, (WPARAM)
hFontTitle, MAKELPARAM(false, 0));
    // To 評価結果記録用ボタン
    SendMessage(Button1, WM_SETFONT, (WPARAM)hFont2,
MAKELPARAM(false, 0));
    SendMessage(Button2, WM_SETFONT, (WPARAM)hFont2,
MAKELPARAM(false, 0));
    SendMessage(Button3, WM_SETFONT, (WPARAM)hFont2,
MAKELPARAM(false, 0));
    SendMessage(Button4, WM_SETFONT, (WPARAM)hFont2,
MAKELPARAM(false, 0));
```

```
    SendMessage(Button11, WM_SETFONT, (WPARAM)hFont2,
MAKELPARAM(false, 0));
    SendMessage(Button22, WM_SETFONT, (WPARAM)hFont2,
MAKELPARAM(false, 0));
    SendMessage(Button33, WM_SETFONT, (WPARAM)hFont2,
MAKELPARAM(false, 0));
    SendMessage(Button44, WM_SETFONT, (WPARAM)hFont2,
MAKELPARAM(false, 0));
    SendMessage(hSpeak, WM_SETFONT, (WPARAM)hFont1, MAKELPARAM
(false, 0));
    SendMessage(hLate, WM_SETFONT, (WPARAM)hFont1, MAKELPARAM(
false, 0));
    // To 性別選択エディットボックス
    SendMessage(CheckMen, WM_SETFONT, (WPARAM)hFont,
MAKELPARAM(false, 0));
    SendMessage(CheckLady, WM_SETFONT, (WPARAM)hFont,
MAKELPARAM(false, 0));
    // To 実験回数
    SendMessage(hNumberStatic, WM_SETFONT, (WPARAM)hFont,
MAKELPARAM(false, 0));
    SendMessage(hNumber, WM_SETFONT, (WPARAM)hFont1,
MAKELPARAM(false, 0));
    // To 下のボタン
    SendMessage(hCancel, WM_SETFONT, (WPARAM)hFont, MAKELPARAM
(false, 0));
    SendMessage(hQuit, WM_SETFONT, (WPARAM)hFont, MAKELPARAM(
false, 0));
    SendMessage(hReserch, WM_SETFONT, (WPARAM)hFont,
MAKELPARAM(false, 0));
    SendMessage(hNext, WM_SETFONT, (WPARAM)hFont, MAKELPARAM(
false, 0));
    // To 個人情報の登録ボタン
    SendMessage(hDialogUser, WM_SETFONT, (WPARAM)hFont,
MAKELPARAM(false, 0));
    // 名前入力エディットボックス To
    SendMessage(hEditName, WM_SETFONT, (WPARAM)hFont,
```

```
    MAKELPARAM(false, 0));
        // To 年齢
        SendMessage(hEditOld, WM_SETFONT, (WPARAM)hFont,
    MAKELPARAM(false, 0));
        // To コンボボックス
        SendMessage(latedatacombo, WM_SETFONT, (WPARAM)hFont,
    MAKELPARAM(false, 0));



        return true;
}
//////////////////////
// ダイアログの呼び出し
//////////////////////
bool DialogUserButton(HWND hWnd, LPARAM lParam) {

        hDialogUser = CreateWindowEx(
                WS_EX_WINDOWEDGE,
                _T("BUTTON"),
                _T("ここを押す(R)"),
                WS_VISIBLE | WS_CHILD | BS_PUSHBUTTON | BS_CENTER,
                Winsize.right - 30 - 350 - 260 - 20 + 90, 25 + 30
    + 20, 260, 40,
                hWnd,
                (HMENU)ID_DIALOG_OPEN,
                ((LPCREATESTRUCT)(lParam))->hInstance,
                NULL
        );

        // ステティックコントロール
        hStaticTitle = CreateWindow(
                _T("STATIC"),
                _T(""),
                WS_CHILD | WS_VISIBLE | SS_LEFT,
                50, 53, 350, 100,
                hWnd, (HMENU)ID_STATIC_DIALOG,
```

```
                ((LPCREATESTRUCT)(lParam))->hInstance,
                NULL
        );

        hStaticDialogUser = CreateWindow(
                _T("STATIC"),
                _T("ユーザー情報の登録:␣"),
                WS_CHILD | WS_VISIBLE | SS_LEFT,
                Winsize.right - 30 - 350 - 260 - 20 + 90, 25 + 20,
    260, 30,
                hWnd, (HMENU)ID_STATIC_DIALOG,
                ((LPCREATESTRUCT)(lParam))->hInstance,
                NULL
        );

        return true;
}

//////////////////////
// プログレスバーの作成
//////////////////////
bool CreateProgressBar(HWND hWnd, int sbHeight, LPARAM lParam) {

        //  プログレスバーの作成
        hProg = CreateWindowEx(
                PBS_SMOOTH,
                PROGRESS_CLASS,
                NULL,
                WS_CHILD | WS_VISIBLE,
                //130, //Winsize.right − 30 − 425,    // 左下
                //Winsize.bottom − 65, 330, sbHeight,   左下//
                //FrameProg.right − 440, FrameProg.top + 45, 330,
    sbHeight,
                200, 15, Winsize.right - 400, 40,
                //cyVScroll,
                hWnd,
```

```
                (HMENU)ID_PROG,
                ((LPCREATESTRUCT)(lParam))->hInstance,
                NULL
        );

        // 現在位置を表示させるステティックコントロール
        hStaticProg = CreateWindow(
                _T("STATIC"),
                _T("1␣/␣9"),
                WS_CHILD | WS_VISIBLE | SS_CENTER,
                //490, //Winsize.right − 3 − 100, 左下//
                //Winsize.bottom − 60, 53, 40,    // 左下
                //FrameProg.right − 88, FrameProg.top + 50, 53, 40,
                Winsize.right - 185, 20, 53, 30,
                hWnd, (HMENU)ID_STATICPROG,
                ((LPCREATESTRUCT)(lParam))->hInstance, NULL
        );

        hStaticProg2 = CreateWindow(
                _T("STATIC"),
                _T("進行状況:"),
                WS_CHILD | WS_VISIBLE | SS_CENTER,
                85, 20, 100, 30,
                hWnd, (HMENU)ID_STATICPROG2,
                ((LPCREATESTRUCT)(lParam))->hInstance, NULL
        );

        hStaticSentence = CreateWindow(
                _T("STATIC"),
                _T("開始ボタンを押すとここに文章の番号が表示されます。"),
                WS_CHILD | WS_VISIBLE | SS_CENTER,
                200, FrameProgTop.bottom + 20, Winsize.right -
    400, FrameCenter.top - FrameProgTop.bottom -20,
                hWnd, (HMENU)ID_STATICSENTENCE,
                ((LPCREATESTRUCT)(lParam))->hInstance, NULL
```

```
        // プログレスバーの範囲指定
        );
        SendMessage(hProg, PBM_SETRANGE, (WPARAM)0, MAKELPARAM(0,
    9));
        // 現在位置を設定
        SendMessage(hProg, PBM_SETPOS, (WPARAM)1, 0);
        // 回の増加分を指定 1
        SendMessage(hProg, PBM_SETSTEP, (WPARAM)1, 0);
        //最大値を取得
        _MAX = SendMessage(hProg, PBM_GETRANGE, 0, 0);

        return true;
}

////////////////////////////
// 評価結果入力ボタンの作成
////////////////////////////
bool GradeButtonFunc(HWND hWnd, LPARAM lParam) {

        /*int height2, width2, height3, width3;

      height2 = Frame2.bottom − Frame2.top;
      width2 = Frame2.right − Frame2.left;
      height3 = Frame3.bottom − Frame3.top;
      width3 = Frame3.right − Frame3.left;*/

        int Height, Width, WidthHalf, HeightButton, Margin;

        Height = (FrameBottom.top - 20) - (FrameCenter.top + 75);
        Width = Winsize.right - 60;
        WidthHalf = Width / 2 - 10;
        Margin = 12;
        HeightButton = (Height - 3 * Margin) / 4;

        hSpeak = CreateWindow(
                _T("STATIC"),
```

```
            _T("Q1読み上げたときにしゃべりにくさを感じますか？."),
            WS_CHILD | WS_VISIBLE | SS_CENTER,
            30, FrameCenter.top + 20, Winsize.right / 2 - 30,
35,
            hWnd,
            (HMENU)ID_HSPEAK,
            ((LPCREATESTRUCT)(lParam))->hInstance,
            NULL
    );

    hLate = CreateWindow(
            _T("STATIC"),
            _T("Q2読み上げたときに遅れが気になりますか？."),
            WS_CHILD | WS_VISIBLE | SS_CENTER,
            (Winsize.right - 30) / 2, FrameCenter.top + 20,
Winsize.right / 2 - 30, 35,
            hWnd,
            (HMENU)ID_HLATE,
            ((LPCREATESTRUCT)(lParam))->hInstance,
            NULL
    );

    Button4 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"),_T
("しゃべりにくくない優
()"),WS_VISIBLE | WS_CHILD | WS_GROUP | BS_AUTORADIOBUTTON |
WS_DISABLED,
            //10, (rectSpeak.bottom − rectSpeak.top) / 16 + 10, (
rectSpeak.right − rectSpeak.left) − 15, 40,
            //Frame2.left + 50, Frame2.top + (height2) ∗ 2 / 17, (
width2) − 55, 40,
            30, FrameCenter.top + 75, WidthHalf, HeightButton,
            hWnd,(HMENU)ID_BUTTON4,((LPCREATESTRUCT)(lParam))
->hInstance,NULL);

    Button3 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"),
_T("良　（しゃべりにくいが気にならない）
"), WS_VISIBLE | WS_CHILD | BS_AUTORADIOBUTTON | WS_DISABLED,
```

```
            //10, (rectSpeak.bottom − rectSpeak.top) ∗ 5 / 16 + 5, (
rectSpeak.right − rectSpeak.left) − 15, 40,
            //Frame2.left + 50, Frame2.top + (height2) ∗ 6 / 17, (
width2) − 55, 40,
            30, FrameCenter.top + 75 + HeightButton + Margin,
WidthHalf, HeightButton,
            hWnd, (HMENU)ID_BUTTON3, ((LPCREATESTRUCT)(lParam)
)->hInstance, NULL);

    Button2 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"),
_T("可　（しゃべりにくい）
"), WS_VISIBLE | WS_CHILD | BS_AUTORADIOBUTTON | WS_DISABLED,
            //10, (rectSpeak.bottom − rectSpeak.top) ∗ 9 / 16 − 5, (
rectSpeak.right − rectSpeak.left) − 15, 40,
            //Frame2.left + 50, Frame2.top + (height2) ∗ 10 / 17, (
width2) − 55, 40,
            30, FrameCenter.top + 75 + 2 * HeightButton + 2 *
Margin, WidthHalf, HeightButton,
            hWnd, (HMENU)ID_BUTTON2, ((LPCREATESTRUCT)(lParam)
)->hInstance, NULL);

    Button1 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"),
_T("不可（とてもしゃべりにくい）
"), WS_VISIBLE | WS_CHILD | BS_AUTORADIOBUTTON | WS_DISABLED,
            //10, (rectSpeak.bottom − rectSpeak.top) ∗ 13 / 16 − 10, (
rectSpeak.right − rectSpeak.left) − 15, 40,
            //Frame2.left + 50, Frame2.top + (height2) ∗ 14 / 17, (
width2) − 55, 40,
            30, FrameCenter.bottom - 20 - HeightButton,
WidthHalf, HeightButton,
            hWnd, (HMENU)ID_BUTTON1, ((LPCREATESTRUCT)(lParam)
)->hInstance, NULL);

    Button44 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"),
_T("優　（遅れがまったく分からない）
"), WS_VISIBLE | WS_CHILD | WS_GROUP | BS_AUTORADIOBUTTON |
WS_DISABLED,
            //10, (rectLate.bottom − rectLate.top) / 16 + 10, (rectLate
```

```
.right − rectLate.left) − 15, 40,
            //Frame3.left + 50, Frame3.top + (height3) ∗ 2 / 17, (
width3) − 55, 40,
            30 + (Width / 2) + 10, FrameCenter.top + 75,
WidthHalf, HeightButton,
            hWnd, (HMENU)ID_BUTTON44, ((LPCREATESTRUCT)(lParam
))->hInstance, NULL);


    Button33 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"),
_T("良　（遅れが分かるが気にならない）
"), WS_VISIBLE | WS_CHILD | BS_AUTORADIOBUTTON | WS_DISABLED,
            //10, (rectLate.bottom − rectLate.top) ∗ 5 / 16 + 5, (
rectLate.right − rectLate.left) − 15, 40,
            //Frame3.left + 50, Frame3.top + (height3) ∗ 6 / 17, (
width3) − 55, 40,
            30 + (Width / 2) + 10, FrameCenter.top + 75 +
HeightButton + Margin, WidthHalf, HeightButton,
            hWnd, (HMENU)ID_BUTTON33, ((LPCREATESTRUCT)(lParam
))->hInstance, NULL);


    Button22 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"),
_T("可　（遅れが気になる）
"), WS_VISIBLE | WS_CHILD | BS_AUTORADIOBUTTON | WS_DISABLED,
            //10, (rectLate.bottom − rectLate.top) ∗ 9 / 16 − 5, (
rectLate.right − rectLate.left) − 15, 40,
            //Frame3.left + 50, Frame3.top + (height3) ∗ 10 / 17, (
width3) − 55, 40,
            30 + (Width / 2) + 10, FrameCenter.top + 75 + 2 ∗
HeightButton + 2 ∗ Margin, WidthHalf, HeightButton,
            hWnd, (HMENU)ID_BUTTON22, ((LPCREATESTRUCT)(lParam
))->hInstance, NULL);


    Button11 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"),
_T("不可（遅れがはっきり分かる）
"), WS_VISIBLE | WS_CHILD | BS_AUTORADIOBUTTON | WS_DISABLED,
            //10, (rectLate.bottom − rectLate.top) ∗ 13 / 16 − 10, (
rectLate.right − rectLate.left) − 15, 40,
            //Frame3.left + 50, Frame3.top + (height3) ∗ 14 / 17, (
```

```
width3) − 55, 40,
            30 + (Width / 2) + 10, FrameCenter.bottom - 20 -
HeightButton, WidthHalf, HeightButton,
            hWnd, (HMENU)ID_BUTTON11, ((LPCREATESTRUCT)(lParam
))->hInstance, NULL);

    /*if (!SetRect(&RectButton1, Frame2.left + 50 − 5, Frame2.top + (
height2) * 4 / 16, 400, Frame2.top + height2 * 4 / 16 + 40)) {
        return false;
  }*/
    RectButton1 = Button_ScreenToClient(hWnd, Button1,
RectButton1, 26);
    RectButton2 = Button_ScreenToClient(hWnd, Button2,
RectButton2, 26);
    RectButton3 = Button_ScreenToClient(hWnd, Button3,
RectButton3, 26);
    RectButton4 = Button_ScreenToClient(hWnd, Button4,
RectButton4, 26);

    RectButton11 = Button_ScreenToClient(hWnd, Button11,
    RectButton11, RectButton1.right + Margin);
    RectButton22 = Button_ScreenToClient(hWnd, Button22,
    RectButton22, RectButton2.right + Margin);
    RectButton33 = Button_ScreenToClient(hWnd, Button33,
    RectButton33, RectButton3.right + Margin);
    RectButton44 = Button_ScreenToClient(hWnd, Button44,
    RectButton44, RectButton4.right + Margin);

    return true;
}

// CreateOWnerDrawButton Func
bool CreateOwnerDrawButton(HWND hWnd, LPARAM lParam) {

    int Height, Width, WidthHalf, HeightButton, Margin;
```

```
    Height = (FrameBottom.top - 20) - (FrameCenter.top + 75);
    Width = Winsize.right - 60;
    WidthHalf = Width / 2 - 10;
    Margin = 12;
    HeightButton = (Height - 3 * Margin) / 4;

    hSpeak = CreateWindow(
            _T("STATIC"),
            _T("Q1読み上げたときにしゃべりにくさを感じますか？."),
            WS_CHILD | WS_VISIBLE | SS_CENTER,
            30, FrameCenter.top + 20, Winsize.right / 2 - 30,
35,
            hWnd,
            (HMENU)ID_HSPEAK,
            ((LPCREATESTRUCT)(lParam))->hInstance,
            NULL
    );

    hLate = CreateWindow(
            _T("STATIC"),
            _T("Q2読み上げたときに遅れが気になりますか？."),
            WS_CHILD | WS_VISIBLE | SS_CENTER,
            (Winsize.right - 30) / 2, FrameCenter.top + 20,
Winsize.right / 2 - 30, 35,
            hWnd,
            (HMENU)ID_HLATE,
            ((LPCREATESTRUCT)(lParam))->hInstance,
            NULL
    );

    Button4 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"),
_T("しゃべりにくくない優
()"), WS_VISIBLE | WS_CHILD | WS_GROUP | BS_OWNERDRAW |
WS_DISABLED | SS_LEFT,
            //10, (rectSpeak.bottom − rectSpeak.top) / 16 + 10, (
rectSpeak.right − rectSpeak.left) − 15, 40,
            //Frame2.left + 50, Frame2.top + (height2) ∗ 2 / 17, (
```

```
width2) − 55, 40,
            30, FrameCenter.top + 75, WidthHalf, HeightButton,
            hWnd, (HMENU)ID_BUTTON4, ((LPCREATESTRUCT)(lParam)
)->hInstance, NULL);

    Button3 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"),
_T("しゃべりにくいが気にならない良
()"), WS_VISIBLE | WS_CHILD | BS_OWNERDRAW | WS_DISABLED,
            //10, (rectSpeak.bottom − rectSpeak.top) ∗ 5 / 16 + 5, (
rectSpeak.right − rectSpeak.left) − 15, 40,
            //Frame2.left + 50, Frame2.top + (height2) ∗ 6 / 17, (
width2) − 55, 40,
            30, FrameCenter.top + 75 + HeightButton + Margin,
WidthHalf, HeightButton,
            hWnd, (HMENU)ID_BUTTON3, ((LPCREATESTRUCT)(lParam)
)->hInstance, NULL);

    Button2 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"),
_T("しゃべりにくい可
()"), WS_VISIBLE | WS_CHILD | BS_OWNERDRAW | WS_DISABLED,
            //10, (rectSpeak.bottom − rectSpeak.top) ∗ 9 / 16 − 5, (
rectSpeak.right − rectSpeak.left) − 15, 40,
            //Frame2.left + 50, Frame2.top + (height2) ∗ 10 / 17, (
width2) − 55, 40,
            30, FrameCenter.top + 75 + 2 * HeightButton + 2 *
Margin, WidthHalf, HeightButton,
            hWnd, (HMENU)ID_BUTTON2, ((LPCREATESTRUCT)(lParam)
)->hInstance, NULL);

    Button1 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"),
_T("とてもしゃべりにくい不可
()"), WS_VISIBLE | WS_CHILD | BS_OWNERDRAW | WS_DISABLED,
            //10, (rectSpeak.bottom − rectSpeak.top) ∗ 13 / 16 − 10, (
rectSpeak.right − rectSpeak.left) − 15, 40,
            //Frame2.left + 50, Frame2.top + (height2) ∗ 14 / 17, (
width2) − 55, 40,
            30, FrameCenter.bottom - 20 - HeightButton,
WidthHalf, HeightButton,
```

```
            hWnd, (HMENU)ID_BUTTON1, ((LPCREATESTRUCT)(lParam)
)->hInstance, NULL);

    Button44 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"),
_T("遅れがまったく分からない優
()"), WS_VISIBLE | WS_CHILD | WS_GROUP | BS_OWNERDRAW |
WS_DISABLED,
            //10, (rectLate.bottom − rectLate.top) / 16 + 10, (rectLate
.right − rectLate.left) − 15, 40,
            //Frame3.left + 50, Frame3.top + (height3) ∗ 2 / 17, (
width3) − 55, 40,
            30 + (Width / 2) + 10, FrameCenter.top + 75,
WidthHalf, HeightButton,
            hWnd, (HMENU)ID_BUTTON44, ((LPCREATESTRUCT)(lParam
))->hInstance, NULL);

    Button33 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"),
_T("遅れが分かるが気にならない良
()"), WS_VISIBLE | WS_CHILD | BS_OWNERDRAW | WS_DISABLED,
            //10, (rectLate.bottom − rectLate.top) ∗ 5 / 16 + 5, (
rectLate.right − rectLate.left) − 15, 40,
            //Frame3.left + 50, Frame3.top + (height3) ∗ 6 / 17, (
width3) − 55, 40,
            30 + (Width / 2) + 10, FrameCenter.top + 75 +
HeightButton + Margin, WidthHalf, HeightButton,
            hWnd, (HMENU)ID_BUTTON33, ((LPCREATESTRUCT)(lParam
))->hInstance, NULL);

    Button22 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"),
_T("遅れが気になる可
()"), WS_VISIBLE | WS_CHILD | BS_OWNERDRAW | WS_DISABLED,
            //10, (rectLate.bottom − rectLate.top) ∗ 9 / 16 − 5, (
rectLate.right − rectLate.left) − 15, 40,
            //Frame3.left + 50, Frame3.top + (height3) ∗ 10 / 17, (
width3) − 55, 40,
            30 + (Width / 2) + 10, FrameCenter.top + 75 + 2 *
HeightButton + 2 * Margin, WidthHalf, HeightButton,
            hWnd, (HMENU)ID_BUTTON22, ((LPCREATESTRUCT)(lParam
```

```
    ))->hInstance, NULL);

        Button11 = CreateWindowEx(WS_EX_WINDOWEDGE, _T("BUTTON"),
    _T("遅れがはっきり分かる不可
    ()"), WS_VISIBLE | WS_CHILD | BS_OWNERDRAW | WS_DISABLED,
                //10, (rectLate.bottom − rectLate.top) ∗ 13 / 16 − 10, (
    rectLate.right − rectLate.left) − 15, 40,
                //Frame3.left + 50, Frame3.top + (height3) ∗ 14 / 17, (
    width3) − 55, 40,
                30 + (Width / 2) + 10, FrameCenter.bottom - 20 -
    HeightButton, WidthHalf, HeightButton,
                hWnd, (HMENU)ID_BUTTON11, ((LPCREATESTRUCT)(lParam
    ))->hInstance, NULL);

        return true;
}

////////////////////////////////////////
// ボタンの位置座標をクライアント座標に変換
////////////////////////////////////////
RECT Button_ScreenToClient(HWND hWnd, HWND Button, RECT RectButton
    , int left) {

        // ポイント構造体
        POINT P1Button, P2Button;

        // ウィンドウ座標をスクリーン座標で取得
        GetWindowRect(Button, &RectButton);
        P1Button = { RectButton.left, RectButton.top };
        P2Button = { RectButton.right, RectButton.bottom };
        // スクリーン座標からクライアント座標に変換
        ScreenToClient(hWnd, &P1Button);
        ScreenToClient(hWnd, &P2Button);
        // ウィンドウのクライアントの位置座標を保存
        SetRect(&RectButton, left, P1Button.y - 2, P2Button.x + 2,
     P2Button.y + 2);
```

```
        return RECT(RectButton);
}

////////////////////////////////////////////////////
// 実験開始ボタン・ボタン・戻るボタン・終了ボタン Next
////////////////////////////////////////////////////
bool ReserchStartFunc(HWND hWnd, LPARAM lParam) {

        hReserch = CreateWindowEx(
                NULL, //WS_EX_WINDOWEDGE,
                _T("BUTTON"),
                _T("開始(S)"),
                WS_VISIBLE | WS_CHILD | BS_PUSHBUTTON | BS_CENTER
    | WS_DISABLED,
                //Winsize.right − 676, (Winsize.bottom − 57), 150, 40,
                Winsize.left + 30, (Winsize.bottom - 67), 150, 50,
                hWnd,
                (HMENU)ID_ReserchStart,
                ((LPCREATESTRUCT)(lParam))->hInstance,
                NULL
        );

        hCancel = CreateWindowEx(
                0, //WS_EX_STATICEDGE,
                _T("BUTTON"),
                _T("＜前へ戻る(B)"),
                WS_VISIBLE | WS_CHILD | BS_PUSHBUTTON | BS_CENTER
    | WS_DISABLED,
                Winsize.right - 150 - 350 - 40 - 100 + 10, (
    Winsize.bottom - 67), 160, 50,
                hWnd,
                (HMENU)ID_CANCEL,
                ((LPCREATESTRUCT)(lParam))->hInstance,
                NULL
        );
```

```
            hNext = CreateWindowEx(
                    WS_EX_WINDOWEDGE,
                    _T("BUTTON"),
                    _T("次の文章に進む (N)＞"),
                    WS_VISIBLE | WS_CHILD | BS_PUSHBUTTON | BS_CENTER
        | WS_DISABLED,
                    Winsize.right - 140 - 300 - 30 + 9, (Winsize.
        bottom - 67), 250, 50,
                    hWnd,
                    (HMENU)ID_NEXTBUTTON,
                    ((LPCREATESTRUCT)(lParam))->hInstance,
                    NULL
                    );



            hQuit = CreateWindowEx(
                    0,//WS_EX_STATICEDGE,
                    _T("BUTTON"),
                    _T("終了 (E)"),
                    WS_VISIBLE | WS_CHILD | BS_PUSHBUTTON | BS_CENTER,
                    Winsize.right - 32 - 160, (Winsize.bottom - 67),
        160, 50,
                    hWnd,
                    (HMENU)ID_QUIT,
                    ((LPCREATESTRUCT)(lParam))->hInstance,
                    NULL
            );

            return true;
    }

    ////////////////////////////////////////////
    // コンボボックス作成, ファイルの読み込み ini
    ////////////////////////////////////////////
    bool ComboboxFunc(HWND hWnd, LPARAM lParam) {
```

```
    stringstream ss, tt;
    string s, t;
    char   latename[STRLEN], latedata[STRLEN];
    latename[0] = '\0';
    latedata[0] = '\0';
    char Path[MAX_PATH + 1];
    char settingpath[MAX_PATH + 1];
    settingpath[0] = '\0'; // 終端 NULL

    // 遅延データ設定用コンボボックスの作成
    latedatacombo = CreateWindow(
            _T("COMBOBOX"),
            NULL,
            WS_CHILD | WS_VISIBLE | CBS_DROPDOWNLIST,
            Winsize.right - 330,
            605,
            260,
            400,
            //30, 130, 340, 400,
            hWnd,
            (HMENU)ID_LATEINI,
            ((LPCREATESTRUCT)(lParam))->hInstance,
            NULL
    );

    // 実行ファイルのパスを取得
    if (0 != GetModuleFileName(NULL, Path, MAX_PATH)) {
            char drive[MAX_PATH + 1], dir[MAX_PATH + 1], fname
[MAX_PATH + 1], ext[MAX_PATH + 1];
            // ドライブ名, ディレクトリパス名, ファイル名, 拡張子
            _splitpath_s(Path, drive, sizeof(drive), dir,
sizeof(dir),
                    fname, sizeof(fname), ext, sizeof(ext));
            _stprintf_s(settingpath, MAX_PATH + 1, _T("%s%
ssetting.ini"), drive, dir);
```

```
        }
        // ファイルの読み込み ini
        GetPrivateProfileString(_T("setting"), _T("latedataname"),
    _T("error"), latename, sizeof(latename), settingpath);

        ss << latename;

        // 遅延データのキーをコンボボックスに詰めていく
        int i = 0;
        while (getline(ss, s, ',')) {
                char* cstr = new char[s.size() + 1];
                char_traits<char>::copy(cstr, s.c_str(), s.size()
    + 1);
                SendMessage(GetDlgItem(hWnd, (int)ID_LATEINI),
    CB_INSERTSTRING, i, (LPARAM)cstr);
                i++;
        }

        SendMessage(GetDlgItem(hWnd, (int)ID_LATEINI),
    CB_SETCURSEL, 0, 0);

        return true;
}
```

ソースコード B.3: file.cpp

```
//////////////////////////////////////////
// file.cpp: ファイル入出力関係の処理を記述
//////////////////////////////////////////

#include<windows.h>
#include<tchar.h>
#include<stdio.h>
#include<string>
#include<string.h>
#include<locale.h>
```

```
//////////////////////
// 自作ヘッダファイル
//////////////////////
#include "window.h"
#include "main.h"
#include "file.h"

using namespace std;

///////////
//定数/
///////////
#define STRLEN 256
#define LENSNUMBER 128
#define FILESIZE 100

/// <summary>
/// グローバル変数
/// ///////////////
extern HANDLE hfile;                                          //
    ファイルハン
    ドル
extern int  _MAX;                                              //
    読んだ回数を保持するグローバル
    変数
int SNumber[LENSNUMBER];                                  // 読んだ文章
    の番号
unsigned int NumberOfTimes;                                // 実験回数
    * 2
const char* IpBuffer;                                          //
    遅延時間を書き込む際に
    必要
char latedata[STRLEN];                                       // 遅延
    時間
unsigned int GradeS[STRLEN], GradeL[STRLEN];  // 評価結果
char GradeSpeak[STRLEN], GradeLate[STRLEN];
DWORD dWFileSizeByte;                                       //
    WriteFile()で使
    用
```

```
std::string fileNameCapture;

////////////////////////////////////
//  グローバル変数（file.でのみ使用）cpp
////////////////////////////////////
static char FileName[MAX_PATH];              // ユーザーが入力したファ
    イル名


//////////////////////
// ファイルのオープン
//////////////////////
bool file_open(HWND hWnd) {



        hfile = CreateFile(
                FileName,        // ファイルの名前
                GENERIC_READ | GENERIC_WRITE,   // 書き込みモード
                0,               // 共有モード
                NULL,            // セキュリティ
                OPEN_ALWAYS,   // 上書き保存
                FILE_ATTRIBUTE_NORMAL,
                NULL
        );
        if (hfile == INVALID_HANDLE_VALUE) {
                MessageBox(NULL, _T("ファイルの作成に失敗しました。
    "), _T("エラー"), MB_OK);
                return 1;
        }
        // ファイルサイズの取得
        dWFileSizeByte = GetFileSize(hfile, NULL);

        return true;
}

// 引数で指定した文字列をファイルへの書き込み
```

```
bool WriteFileStr(LPCSTR strText) {

        DWORD dwWriteSize;

        WriteFile(hfile, strText, (DWORD)_tcslen(strText), &
    dwWriteSize, NULL);

        return true;
}

// 評価結果をファイルへ書き込む
bool file_write() {

        DWORD dwWriteSize;
        string GradeS_str, GradeL_str;

        //const char* IpBuffer1;
        //const char* IpBuffer2;
        //// 入力結果を二つに分ける「しゃべりにくさ」と「遅れが気になるか」)
    (
        //DivideGradeFunc();
        //IpBuffer1 = GradeSpeak;
        //IpBuffer2 = GradeLate;
        //
        unsigned int tempS = 0;
        for (unsigned int item : GradeS) {
                if (tempS >= NumberOfTimes / 2) break;
                else {
                        GradeS_str += to_string(item) + ",";
                        tempS += 1;
                }
        }
        tempS = 0;
        for (unsigned int item : GradeL) {
                if (tempS >= NumberOfTimes / 2) break;
                else {
                        GradeL_str += to_string(item) + ",";
```

```
                        tempS += 1;
                }
        }
        // ファイルに書き込む
        WriteFile(hfile, GradeS_str.data(), GradeS_str.length(), &
    dwWriteSize, NULL);
        //WriteFile(hfile, _T("\r\n"), (DWORD)_tcslen(_T("\r\n")), &
    dwWriteSize, NULL); // 改行コー
    ド
        WriteFile(hfile, GradeL_str.data(), GradeL_str.length(), &
    dwWriteSize, NULL);

        return true;
}

////////////////////////////////////
/// ファイルへの書き込み用関数の定義
////////////////////////////////////
bool WriteFileFunc() {

        DWORD dwWriteSize = 0;

        // ロケールを日本に設定
        setlocale(LC_ALL, "Japanese");

        // 指定したファイルが存在した場合
        if (dWFileSizeByte > FILESIZE) {
                // ファイルを上書きする場合出力の開始点を現在のファイルの
    終了位置に設定,
                SetFilePointer(hfile, 0, NULL, FILE_END);
                // 入力結果の書き込み
                WriteFileResult(dwWriteSize);
        }else {
        // 指定したファイルが存在しなかった場合
                WriteFile(hfile, _T("ユーザー情報年齢性別文章
    ,,,"), (DWORD)_tcslen(_T("ユーザー情報年齢性別文章
    ,,,")), &dwWriteSize, NULL);
                if (_MAX == 9) {
```

```
                    WriteFile(hfile, _T(",,,,,,,,,"), (DWORD)
_tcslen(_T(",,,,,,,,,")), &dwWriteSize, NULL);
            }
            else {
                    WriteFile(hfile, _T(",,,,,,,,"), (DWORD)
_tcslen(_T(",,,,,,,,")), &dwWriteSize, NULL);
            }
            WriteFile(hfile, _T("遅延時間
"), (DWORD)_tcslen(_T("遅延時間")), &dwWriteSize, NULL);
            if (_MAX == 9) {
                    WriteFile(hfile, _T(",,,,,,,,,,"), (DWORD)
_tcslen(_T(",,,,,,,,,,")), &dwWriteSize, NULL);
            }
            else {
                    WriteFile(hfile, _T(",,,,,,,,,"), (DWORD)
_tcslen(_T(",,,,,,,,,")), &dwWriteSize, NULL);
            }
            WriteFile(hfile, _T("しゃべりにくさ
"), (DWORD)_tcslen(_T("しゃべりにくさ")), &dwWriteSize, NULL);
            if (_MAX == 9) {
                    WriteFile(hfile, _T(",,,,,,,,,"), (DWORD)
_tcslen(_T(",,,,,,,,,")), &dwWriteSize, NULL);
            }
            else {
                    WriteFile(hfile, _T(",,,,,,,,"), (DWORD)
_tcslen(_T(",,,,,,,,")), &dwWriteSize, NULL);
            }
            WriteFile(hfile, _T("遅れ"), (DWORD)_tcslen(_T("遅
れ")), &dwWriteSize, NULL);
            // 入力結果の書き込み
            WriteFileResult(dwWriteSize);
    }
    // メモリの解放
    //free(strTextName);
    //free(strTextOld);

    return true;
```

```
}

//////////////////////
// 評価結果の書き込み
//////////////////////
bool WriteFileResult(DWORD dwWriteSize) {

        // 改行コード
        WriteFile(hfile, _T("\r\n"), (DWORD)_tcslen(_T("\r\n")), &
    dwWriteSize, NULL);
        // ハイパーリンクの作成
        char data[STRLEN];
        std::string NameTemp = _T("ここをクリック");
        sprintf_s(
                data,
                "\"=HYPERLINK(\"\"%s\"\",\"\"%s\"\")\"",
        fileNameCapture.c_str(),NameTemp.c_str());
        // ファイル名の書き込み
        WriteFile(hfile, data, strlen(data), &dwWriteSize, NULL);
        // 性別を書き込む
        WriteFile(hfile, strMenLady, (DWORD)strlen(strMenLady), &
    dwWriteSize, NULL);
        // 読んだ文章の番号を書き込む
        string array1_str;
        unsigned int temp = 0;
        for (int item : SNumber) {
                if (temp >= NumberOfTimes / 2) break;
                else {
                        array1_str += to_string(item) + ",";
                        temp += 1;
                }
        }
        WriteFile(hfile, array1_str.data(), array1_str.length(), &
    dwWriteSize, NULL);
        // テキストファイルに遅延時間を書き込む
        IpBuffer = latedata;
```

```
        WriteFile(hfile, IpBuffer, lstrlen(IpBuffer), &dwWriteSize
    , NULL);
        // カンマ
        WriteFile(hfile, _T(","), (DWORD)_tcslen(_T(",")), &
    dwWriteSize, NULL);
        // ファイルへ評価結果を書き込む
        file_write();

        return true;
}
////////////////////////////////////////////
// 名前を付けて保存ダイアログボックスの作成 []
////////////////////////////////////////////
bool SelectFile(HWND hWnd) {


        static OPENFILENAME ofn;
        memset(&ofn, 0, sizeof(ofn));
        memset(&FileName, 0, sizeof(char) * MAX_PATH);

        // OPENFILENAME 構造体の初期化
        ofn.lStructSize = sizeof(ofn);
        ofn.hwndOwner = hWnd;
        //ofn.lpstrFilter = _T("Text Files(*.txt)\0*.txt\0All Files(*.*)
    \0*.*\0\0");
        ofn.lpstrFilter = _T("CSV␣Files(*.csv)\0*.csv\0All␣Files
    (*.*)\0*.*\0\0");
        ofn.lpstrFile = FileName;
        ofn.nMaxFile = MAX_PATH;
        ofn.lpstrDefExt = _T("csv");
        ofn.Flags = OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT |
    OFN_PATHMUSTEXIST;
        ofn.lpstrTitle = "名前を付けて保存";

        if (!GetSaveFileName(&ofn)) {
                MessageBox(hWnd, _T("ファイルを作成できませんでした。
```

```
    "), _T("エラー"), MB_OK | MB_ICONERROR);
                return false;
        }
        // 指定ファイルが既に開かれていたとき、注意喚起
        //WarningFileOpen(hWnd);


        return true;
}

//////////////////////////////////////////
        // 指定ファイルが既に開かれていたとき、注意喚起
        //////////////////////////////////////////
bool WarningFileOpen(HWND hWnd) {

        HANDLE hFile = NULL;
        bool Succeeded = false;
        int MsgResult;

        // ファイルを読み取り専用でオープン
        hFile = FileOpenReadOnly(FileName, hFile);

        // ファイルが閉じられるまで無限ループ
        while (!Succeeded) {
                if (hFile != INVALID_HANDLE_VALUE || hFile != (
    HANDLE)0xffffffff) {
                        // ファイルが開かれていない場合
                        CloseHandle(hFile);
                        Succeeded = true;
                }
                else {
                        // 指定ファイルが既に開かれていた場合
                        MsgResult = MessageBox(
                                hWnd,
                                _T("指定されたファイルは既に開かれてい
    ます。\r\ファイルを閉じてからボタンを押してください。nOK"),
                                _T("ファイルが使用中です"),
```

```
                                MB_OK | MB_ICONWARNING);
                        if (MsgResult == IDOK) {
                                // ボタンが押された場合 OK
                                CloseHandle(hFile);
                                hFile = FileOpenReadOnly(FileName,
    hFile);
                        }
                }
        }

        return true;
}

////////////////////////////////////
// ファイルを読み取り専用でオープン
////////////////////////////////////
HANDLE FileOpenReadOnly(LPCSTR FileName, HANDLE hFile) {

        hFile = CreateFile(
                FileName,
                GENERIC_READ,
                0,
                NULL,
                OPEN_EXISTING,
                FILE_ATTRIBUTE_NORMAL,
                NULL);

        return hFile;
}
```

ソースコード B.4: UserInfoWindow.cpp

```
  ////////////////////////////////////////////
///UserInfoWindow.cpp: ユーザー情報入力用関数の定義
////////////////////////////////////////////

#include<windows.h>
```

```
#include<tchar.h>
#include <atlbase.h>
#include<atlimage.h>
#include<string>
#include<io.h>
#include<CommCtrl.h>
//#include"torch/torch.h"
//////////////////////
// 自作ヘッダファイル
//////////////////////
#include "window.h"
#include "main.h"
#include"file.h"
#include"resource.h"
#include"UserInfoWindow.h"

////////////////////////////////////////
// GDI+
////////////////////////////////////////
#pragma comment(lib,"Gdiplus.lib")
#include <ole2.h>
#include <gdiplus.h>

#pragma comment(lib, "comctl32.lib")

//////////
// 定数
//////////
#define STRLEN 256
#define MAX_TOUCH_POINTS 3000          // タッチポイントの記憶上限
#define MAX_PATH 260
#define black_pen RGB(0, 0, 0)
#define white_pen RGB(255, 255, 255)
#define red_pen RGB(255, 0, 0)
#define blue_pen RGB(0, 0, 255)
#define yellow_pen RGB(255, 255, 0)
```

```
#define purple_pen RGB(128, 0, 128)

#define Black 1
#define White 2
#define Red 3
#define Blue 4
#define Yellow 5
#define Purple 6

//////////////////////
// 消しゴムの大きさ
//////////////////////
#define ESmall 11
#define EMidium 12
#define EBig 13

// コントロールのハンドル
HWND hButtonNext, hStaticName, hStaticOld, hStaticS, hRadioMen,
    hRadioLady, hStaticComment1, hStaticCombo, hStaticUserInfo,
    hNameErase, hOldErase;

extern const HBRUSH BackGround_clear;
extern std::string fileNameCapture;

// ツールバーの背景色
static const HBRUSH hToolbarBackGroundBrush = CreateSolidBrush(RGB
    (0, 255, 0));

// ペン
bool Erase = true;
UINT PenColor=Black;

// グローバル変数
LPTSTR strTextName;
LPTSTR strTextOld;
LPCSTR strMenLady;
```

```
// フォント
HFONT hFontUserInfoWindow1, hFontUserInfoWindowRadio, hFontBig,
    hTopComment;

// WM_PAINT
HDC hMemDC;
HGDIOBJ hOld;

// WM_TOUCH
POINT g_ptPrevious[MAX_TOUCH_POINTS];
int idLookup[MAX_TOUCH_POINTS];
bool g_wasinside[MAX_TOUCH_POINTS];

// WM_COMMAND
bool clickedRadio = false;

// 構造体変数の定義 RECT
RECT RectTouchTop = { 30, 0, Winsize.right - 30, 85 };
RECT RectTouchOld = {180, 350, Winsize.right - 70,  500};
RECT RectTouchOld_1 = {150, 350, 150 + 220, 350 + 220};
RECT RectTouchOld_2 = { 150 + 220 + 50, 350, 150 + 440 + 50, 350 +
     220};
RECT RectTouchOld_3 = { RectTouchOld_2.right + 50, RectTouchOld_2.
    top, RectTouchOld_3.left + 220, RectTouchOld_3.top + 220 };
RECT RectTouchName = { 150, 105, RectTouchOld_3.right,  310 };
RECT RectTouchS = { 180, 540, 440, 600};
RECT rcName, rcOld100, rcOld10, rcOld1;

////////////////////////////////
// 最初に表示されるウィンドウの作成
////////////////////////////////
HWND CreateNewWindow(HWND hWnd, HINSTANCE hThisInst, const int
    nWidth, const int nHeight, int nWinMode) {

        WNDCLASSEX      wcl2{};
```

```
    HWND htb{};

    //ウィンドウクラスの定義
    wcl2.cbSize = sizeof(WNDCLASSEX);
        //構造体のサイズ
WNDCLASSEX
    wcl2.style = 0;
                        //ウィンドウクラススタ
イル
    wcl2.lpfnWndProc = WndProc2;
        //ウィンドウ関
数
    wcl2.cbClsExtra = 0;
                //ウィンドウクラスのエキス
トラ
    wcl2.cbWndExtra = 0;
                //ウィンドウインスタンスのエキス
トラ
    wcl2.hInstance = hThisInst;
                //このプログラムのインスタンスへのハン
ドル
    wcl2.hIcon = LoadIcon(NULL, IDI_APPLICATION);   //アイコンへ
のハンドル
    wcl2.hCursor = LoadCursor(NULL, IDC_ARROW);             //
カーソルへのハン
ドル
    wcl2.hbrBackground = BackGround_clear;           //(
HBRUSH)GetStockObject(NULL_BRUSH); 背景ブラシへのハンドル
//
    wcl2.lpszMenuName = MAKEINTRESOURCE(IDR_MENU2);;        //
メ
ニュー
    wcl2.lpszClassName = _T("NewWindowClass");
                        //ウィンドウクラ
ス名
    wcl2.hIconSm = LoadIcon(NULL, IDI_WINLOGO);             //
スモールアイコンへのハン
ドル

    if (!RegisterClassEx(&wcl2)) {
            return FALSE;
```

```
    }

    hWnd = CreateWindowEx(
            0,
            _T("NewWindowClass"),
                                                    //ウィンドウクラ
ス名
            _T("ユーザー情報の入力"),
                                                          //ウィンド
ウ名
            WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU |
WS_MINIMIZEBOX,         //ウィンドウスタ
イル
            CW_USEDEFAULT, CW_USEDEFAULT,
            nWidth,  nHeight,
            HWND_DESKTOP,    //親ウィンドウへのハンドル
            NULL,                                          //メニューへ
のハンドル
            hThisInst,                                     //このプログ
ラムのインスタンスへのハンドル
            NULL                                             //追加
引数
    );

    if (!hWnd) {
            return FALSE;
    }
    // ウィンドウの表示
    ShowWindow(hWnd, nWinMode);
    UpdateWindow(hWnd);



    if (!RegisterTouchWindow(hWnd, 0)) {
            MessageBox(hWnd, _T("このデバイスにはタッチ機能がありま
せん。"), _T("Error"), MB_OK);
            return FALSE;
    }
```

```
        // 構造体の配列の初期化（で使用）POINTWM_TOUCH
        for (size_t i = 0; i < MAX_TOUCH_POINTS; i++) {
                g_ptPrevious[i].x = -1;
                g_ptPrevious[i].y = -1;
                idLookup[i] = -1;
        }

        // アプリのアイコンを変更
        HICON hIcon;
        hIcon = (HICON)LoadImage(hThisInst, MAKEINTRESOURCE(
    IDI_ICON1), IMAGE_ICON, 48, 48, 0);
        SendMessage(hWnd, WM_SETICON, ICON_SMALL, (LPARAM)hIcon);

        return hWnd;
}

////////////////////////////
// ツールバー作成用関数
////////////////////////////
HWND CreateToolbarFunc(HWND hWnd, LPARAM lParam) {

        INITCOMMONCONTROLSEX icc{};
        icc.dwSize = sizeof(INITCOMMONCONTROLSEX);
        icc.dwICC = ICC_BAR_CLASSES;
        InitCommonControlsEx(&icc);

        HIMAGELIST g_hImageList = NULL;

        // Declare and initialize local constants
        const int numButtons = 2;
        const int bitmapSize = 26;


        // Create the toolbar
        HWND hToolbar = CreateWindowEx(
                0,
```

```
                TOOLBARCLASSNAME,
                NULL,
                WS_CHILD | WS_VISIBLE | TBSTYLE_LIST,
                0, 0, 0, 0,
                hWnd,
                (HMENU)IDT_TOOLBAR,
                ((LPCREATESTRUCT)(lParam))->hInstance,
                NULL
        );

        if (hToolbar == NULL) {
                return NULL;
        }

        // ツールバーの初期化
        SendMessage(hToolbar, TB_BUTTONSTRUCTSIZE, (WPARAM)sizeof(
TBBUTTON), 0);
        SendMessage(hToolbar, TB_SETBITMAPSIZE, 0, MAKELPARAM(
bitmapSize, bitmapSize));

        // ビットマップの登録
        TBADDBITMAP tb{};
        tb.hInst = NULL;
        tb.nID = (UINT_PTR)LoadBitmap(((LPCREATESTRUCT)(lParam))->
hInstance, MAKEINTRESOURCE(IDB_BITMAP10));
        SendMessage(hToolbar, TB_ADDBITMAP, (WPARAM)numButtons, (
LPARAM)&tb);

        //  ボタンの定義と登録
        TBBUTTON tbb[numButtons] = {
                {0, IDT_TOOLBAR_PEN, TBSTATE_ENABLED,
                BTNS_BUTTON | BTNS_CHECKGROUP,
                {0}, 0, (INT_PTR)_T("ペン")},
                {1, IDT_TOOLBAR_ERASER, TBSTATE_ENABLED,
                BTNS_BUTTON | BTNS_CHECKGROUP,
                {0}, 0, (INT_PTR)_T("消しゴム")}
```

```
        };
        SendMessage(hToolbar, TB_ADDBUTTONS, (WPARAM)numButtons, (
    LPARAM)&tbb);


        return hToolbar;
}

//////////////////////////////
// ウィンドウプロージャの定義
//////////////////////////////
LRESULT CALLBACK WndProc2(HWND hWnd, UINT uMsg, WPARAM wParam,
    LPARAM lParam){
        switch (uMsg) {

        case WM_DESTROY:
                // 後始末
                DeleteObject(hFontUserInfoWindow1);
                DeleteObject(hFontUserInfoWindowRadio);
                DeleteObject(hToolbarBackGroundBrush);
                DeleteObject(hFontBig);
                DeleteObject(hTopComment);

                SelectObject(hMemDC, hOld);
                DeleteDC(hMemDC);

                PostQuitMessage(0);
                return 0;

        case WM_CREATE:
                OnCreate(hWnd, wParam, lParam);

                break;

        case WM_COMMAND:
                OnCommand(hWnd, wParam, lParam);
```

```
            break;

    case WM_TOUCH:
            OnTouch(hWnd, wParam, lParam);
            break;

    case WM_CTLCOLORSTATIC:
            return (OnCtlColorStatic(wParam, lParam));

    case WM_ERASEBKGND:
            // 何も処理しない画面のちらつき防止)(
            return 1;
            break;

    case WM_PAINT:
    {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hWnd, &ps);

            // メモリからにコピー DCDC
            BitBlt(ps.hdc,
                    ps.rcPaint.left, ps.rcPaint.top,
                    ps.rcPaint.right - ps.rcPaint.left, ps.
rcPaint.bottom - ps.rcPaint.top,
                    hMemDC,
                    ps.rcPaint.left, ps.rcPaint.top,
                    SRCCOPY);

            EndPaint(hWnd, &ps);
            return 0;
    }

    case WM_CLOSE:
            if (MessageBox(hWnd, _T("終了しますか？"), _T("終了確
認"), MB_YESNO | MB_ICONQUESTION) == IDYES) {
                    DestroyWindow(hWnd);
            }
```

```
                break;

        default:
                return (DefWindowProc(hWnd, uMsg, wParam, lParam))
    ;
        }
        return(0);
}

//////////////////////////
// 内の処理 WM_TOUCH
//////////////////////////
bool OnTouch(HWND hWnd, WPARAM wParam, LPARAM lParam) {

        UINT cInputs = LOWORD(wParam);
        PTOUCHINPUT pInputs = new TOUCHINPUT[cInputs];
        HPEN hNewPen{};
        int index;

        // ペンの色の選定
        hNewPen = SelectPenColor(hNewPen);

        if (pInputs){
                if (GetTouchInputInfo((HTOUCHINPUT)lParam, cInputs
    , pInputs, sizeof(TOUCHINPUT))){
                        HDC hdc = GetDC(hWnd);
                        HPEN hOldPen = (HPEN)SelectObject(hdc,
    hNewPen);
                        if (hdc) {
                                for (UINT i = 0; i < cInputs; i++)
     {
                                        TOUCHINPUT ti = pInputs[i
    ];
                                        POINT pt = {0, 0};
                                        // タッチポイントを取得
                                        pt.x =
```

```
TOUCH_COORD_TO_PIXEL(ti.x);
                                        pt.y =
TOUCH_COORD_TO_PIXEL(ti.y);
                                        // 絶対座標に変換
                                        ScreenToClient(hWnd, &pt);
                                        index = GetContactIndex(ti
.dwID);
                                        // タッチポイントがタッチパネ
ル内にあるとき
                                        if (PtInRect(&
RectTouchName, pt) || PtInRect(&RectTouchOld_1, pt) || PtInRect
(&RectTouchOld_2, pt) || PtInRect(&RectTouchOld_3, pt)) {
                                                if (ti.dwFlags &
TOUCHEVENTF_DOWN) {
                                                        if (index
< MAX_TOUCH_POINTS && index >= 0) {
                                                                //
タッチポイントの座標を記憶
する

g_ptPrevious[index] = pt;
                                                                //
タッチパネル内であることを
記憶

g_wasinside[index] = true;
                                                        }
                                                }
                                                else if (ti.
dwFlags & TOUCHEVENTF_MOVE) {
                                                        if (index
< MAX_TOUCH_POINTS && index >= 0) {
                                                                if
 (g_wasinside[index]) { // 前回のタッチポイントがタッチパネル内で
あったかのチェック

        POINT ptCurrent = g_ptPrevious[index];
```

```
                // 記憶したタッチポイントから次のタッチポイントまで線を
        引く

                MoveToEx(hdc, ptCurrent.x, ptCurrent.y, NULL);

                LineTo(hdc, pt.x, pt.y);
                                                                }


                g_ptPrevious[index] = pt;

                g_wasinside[index] = true;
                                                                    }
                                                            }
                                                    }
                                                    // タッチポイントがタッチパネ
        ル外にあるとき
                                                    else {
                                                            if (index <
        MAX_TOUCH_POINTS && index >= 0) {

        g_wasinside[index] = false;
                                                            }
                                                            break;
                                                    }
                                            }
                                            // ペンの削除
                                            SelectObject(hdc, hOldPen);
                                            DeleteObject(hNewPen);
                                            // デバイスコンテキストの解放
                                            ReleaseDC(hWnd, hdc);
                                    }
                            }
                            delete[] pInputs;
                            CloseTouchInputHandle((HTOUCHINPUT)lParam);
                    }
                    return true;
            }
```

```
//////////////////////////////////////////////////////////////////
// タッチイベント情報用配列のインデックス番号への変換と取得
//////////////////////////////////////////////////////////////////
int GetContactIndex(int dwID) {
        for (int i = 0; i < MAX_TOUCH_POINTS; i++) {
                if (idLookup[i] == -1) {
                        idLookup[i] = dwID;
                        return i;
                }
                else {
                        if (idLookup[i] == dwID) {
                                return i;
                        }
                }
        }
        // Out of contacts
        return -1;
}
////////////////////
// ペンの色の選定
////////////////////
HPEN SelectPenColor(HPEN hNewPen) {
        // ペンの選定
        switch (PenColor) {
        case Black:
                hNewPen = CreatePen(PS_SOLID, 5, black_pen);
                break;
        case Red:
                hNewPen = CreatePen(PS_SOLID, 5, red_pen);
                break;
        case Blue:
                hNewPen = CreatePen(PS_SOLID, 5, blue_pen);
                break;
        case Yellow:
                hNewPen = CreatePen(PS_SOLID, 5, yellow_pen);
```

```
                break;
        case Purple:
                hNewPen = CreatePen(PS_SOLID, 5, purple_pen);
                break;
        case ESmall:
                hNewPen = CreatePen(PS_SOLID, 4, white_pen);
                break;
        case EMidium:
                hNewPen = CreatePen(PS_SOLID, 12, white_pen);
                break;
        case EBig:
                hNewPen = CreatePen(PS_SOLID, 20, white_pen);
                break;
        default:
                break;
        }
        return hNewPen;
}

////////////////////
//内の処理 WM_CREATE
////////////////////
bool OnCreate(HWND hWnd, WPARAM wParam, LPARAM lParam){
        // ウィンドウをデスクトップ上の中央に移動
        DesktopCenterWindow(hWnd);
        // コントロールの作成
        CreateControl(hWnd, wParam, lParam);
        //名前を付けて保存ダイアログボックスを呼び出す []
        SelectFile(hWnd);
        // メモリデバイスコンテキストの取得
        GetUserInfoWindowMemDC(hWnd);
        // メモリデバイスコンテキストへの描画
        PaintToMemDC(hWnd);
        // フォントの変更
        OnFont();
        // ツールバーの作成
```

```
        //CreateToolbarFunc(hWnd, lParam);

        return true;
}

////////////////////
// ウィンドウの座標を構造体で取得 RECT
// ////////////////////
RECT GetScreenRectFunc(HWND hWnd, RECT rc) {

        POINT topLeft = { rc.left, rc.top };
        POINT bottomRight = { rc.right, rc.bottom };
        ClientToScreen(hWnd, &topLeft);
        ClientToScreen(hWnd, &bottomRight);
        rc.left = topLeft.x;
        rc.top = topLeft.y;
        rc.right = bottomRight.x;
        rc.bottom = bottomRight.y;

        return rc;
}

////////////////////
//内の処理 WM_COMMAND
////////////////////
bool OnCommand(HWND hWnd, WPARAM wParam, LPARAM lParam) {

        switch (LOWORD(wParam)) {
        case ID2_PUSHBUTTON1:
                if (HIWORD(wParam) == BN_CLICKED) {
                        // 指定したウィンドウ領域のキャプチャ
                        GetWindowImage(hWnd);

                        // つ目のウィンドウを呼び出す処理 2
                        ShowWindow(hWnd, SW_HIDE);
                        CreateParentWindow(hWnd, hInst, SW_SHOW,
```

```
_T("Group1"),
                                Winsize.right - Winsize.left,
Winsize.bottom - Winsize.top);
                        // 開始ボタンの有効化
                        EnableWindow(hReserch, TRUE);
                }
                break;
        case ID2_RADIOMEN:
                if (BST_CHECKED == SendMessage(GetDlgItem(hWnd,
ID2_RADIOMEN), BM_GETCHECK, 0, 0)) {
                        strMenLady = _T("男,,,");
                        if (!clickedRadio) {
                                EnableWindow(GetDlgItem(hWnd,
ID2_PUSHBUTTON1), TRUE);
                                clickedRadio = true;
                        }
                }
                break;
        case ID2_RADIOLADY:
                if (BST_CHECKED == SendMessage(GetDlgItem(hWnd,
ID2_RADIOLADY), BM_GETCHECK, 0, 0)) {
                        strMenLady = _T("女,,,");
                        if (!clickedRadio) {
                                EnableWindow(GetDlgItem(hWnd,
ID2_PUSHBUTTON1), TRUE);
                                clickedRadio = true;
                        }
                }
                break;

        case ID_LATEINI:
                LateIniFunc(hWnd, wParam);
                break;

        case ID2_PUSHBUTTONERASE_1:
                if (HIWORD(wParam) == BN_CLICKED) {
```

```
                        // 名前の取り消しボタン
                        EraseFunc(hWnd, RectTouchName);
                }
                break;

        case ID2_PUSHBUTTONERASE_2:
                if (HIWORD(wParam) == BN_CLICKED) {
                        // 年齢の取り消しボタン
                        //EraseFunc(hWnd, RectTouchOld_1);
                        EraseFunc(hWnd, RectTouchOld_2);
                        EraseFunc(hWnd, RectTouchOld_3);
                }
                break;

        ////////////////
        // ツールバー
        ////////////////
        case IDT_TOOLBAR_PEN:
                MessageBox(hWnd, _T("ツールバーがクリックされました。
1"), _T("ボタン 1"), MB_OK);
                break;

        case IDT_TOOLBAR_ERASER:
                MessageBox(hWnd, _T("ツールバーがクリックされました。
2"), _T("ボタン 2"), MB_OK);
                break;

        ////////////////
        // メニューバー
        // //////////////
        // 消しゴム
        case ID_ERASE_SMALL:
                GetClicked(wParam, ESmall);
                break;
        case ID_ERASE_MIDIUM:
                GetClicked(wParam, EMidium);
                break;
```

```
        case ID_ERASE_BIG:
                GetClicked(wParam, EBig);
                break;
                // ペン
        case ID_PEN_BLACK:
                GetClicked(wParam, Black);
                break;
        case ID_PEN_RED:
                GetClicked(wParam, Red);
                break;
        case ID_PEN_BLUE:
                GetClicked(wParam, Blue);
                break;
        case ID_PEN_YELLOW:
                GetClicked(wParam, Yellow);
                break;
        case ID_PEN_PURPLE:
                GetClicked(wParam, Purple);
                break;

        default:
                break;
        }

        return true;
}

//////////////////////
// クリックの検出と色の保存
//////////////////////
UINT GetClicked(WPARAM wParam, UINT Color) {
        if (HIWORD(wParam) == BN_CLICKED) {
                PenColor = Color;
        }
        return PenColor;
}
```

```
//////////////////////
// ビットマップの作成
// //////////////////////
bool CreateBitmap(HDC hdcWindow, int width, int height) {
        HDC hdcMemDC = CreateCompatibleDC(hdcWindow);
        HBITMAP hBitmap = CreateCompatibleBitmap(hdcWindow, width,
     height);
        HGDIOBJ hdcMemDC_old = SelectObject(hdcMemDC, hBitmap);
        return true;
}

//////////////////////
// ウィンドウをキャプチャ
//////////////////////
bool GetWindowImage(HWND hWnd) {

        // キャプチャ対象のウィンドウのを取得 DC
        HDC hdcWindow = GetDC(hWnd);

        // キャプチャ対象のウィンドウのクライアント領域の大きさを取得
        RECT rcClient;
        if (!GetClientRect(hWnd, &rcClient)) {
                return false;
        }
        // クライアント領域のサイズを計算
        int width = rcClient.right - rcClient.left;
        int height = rcClient.bottom - rcClient.top;
        // 年齢のの桁 100
        int width_1 = RectTouchOld_1.right - RectTouchOld_1.left;
        int height_1 = RectTouchOld_1.bottom - RectTouchOld_1.top;
        // 年齢のの桁 10
        int width_2 = RectTouchOld_2.right - RectTouchOld_2.left;
        int height_2 = RectTouchOld_2.bottom - RectTouchOld_2.top;
        // 年齢のの桁 1
        int width_3 = RectTouchOld_3.right - RectTouchOld_3.left;
```

```
    int height_3 = RectTouchOld_3.bottom - RectTouchOld_3.top;

    // 指定された大きさのビットマップを作成
    HDC hdcMemDC = CreateCompatibleDC(hdcWindow);
    HDC hdcMemDC_1 = CreateCompatibleDC(hdcWindow);
    HDC hdcMemDC_2 = CreateCompatibleDC(hdcWindow);
    HDC hdcMemDC_3 = CreateCompatibleDC(hdcWindow);
    HBITMAP hBitmap = CreateCompatibleBitmap(hdcWindow, width,
 height);
    HBITMAP hBitmap_1 = CreateCompatibleBitmap(hdcWindow,
width_1, height_1);
    HBITMAP hBitmap_2 = CreateCompatibleBitmap(hdcWindow,
width_1, height_1);
    HBITMAP hBitmap_3 = CreateCompatibleBitmap(hdcWindow,
width_1, height_1);
    HGDIOBJ hdcMemDC_old = SelectObject(hdcMemDC, hBitmap);
    HGDIOBJ hdcMemDC_old_1 = SelectObject(hdcMemDC_1,
hBitmap_1);
    HGDIOBJ hdcMemDC_old_2 = SelectObject(hdcMemDC_2,
hBitmap_2);
    HGDIOBJ hdcMemDC_old_3 = SelectObject(hdcMemDC_3,
hBitmap_3);

    // 作成したビットマップにキャプチャ内容を描画
    // ウィンドウ全体
    BitBlt(hdcMemDC,
            0, 0, width, height,  // コピー先
            hdcWindow,
            0, 0,                          // コピー元のビットマップ
の左上隅の座標
            SRCCOPY);

    // 年齢桁 100
    BitBlt(hdcMemDC_1,
            0, 0, width_1, height_1,
            hdcWindow,
            RectTouchOld_1.left, RectTouchOld_1.top,
```

```
            SRCCOPY);
    // 年齢桁 10
    BitBlt(hdcMemDC_2,
            0, 0, width_2, height_2,
            hdcWindow,
            RectTouchOld_2.left, RectTouchOld_2.top,
            SRCCOPY);
    // 年齢桁 1
    BitBlt(hdcMemDC_3,
            0, 0, width_3, height_3,
            hdcWindow,
            RectTouchOld_3.left, RectTouchOld_3.top,
            SRCCOPY);

    // クラスのインスタンスを作成 CImage
    CImage image, image1, image2, image3;
    // 作成したビットマップをクラスにロード CImage
    image.Attach(hBitmap);
    image1.Attach(hBitmap_1);
    image2.Attach(hBitmap_2);
    image3.Attach(hBitmap_3);

    // キャプチャした画像の保存先フォルダーの作成
    const char* folderName = _T("Capture");
    if (!CreateCaptureFolder(hWnd, folderName)) {  // が存在しない場合 folderName
            // フォルダーの作成
            CreateDirectory(folderName, NULL);
            // サブフォルダーの作成
            const char* subdirectories[] = { _T("One_digits"), _T("Ten_digits"), _T("Handred_digits"), _T("Window_capture") };
            createSubdirectories(folderName, subdirectories, sizeof(subdirectories) / sizeof(subdirectories[0]));
    }

    // ファイル名の定義
```

```
    std::string filename_100, filename_10, filename_1;
    fileNameCapture = _T("./Capture/Window_capture/capture_0.
jpeg");
    filename_1 = _T("./Capture/One_digits/One_digits_0.jpeg");
    filename_10 = _T("./Capture/Ten_digits/Ten_digits_0.jpeg")
;
    filename_100 = _T("./Capture/Handred_digits/
Handred_digits_0.jpeg");

    for (size_t i = 1; i < 1000; i++){
            if (_access_s(fileNameCapture.c_str(), 0 == ENOENT
)){
                    // FileName does not exist
                    break;
            }
            fileNameCapture = _T("./Capture/Window_Capture/
capture_")
                    + std::to_string(i) + _T(".jpeg");
            filename_1 = _T("./Capture/One_digits/One_digits_"
) +
                    std::to_string(i) + _T(".jpeg");
            filename_10 = _T("./Capture/Ten_digits/Ten_digits_
") +
                    std::to_string(i) + _T(".jpeg");
            filename_100 = _T("./Capture/Handred_digits/
Handred_digits_") +
                    std::to_string(i) + _T(".jpeg");
    }

    // 画像を保存
    HRESULT a, b, c, d;
    a = image.Save(fileNameCapture.c_str(), Gdiplus::
ImageFormatJPEG);
    b = image3.Save(filename_1.c_str(), Gdiplus::
ImageFormatJPEG);
    c = image2.Save(filename_10.c_str(), Gdiplus::
```

```
ImageFormatJPEG);
    d = image1.Save(filename_100.c_str(), Gdiplus::
ImageFormatJPEG);
    if (a != S_OK || b != S_OK || c != S_OK || d != S_OK) {
            MessageBox(hWnd, _T("画像の保存に失敗しました。
"), _T("エラー"), MB_ICONWARNING);
            return false;
    }
    // ビットマップのリサイズ（× 2828)
    //CImage resizedImage = ResizeImage(image, 28, 28);
    // リサイズされたビットマップを torch::に変換 Tensor
    //torch::Tensor reseizedTensor = CImageToTensor(resizedImage);
    //
    // からをデタッチ CImagehBitmap
    image.Detach();
    image1.Detach();
    image2.Detach();
    image3.Detach();
    // リソースを解放
    ReleaseDC(hWnd, hdcWindow);
    // メモリ削除前にメモリを変更前のグラフィックスオブジェクトと関連付
ける DCDC
    SelectObject(hdcMemDC, hdcMemDC_old);
    SelectObject(hdcMemDC_1, hdcMemDC_old_1);
    SelectObject(hdcMemDC_2, hdcMemDC_old_1);
    SelectObject(hdcMemDC_3, hdcMemDC_old_1);

    // 先にビットマップを削除する
    DeleteObject(hBitmap);
    DeleteObject(hBitmap_1);
    DeleteObject(hBitmap_2);
    DeleteObject(hBitmap_3);
    // メモリの削除 DC
    DeleteDC(hdcMemDC);
    DeleteDC(hdcMemDC_1);
    DeleteDC(hdcMemDC_2);
    DeleteDC(hdcMemDC_3);
```

```
        return true;
}

//////////////////
// フォルダの作成
// ////////////////
bool CreateCaptureFolder(HWND hWnd, const char* folderName) {
        // カレントディレクトリのハンドルを取得
        HANDLE hFind;
        WIN32_FIND_DATAA findData;
        char searchPath[MAX_PATH];

        // カレントディレクトリを取得
        GetCurrentDirectory(MAX_PATH, searchPath);
        // 探索パターンを追加
        strcat_s(searchPath, MAX_PATH, "\\*");

        bool folderExists = false;
        // ディレクトリの探索を開始
        hFind = FindFirstFile(searchPath, &findData);

        if (hFind != INVALID_HANDLE_VALUE) {
                do {
                        if (findData.dwFileAttributes &
    FILE_ATTRIBUTE_DIRECTORY) { // ディレクトリかどうかを
    確認
                                if (strcmp(findData.cFileName,
    folderName) == 0) { // ディレクトリ名が一致するかどうか
    確認
                                        folderExists = true;
                                        break;
                                }
                        }
                } while (FindNextFile(hFind, &findData));
                {FindClose(hFind); }// ハンドルを閉じる
        }
```

```
        return folderExists;
}

//////////////////////////
// サブディレクトリの作成
//////////////////////////
bool createSubdirectories(const char* parentPath, const char*
    subdirectories[], UINT numSubdirectories) {
        char subdirectoryPath[MAX_PATH];

        for (UINT i = 0; i < numSubdirectories; i++) {
                snprintf(subdirectoryPath, sizeof(subdirectoryPath
    ), _T("%s\\%s"), parentPath, subdirectories[i]);
                CreateDirectory(subdirectoryPath, NULL);
        }
        return true;
}


//////////////////
//コントロールの作成
//////////////////
bool CreateControl(HWND hWnd, WPARAM wParam, LPARAM lParam) {

        // Push Button Style
        hNameErase = CreateWindowEx(
                WS_EX_WINDOWEDGE,
                _T("BUTTON"),
                _T("取り消し"),
                WS_VISIBLE | WS_CHILD | BS_PUSHBUTTON,
                RectTouchName.right + (Winsize.right -
    RectTouchName.right) / 2 - 75,
                RectTouchName.top + (RectTouchName.bottom -
    RectTouchName.top) / 2 - 50,
                150, 100,
                hWnd, (HMENU)ID2_PUSHBUTTONERASE_1, ((
```

```
LPCREATESTRUCT)(lParam))->hInstance, NULL);

    hOldErase = CreateWindowEx(
            WS_EX_WINDOWEDGE,
            _T("BUTTON"),
            _T("取り消し"),
            WS_VISIBLE | WS_CHILD | BS_PUSHBUTTON,
            RectTouchOld_3.right + (Winsize.right -
RectTouchOld_3.right) / 2 - 75,
            RectTouchOld_3.top + (RectTouchOld_3.bottom -
RectTouchOld_3.top)  /  2  - 50,
            150, 100,
            hWnd, (HMENU)ID2_PUSHBUTTONERASE_2, ((
LPCREATESTRUCT)(lParam))->hInstance, NULL);

    hButtonNext = CreateWindowEx(
            WS_EX_WINDOWEDGE,
            _T("BUTTON"),
            _T("次の画面に進む＞"),
            WS_VISIBLE | WS_CHILD | BS_PUSHBUTTON |
WS_DISABLED,
            Winsize.right / 2 - 200, Winsize.bottom - 72, 400,
 55,
            hWnd, (HMENU)ID2_PUSHBUTTON1, ((LPCREATESTRUCT)(
lParam))->hInstance, NULL);

    // Static Style

    /*hStaticUserInfo = CreateWindow(
      _T("STATIC"),
      _Tユーザー情報の入力 (""),
      WS_CHILD | WS_VISIBLE | SS_LEFT,
      RectTouchName.left, RectTouchName.top − 145, 800, 50,
      hWnd, (HMENU)ID2_STATICUSERINFO,
      ((LPCREATESTRUCT)(lParam))−>hInstance, NULL
 );*/
```

```
    hStaticCombo = CreateWindow(
            _T("STATIC"),
            _T("遅延時間の設定:"),
            WS_CHILD | WS_VISIBLE | SS_LEFT,
            Winsize.right - 300 - 260, RectTouchS.top + 65,
260, 70,
            hWnd, (HMENU)ID2_STATICCOMBO,
            ((LPCREATESTRUCT)(lParam))->hInstance, NULL
    );

    hStaticComment1 = CreateWindow(
            _T("STATIC"),
            _T("名前・年齢・性別を回答してください。"),
            WS_CHILD | WS_VISIBLE | SS_LEFT,
            RectTouchName.left, RectTouchName.top -80 + 20,
800 ,  50 - 10,
            hWnd, (HMENU)ID2_STATICCOMMENT1,
            ((LPCREATESTRUCT)(lParam))->hInstance, NULL
    );

    hStaticName = CreateWindow(
            _T("STATIC"),
            _T("名前:"),
            WS_CHILD | WS_VISIBLE | SS_LEFT,
            30, RectTouchName.top + 65, 100, 60,
            hWnd, (HMENU)ID2_STATICNAME,
            ((LPCREATESTRUCT)(lParam))->hInstance, NULL
    );

    hStaticOld = CreateWindow(
            _T("STATIC"),
            _T("年齢:"),
            WS_CHILD | WS_VISIBLE | SS_LEFT,
            30, RectTouchOld.top + 65, 100, 60,
            hWnd, (HMENU)ID2_STATICOLD,
```

```
                ((LPCREATESTRUCT)(lParam))->hInstance, NULL
        );

        hStaticS = CreateWindow(
                _T("STATIC"),
                _T("性別:"),
                WS_CHILD | WS_VISIBLE | SS_LEFT,
                30, RectTouchS.top + 65, 100, 60,
                hWnd, (HMENU)ID2_STATICS,
                ((LPCREATESTRUCT)(lParam))->hInstance, NULL
        );

        // Radio Button Style
        hRadioMen = CreateWindow(
                _T("BUTTON"),
                _T("男性"),
                WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON |
SS_CENTER,
                RectTouchS.left, RectTouchS.top + 55, 100, 60,
                hWnd, (HMENU)ID2_RADIOMEN,
                ((LPCREATESTRUCT)(lParam))->hInstance, NULL
        );

        hRadioLady = CreateWindow(
                _T("BUTTON"),
                _T("女性"),
                WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON |
SS_CENTER,
                RectTouchS.left+ 150 , RectTouchS.top + 55, 100,
60,
                hWnd, (HMENU)ID2_RADIOLADY,
                ((LPCREATESTRUCT)(lParam))->hInstance, NULL
        );

        ComboboxFunc(hWnd, lParam);             // 遅延時間設定用
```

```
        // コンボボックスの先頭項目の文字列の長さを取得
        int intTxtLen = SendMessage(GetDlgItem(hWnd, (int)
    ID_LATEINI), CB_GETLBTEXTLEN, 0, 0);
        char* pszBuf = new char[intTxtLen + 1];
        // コンボボックスの先頭項目の文字列を取得
        if (SendMessage(GetDlgItem(hWnd, (int)ID_LATEINI),
    CB_GETLBTEXT, 0, (LPARAM)pszBuf) != CB_ERR) {
                char Path[MAX_PATH + 1];
                char settingpath[MAX_PATH + 1]{};
                settingpath[0] = '\0';
                if (0 != GetModuleFileName(NULL, Path, MAX_PATH))
    {
                        char drive[MAX_PATH + 1], dir[MAX_PATH +
    1], fname[MAX_PATH + 1], ext[MAX_PATH + 1];
                        _splitpath_s(Path, drive, sizeof(drive),
    dir, sizeof(dir),
                                fname, sizeof(fname), ext, sizeof(
    ext));
                        _stprintf_s(settingpath, MAX_PATH + 1, _T(
    "%s%ssetting.ini"), drive, dir);
                }
                // ファイルから選択したキーの遅延時間を取得しに保存
    inilatedata
                GetPrivateProfileString(pszBuf, _T("data"), _T("
    error"), latedata, sizeof(latedata), settingpath);
        }
        delete[] pszBuf;

        return true;
}

//////////////////////////
// 内の処理 WM_CTLCOLORSTATIC
//////////////////////////
long OnCtlColorStatic(WPARAM wParam, LPARAM lParam) {

        int i = GetWindowLong((HWND)lParam, GWL_ID);
```

```
        if (i == 0) {
                return -1;
        }else {
                if ((i == ID2_STATICUSERINFO) || (i ==
    ID2_STATICCOMMENT1)) {
                        SetBkMode((HDC)wParam, TRANSPARENT);
                        SetTextColor((HDC)wParam, RGB(30, 144,
    255));
                        return (long)BackGround_clear;
                }
                else if (i == IDT_TOOLBAR){
                        SetBkMode((HDC)wParam, TRANSPARENT);
                        return (long)hToolbarBackGroundBrush;
                }else {
                        SetBkMode((HDC)wParam, TRANSPARENT);
                        return (long)BackGround_clear;
                }
        }
}

////////////////////////////////////////////////
// メモリデバイスコンテキスト描画の事前準備
////////////////////////////////////////////////
bool GetUserInfoWindowMemDC(HWND hWnd) {

        HDC hDC;
        HBITMAP hBitmap;

        // デバイスコンテキストの取得
        hDC = GetDC(hWnd);
        // メモリデバイスコンテキストの取得
        hMemDC = CreateCompatibleDC(hDC);
        // ビットマップハンドルの取得
        hBitmap = CreateCompatibleBitmap(hDC, Winsize.right,
    Winsize.bottom);
        // デバイスコンテキストの解放
```

```
        ReleaseDC(hWnd, hDC);
        // メモリにビットマップを割りつけ DC
        hOld = SelectObject(hMemDC, hBitmap);
        // ビットマップの削除
        DeleteObject(hBitmap);

        return true;
}

//////////////////////////////////////
// メモリデバイスコンテキストへの描画
//////////////////////////////////////
bool PaintToMemDC(HWND hWnd) {
        // 背景の塗りつぶし
        FillRect(hMemDC, &Winsize, BackGround_clear);
        // 名前入力用タッチパネル
        FillRect(hMemDC, &RectTouchName, (HBRUSH)CreateSolidBrush(
    RGB(255, 255, 255)));
        // 年齢入力用タッチパネル
        //FillRect(hMemDC, &RectTouchOld, (HBRUSH)CreateSolidBrush(
    RGB(255, 255, 255)));
        // 年齢入力用タッチパネル の桁 100
        FillRect(hMemDC, &RectTouchOld_1, (HBRUSH)CreateSolidBrush
    (RGB(255,255,255)));
        // 年齢入力用タッチパネル　の桁 10
        FillRect(hMemDC, &RectTouchOld_2, (HBRUSH)CreateSolidBrush
    (RGB(255, 255, 255)));
        // 年齢用入力用タッチパネル の桁 100
        FillRect(hMemDC, &RectTouchOld_3, (HBRUSH)CreateSolidBrush
    (RGB(255, 255, 255)));
        // ボタンの背景色の描画
        FillRect(hMemDC, &FrameBottom, (HBRUSH)CreateSolidBrush(
    RGB(70, 130, 180)));

        // 枠線の描画
        //DrawEdge(hMemDC, &RectTouchTop, EDGE_BUMP,
```

```
    BF_BOTTOM);
        DrawEdge(hMemDC, &RectTouchName, EDGE_BUMP, BF_RECT);
        //DrawEdge(hMemDC, &RectTouchOld, EDGE_BUMP, BF_RECT);
        DrawEdge(hMemDC, &RectTouchOld_1, EDGE_BUMP, BF_RECT);
        DrawEdge(hMemDC, &RectTouchOld_2, EDGE_BUMP, BF_RECT);
        DrawEdge(hMemDC, &RectTouchOld_3, EDGE_BUMP, BF_RECT);
        return true;
}

bool EraseFunc(HWND hWnd, RECT rc){
        HDC hDC = GetDC(hWnd);
        FillRect(hDC, &rc, (HBRUSH)CreateSolidBrush(RGB(255, 255,
    255)));
        DrawEdge(hDC, &rc, EDGE_BUMP, BF_RECT);

        return true;
}
////////////////////
//フォントの変更
////////////////////
bool OnFont() {
        // 新しく型変数を追加する場合は、メッセージ内での
    HFONTWM_DESTROYDeleteObject(HFONT)を忘れないように。

        hFontUserInfoWindow1 = CreateFont(
                40, 0, 0, 0,
                FW_MEDIUM,
                false, false, false,
                SHIFTJIS_CHARSET, OUT_DEFAULT_PRECIS,
                CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                (VARIABLE_PITCH | FF_DONTCARE), _T("メイリオ"));

        hFontUserInfoWindowRadio = CreateFont(
                50, 0, 0, 0,
                FW_MEDIUM,
                false, false, false,
                SHIFTJIS_CHARSET, OUT_DEFAULT_PRECIS,
```

```
                CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                (VARIABLE_PITCH | FF_DONTCARE), _T("メイリオ"));

        hFontBig = CreateFont(
                50, 0, 0, 0,
                FW_BOLD,
                false, false, false,
                SHIFTJIS_CHARSET, OUT_DEFAULT_PRECIS,
                CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                (VARIABLE_PITCH | FF_DONTCARE), _T("メイリオ"));

        hTopComment = CreateFont(
                40, 0, 0, 0,
                FW_BOLD,
                false, false, false,
                SHIFTJIS_CHARSET, OUT_DEFAULT_PRECIS,
                CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                (VARIABLE_PITCH | FF_DONTCARE), _T("メイリオ"));

        // To Static
        SendMessage(hStaticName, WM_SETFONT, (WPARAM)
    hFontUserInfoWindow1, MAKELPARAM(false, 0));
        SendMessage(hStaticOld, WM_SETFONT, (WPARAM)
    hFontUserInfoWindow1, MAKELPARAM(false, 0));
        SendMessage(hStaticS, WM_SETFONT, (WPARAM)
    hFontUserInfoWindow1, MAKELPARAM(false, 0));
        SendMessage(hStaticComment1, WM_SETFONT, (WPARAM)hFontBig,
     MAKELPARAM(false, 0));
        //SendMessage(hStaticUserInfo, WM_SETFONT, (WPARAM)
    hFontBig, MAKELPARAM(false, 0));
        SendMessage(hStaticCombo, WM_SETFONT, (WPARAM)
    hFontUserInfoWindow1, MAKELPARAM(false, 0));

        // To Push Button
        SendMessage(hButtonNext, WM_SETFONT, (WPARAM)
    hFontUserInfoWindow1, MAKELPARAM(false, 0));
```

```
        SendMessage(hNameErase, WM_SETFONT, (WPARAM)
    hFontUserInfoWindow1, MAKELPARAM(false, 0));
        SendMessage(hOldErase, WM_SETFONT, (WPARAM)
    hFontUserInfoWindow1, MAKELPARAM(false, 0));

        // To Radio Button
        SendMessage(hRadioMen, WM_SETFONT, (WPARAM)
    hFontUserInfoWindowRadio, MAKELPARAM(false, 0));
        SendMessage(hRadioLady, WM_SETFONT, (WPARAM)
    hFontUserInfoWindowRadio, MAKELPARAM(false, 0));

        // To ComboBox
        SendMessage(latedatacombo, WM_SETFONT, (WPARAM)
    hFontUserInfoWindow1, MAKELPARAM(false, 0));

        return true;
}


////////////////////
//ビットマップをの型に変換する torchTensor
////////////////////
//torch::Tensor CImageToTensor(const CImage& image)
//{
//    int width = image.GetWidth();
//    int height = image.GetHeight();
//    int channels = image.GetBPP() / 8;
//
//    torch::Tensor tensor = torch::zeros({ 1, 1, height, width });
//    auto tensor_accessor = tensor.accessor<float, 4>();
//
//    for (size_t y = 0; y < height; ++y)
//    {
//        for (size_t x = 0; x < width; ++x)
//        {
//            COLORREF pixel = image.GetPixel(x, y);
```

```
//                  float grayscale_value = (GetRValue(pixel) + GetGValue(
    pixel) + GetBValue(pixel)) / (3.0 * 255);
//                  tensor_accessor[0][0][y][x] = grayscale_value;
//              }
//      }
//
//      return tensor;
//}

////////////////////
//ビットマップのリサイズ
////////////////////
//CImage ResizeImage(const CImage& inputImage, int newWidth, int
    newHeight) {
//
//      CImage resizedImage;
//      resizedImage.Create(newWidth, newHeight, inputImage.GetBPP());
//
//      HDC hdcResized = resizedImage.GetDC();
//      HDC hdcInput = inputImage.GetDC();
//
//      SetStretchBltMode(hdcResized, HALFTONE);
//      StretchBlt(hdcResized, 0, 0, newWidth, newHeight, hdcInput,
//              0, 0, inputImage.GetWidth(), inputImage.GetHeight(), SRCCOPY
    );
//
//      inputImage.ReleaseDC();
//      resizedImage.ReleaseDC();
//
//      return resizedImage;
//}
```