アイコンは投げられるか？ [0]

久野　靖[†]、角田博保[‡]、大木敦雄[†]、粕川正充[†]

[†]筑波大学大学院経営システム科学専攻
[‡]電気通信大学情報工学科

概要

現在の図的ユーザインタフェース (Graphical User Interface, GUI) の大部分は、アイコンをマウスで選択した後、メニューやボタンなどを用いてそのアイコンが表す実体に対する操作を指定するようになっている。この方法は確かに判りやすいが、操作を完了するまでに要する時間は必ずしも短くなく、大量の操作を行うといらいらする。ところで、一部のユーザインタフェースでは上述のものに加えてドラグ & ドロップと呼ばれる操作を許すようになっている。ドラグ & ドロップは、操作の対象となるアイコンをマウスで「摑んだ」まま操作を表すアイコンまで移動して重ねることにより操作を指定する。この方法は操作の手順が「摑む = 移動 = 離す」だけで単純であるため、他の方法に比べて楽に操作できる可能性を秘めている。しかし、実際にはドラグ & ドロップの操作に要する時間は他の方法と比べて大差無い。これはマウスを移動していって重ねるのにかなりの時間が掛かるためである。そこで筆者らは、対象アイコンを目的アイコンの所まで「持って行く」代わりに「投げる」(ドラグしながらボタンを離す) 方法を用いればより効率的な操作が可能なのではないかと考え、予備的な実験を行った。その結果、「投げる」インタフェースはメニュー、パレット、ドラグ & ドロップなど他の方法より高速であり、図的ユーザインタフェースの基本操作として有望であることが明らかになった。

1　はじめに

近年において著しい変化と発展が見られた分野の 1 つとして、計算機システムのユーザインタフェースの分野が挙げられる。具体的には、高性能の CPU と高解像度の表示装置を持った個人用計算機システムを占有使用することが一般的となり、その結果図的ユーザインタフェース (Graphical User Interface、GUI) と呼ばれる形態のものが広く用いられるようになった。

GUI が旧来の画面への文字表示とキーボードによるインタフェースと異なっている点としては、次のものが挙げられる。

(a) 画面上に様々な種類の文字や図形、絵などを様々な大きさ、色を用いて表示できる。
(b) それら表示されたものをポインティングデバイスによって直接指定できる。
(c) 実行すべき操作もポインティングデバイスを用いて様々な形で指定できる。
(d) 複数の作業領域を同時に画面上に表示させたまま並行して作業を進めることができる。

そして、絵やテキストなど直接画面に表示させるのに適したものだけでなく、「ファイル」「ディレクトリ」「プリンタ」など計算機が扱うあらゆる対象についても、それをアイコンと呼ばれる図形に対応させることによって GUI を通じた操作が可能になっている

一般に使われている GUI の多くは複数の作業領域をそれぞれ「窓」と呼ばれる矩形の領域によって表し、作業対象をアイコンによって表し、操作の選択にはお

[0] "Icon Throwing" User Interface by Yasushi KUNO[†], Hiroyasu KAKUDA[‡] Atsuo OHKI[†], and Masaatsu KASUKAWA[†], [†]Graduate School of Systems Management, The University of Tsukuba, Tokyo and [‡]Department of Computer Science, University of Electro-Communications.

もにメニューを用い、ポインティングデバイスによって操作することからWIMP (Windows, Icons, Menus, Pointing devices) インタフェースと呼ばれる [1]。その利点としては、次のものが挙げられる。

- アイコンで「対象」、メニューで「操作」を選択するという枠組みは非常に汎用的であり、どんな作業にでも適用できる。
- アイコンを用いることで、利用者が対象物を直接操作しているような感覚を持つことができるので、利用者にとってわかりやすく、安心感がある。
- メニューにより選択肢が計算機側から提示されるため、利用者にとってわかりやすく、選択肢を覚える負担もない。

しかし一方で、実際にWIMPインタフェースを操作して多量の作業を行うと、繰り返しアイコンを選択してはメニューで項目を選ぶのが煩わしく感じられることも事実である。たとえばUnixに習熟したユーザの場合、GUIが利用可能であっても大多数が端末窓に向かって旧来のシェルコマンドを打ち込んで使っているのは、上記のことも1つの理由となっているのではないかと筆者らは考えている。

ところで、旧来の文字端末とキーボードを使用したユーザインタフェースでも、次の様ないくつかの枠組みが存在した。

- 指令を文字列として打ち込む。
- 画面に書式を表示させ、空欄を打ち込んで埋める。
- メニューを表示して選択させる。

これらは上の方のものほど1画面(ないし1区切りの操作)で入力できる情報量が大きく高速に操作できるが、その代わりコマンドやオペランド、ないし各欄に打ち込める内容を記憶するといった訓練が必要となる。逆に下の方に行くほどわかりやすく初心者でも使えるが、一定の情報を入力するのに時間が掛かる。

これを言い替えれば、初心者にとってのわかりやすさ、訓練の不要さと熟練者による操作の高速さはトレードオフの関係にあると見ることもできる。そして、同様のトレードオフがGUIについても存在しそうなことは容易に想像できる。

もしそうだとすれば、現在のWIMPインタフェースはメニューを中心としている以上、どちらかといえば「訓練の不要さ」に寄ったものであり、高速な操作性を求める熟練ユーザには向かないものになっている可能性がある。Unixユーザがおもに端末窓を使っているという観察はこれを裏付けるものである [2]。

しかし、このことから直ちにGUIは熟練者向けでないと結論することはできない。GUIの歴史はごく浅いものであり、一方その自由度はとても大きい。したがって、まだ試されていない / 考案されていない新しいインタフェースの枠組みが多数あるはずである。

以上の考えに基づき、筆者らは「初心者に厳しい」GUIの枠組みについて検討し、その中から1つの可能性として「投げるインタフェース」を取り上げ、その性能を実験的に評価してみた。

以下第2章ではメニューを代表とする既存の操作選択方式について考察しする。続いて第3章で「投げる」インタフェースを提案し、その有効性を評価するために行った実験の結果と考察を述べる。最後に第4章でまとめを行う。

2 既存の操作選択方式とその比較

2.1 操作選択方式について

ここでいう「操作選択方式」とは、まず操作の対象(アイコンや特定の図形、テキストの一部分など)をポインティングデバイスで指定した後、その対象に適用すべき操作を計算機に伝える方法を指す。(まず適用すべき操作を伝えた後で対象を指定する方法もあるが少数派であり、一般には対象を先に指定するほうが使いやすいとされている。)

なお、ポインティングデバイスとしては電子ペンやトラックボールなども一般的であるが、以下では最も多く使われているマウスを前提として話を進める。

具体的な操作選択方式としては、次のものが一般的である。

- マウスボタン (+ 修飾キー)
- メニュー
- パレット
- キーボード
- ドラグ & ドロップ

以下ではこれら各々について比較検討する。

2.2 マウスボタン

すべてのマウスにはボタンが備わっており、これによって操作対象を選択する。加えて、2ボタン、3ボタンなど複数のボタンを備えたマウスも多く使われて

いる。そのような場合には、例えばどのボタンを押すかによって異なる操作が選択されるように割り当てることが考えられる。また、Machintosh のようにマウスボタンを連続して押したり(ダブルクリック)、または修飾キー(Shift、Option など直接文字に対応しないキー)を押しながらマウスボタンを押すことにより複数の操作を区別するシステムも多い。

マウスボタン(と修飾キー)による選択は、操作対象の指示と操作選択が1動作でできるため、所用時間の短さという点では最も優ると予想される。しかし一方で、各機能をどの操作手順(どの修飾キーとともに、どのマウスボタンを、どのように押すか)に対応させるかを記憶し、正しく実行することは利用者にとって極めて負担が大きい。これはボタンや修飾キーの組合せが(キーボードのキーと違って)自然なニモニックを持たないことにも起因していると思われる[1]。

このため、この方法は頻繁に使われる少数の汎用的な操作に限って用い、それ以外は他の選択方法に頼るというのが一般的である。例えば Machintosh ではダブルクリックは多くの場面で「開く」という操作に対応させられている。また SunView、idraw など多くの X ウィンドウ上のシステムでは中央ボタンを「その対象の移動」に割り当てている。

2.3 メニュー

本稿ではメニューを「普段は見えないが、何らかの操作によって画面に表示され、項目の選択を許すような機構」の意味で用いる。メニューは操作選択方式の中では最も一般的で広く使われる。その理由としては次のものが考えられる。

- 普段は見えないので、画面上の領域を浪費しない。
- 逆に、表示させた時には情報の表示に十分な領域を使うことができる。
- そのため、選択肢が何であるかを示す情報を表示でき、利用者にとってわかりやすく安心感がある。
- 表示が起きる際の状況に適合するようにそのつど構成を変化させることも可能である。

メニューにはいくつかのバリエーションがある。

まず、ポップアップメニューは画面上の任意の(通常はマウスカーソルの)場所に現れるメニューである。その利点は、メニューがカーソルの場所に現れるので、選択のためにマウスを動かす距離が小さいことである。半面、対象指定の操作とメニュー表示の操作を区別する方法が別途必要だという弱点を持つ。その方法としてはマウスのボタンによって区別したり、修飾キーによって区別するなどが一般的である。これをさらに進めて複数のメニューを使い分けるシステムもあるが、あまり多くのメニューを使い分けることは前節に述べた理由から困難である。

一方、プルダウンメニューは画面上にメニューを表示するための領域(メニューボタンと呼ぶ)が用意され、マウスカーソルをそこに持って行ってマウスボタンを押すとメニューが表示されるものをいう。Macintosh をはじめとして広く採用されている方法は、メニューボタンを複数個、窓や画面の上の方に一列に並べ(これをメニューバーと呼ぶ)、メニューボタン上でマウスボタンを押すとメニューバーの下にメニューが現れるというものである。(従って「引き下ろす」ように見えるためプルダウンメニューと呼ぶ。)その他、メニューバーを下に置いたり、画面状の対象物や孤立したメニューボタンからメニューが出るやり方もあるが、ここではそれらももまとめてプルダウンメニューと呼ぶ。プルダウンメニューの利点は、ポップアップメニューのように「区別のための操作」を必要とせず、そのため多数のメニューを用意できることである。その代わり、メニューボタンまでマウスを移動させる手間が余分に掛かる。

ポップアップ / プルダウンのどちらでも、メニューの各項目はメニュー内の区切られた領域として表され、いずれかの領域でボタンを離すことによってその項目を選択する。領域が(中に文字を表示するため)横長の矩形でメニューはそれが縦につらなった形をとるものが多いが、それ以外の形もある。(例えば正方形の領域がマトリクス状に並んだものや、マウスカーソルを中心とした扇型で全体として円形になったものなど。)

1つのメニューに含められる項目にはおのずと限界がある。あまり多くの項目を含めると、メニューが物理的に画面に入り切らなくなったり、そうならないまでも離れた項目を選ぶのに(目で探すのにもそこまでマウスを移動するのにも)時間が掛かり過ぎる。場合によってはメニューの中身を上下にスクロールする方法も使われるが、これだと探すのはさらに大変になる。

そこで、項目の上にマウスが来るとそこに隣接して「子供の」メニュー(サブメニュー)を表示する、とい

[1] これに関連して、マウスボタンでモールス信号を送るならそれは自然なニモニックを持てるというアイデアを考えたので、簡単な実験を行った(付録参照)。

う方式も多く使われる。これをカスケードメニューと呼ぶ。カスケードメニューはメニューによる選択そのものが階層的な構造を持つ場合に適した方法である。

これらのバリエーション全体を通して、メニューの弱点は選択までに

1. メニューを出すための動作を行う。
2. メニューが出たことを認識する。
3. 選択したい場所までカーソルを移動する。
4. 選択動作(通常はボタンを離す)を行う。

という段階が必要で、これが複雑なために多くの時間を必要とするということである。さらに、メニューは表示されるまでは「どこに目的の選択位置があるかわからない」ので、他のマウス操作に多く見られる「マウスを移動しながら目的対象を探す」というオーバーラップも起こりにくい。

2.4 パレット

前述のメニューの弱点を避けるために、選択対象を常時画面に表示しておき、マウスカーソルをその位置に持って行ってマウスボタンを押す(または押して離す)ことで操作を選択することも多く行われる。この場合には選択のプロセスは

1. 選択位置までカーソルを移動する。
2. 選択動作を行う。

だけで、メニューの場合よりずっと簡単である。(プルダウンメニューのメニューバーは「表示するメニューを選択する」ためにこのような機構を採用しているとも見なせる。)

この方式の場合もメニューと同様、いくつかの異なる表現形態を取る。

- 押しボタン: 操作盤のようなものの上にボタンがならんでいて、そのどれかを押すというイメージのもの。
- パレット: お絵描きツールに多く見られ、絵の具のパレットのようにます目の中に「色」や「絵筆の種類」が表示されていてどれかを選択する。
- 引きちぎりメニュー: もとはプルダウンメニューなのだが、それを「引きちぎって」画面の好きな場所に置いておき、あとで直接クリックできる。

以下ではこれらを代表して「パレット」という用語を用いる(マウスボタンとの混同を避けるため)。

パレットの長所は上述のように、表示されるのを待たなくてもよく、操作手順が単純なことである。しかし一方、次の様な弱点もある。

- 常に画面上に場所を占めるため、選択肢が増えると場所を占有する。
- それに対処して1項目の大きさを小さくすると選択位置にマウスカーソルを置くのに時間が掛かるようになる。
- 選択肢が増えると選択対象を探すのが大変になる。
- 自然な配置は直線かマトリクス状なので、メニューの様に階層化を現すのが難しい。
- ポップアップメニューと異なり、常に画面のはじまでマウスを移動して行く必要がある。

2.5 キーボード

キーボードは物理的には多数のボタンが並んだものであり、これを操作選択に使うことは自然である。ただし、どのキーがどの操作選択に対応するかを覚えなければならないという負担はある。これに対しては、メニューの一部に「この選択と同等の操作を現すキー」を表示しておき、繰り返しメニューを使っているうちに対応するキーを覚える、という方式が一般的である。これを「キーボードショートカット」と呼ぶ。

ここで注意すべきなのは、キーはあくまでもマウスで操作対象を選択した後、その対象についての操作を指定するのに使っているということである。そのため、コマンドをフルスペルで打ち込むようなことはまず考えられない(そうするとマウスから手を離して操作しなければならず、それくらいならオペランドまですべてコマンドで打ち込む方がよさそうである)。そこで、キーによる指定は常に1打鍵(ただし修飾キーはついてもよい)とするのが普通である。

キーによる操作指定はボタンよりさらに簡単であり、

1. 問題のキーを押す。

で済んでしまう上、マウスを移動する必要もない。このためキーによる操作指定は高速に行える。そのキーがマウスを持っていない手の受け持ち範囲であれば特にそういえる。

一方キーの弱点としては、操作の数が多くなってくると、それらを覚えるのが負担になることが挙げられる。(1打鍵が基本なので、ニモニックを割り当てるのが難しい。)さらに、修飾キーを押すのと押さないので全く違う操作を割り当てるのは混乱をもたらすので

事実上困難である。また、右手マウスの人を考慮してキーを割り当てると左手マウスの人にとっては不利になるという難しさもある。

2.6 ドラグ & ドロップ

ドラグ & ドロップはアイコン自身のみによって操作を指定するという点でこれまで述べてきた操作選択とは異なっている。すなわち、操作対象も操作自体もともにアイコンとして現し、対象アイコンの上でマウスボタンを押し、その状態で操作アイコンのところまで移動して来て (drag)、重ねた状態でマウスボタンを離す (drop) することで操作を起動する。これは、任意の抽象的な「もの」を現すことができるというアイコンの特徴を活かしたやりかただと考えられる。

ドラグ & ドロップの操作系列は

1. 対象アイコンの上でマウスボタンを押す。
2. 操作アイコンの上までそのまま移動する。
3. マウスボタン離す。

であり、1は対象アイコンを選択する操作を兼ねているため残りの2と3によって操作が選択できることになる。このため手順としては非常に単純である。また、メニューと異なり、操作の選択肢を表示する手間は不要であり、パレットと異なり画面上に別の領域を必要とすることもない (操作アイコンの置き場所は必要である。)

もう1つの特徴として、操作アイコン自体もアイコンであるから、それを好きな位置に配置したり機能を変更するなどの「カスタマイズ」が同じ枠組み中で行えることがある。(これまでに挙げた他の方式では、項目の配置や選択のためのキーの変更といった簡単なカスタマイズでも設定ファイルをエディタで編集するのが普通である。) そのようなユーザインタフェースの試みについては、以前にも報告した [3]。

一方、ドラグ & ドロップにも次のような弱点がある。

- どれが「操作」でどれが「対象」であるか、慣れるまでは混乱しやすい。
- ドラグしていって操作アイコンに重ねる、という操作はそれ自体としては単純だが、思ったより時間が掛かる。

このうち前者については、これまでは頻繁に使う少数の汎用的な操作のみをドラグ & ドロップで指定させるという方策が取られて来た。具体的な例としては次のものがある。

- ファイルやフォルダ (ディレクトリ) をあらわすアイコンを、フォルダやごみ箱といった容れものをあらわすアイコンに重ねることでその中に移す。
- その類推として、ファイルをプリンタアイコンに重ねることでプリンタに打ち出す。
- 文書ファイルやデータファイルを現すアイコンを、それを扱うプログラムのアイコンに重ねることでプログラムを起動する。

しかしこれは要は慣れの問題であり、より広範な操作をドラグ & ドロップにより行わせることも十分有効であると筆者らは考えている。

一方、後者の「重ねる」操作に掛かる時間については人間の運動能力から来る制約であり、簡単に改良することはできない。この点について次節で検討する。

3 「投げる」インタフェースとその評価

3.1 マウス移動操作時間と「投げる」操作の提案

Card らの研究に基づくキーストロークレベルモデル (以下 KL モデル) [4] によれば、マウスを特定の目標に移動させるのに掛かる時間 (ミリ秒) は移動距離を d、目標の大きさを s とした時次の式で表される。

$$T_p = T_W + I_M \, log_2(\frac{d}{s} + 0.5)$$

ここで T_W はマウス操作に付随するオーバヘッドで値は 800msec、I_M は人によって異なるが 100msec 程度の値の定数となっている。

しかし、このモデルはキーボードとマウスの操作が適宜混ざった環境での実験に基づいて構成されたと思われ、予備的な計測の結果、今回のようにずっとマウスを持ったまま操作する場合には特に T_W の値が過大であるように思われた。そこで後述する実験のためのプログラムを応用して、様々な距離で並んだアイコンを順次クリックして行く実験を行い、そこから逆算して T_W の値を求めてみた。その結果に基づき、以下では 300msec という値を使用する。

この値を使用したとしても、マウスの移動時間はおおよそ 350 ～ 750msec となり、これにマウスボタンを押す時間 (キーボードの打鍵時間と同じと考える) を加えると、全体の時間はキーボード打鍵時間 (100 ～ 300msec 程度) の 3 ～ 5 倍とかなり大きくなる。

そして、キーボードの打鍵時間は熟練するにつれて短くなるが、マウスの移動時間は大きく改善されるこ

とはない。これは、手の動作→マウスの移動→アイコンの移動→目によるフィードバック→手の動作 (の調整) …というサイクルを一定回数反復することが避けられない以上やむを得ないことである。そして、メニュー、パレット、ドラグ & ドロップのいずれもがこのマウスによる位置指定を必要とする。多量の操作をしていて「いらいらする」という感覚を持つのはそのせいもありそうである。

ところで、ドラグ & ドロップのためのマウス操作に限って考えてみると、テキストの編集位置や図形の位置を指定する場合と異なり、要は「どれに重ねようとしているのか」さえ計算機に伝わればよいはずである。そう考えてみると、実際に対象アイコンが目標アイコンに重なるまで待たなくても、対象アイコンが動き始めればその経路を延長することでどの目標アイコンをめざしているかは判断できる (経路上に複数のアイコンがあるような場合は後で扱う)。

そのような方式を取った場合、利用者の操作はドラグ & ドロップの前半だけ、すなわち対象アイコンの上でマウスボタンを押し、そのまま移動を開始し、移動の途中でボタンを離す、というものになる。これを我々は「アイコン投げ」と名付けた。

アイコンを投げる場合にどれくらい時間が掛かるかは KL モデルでは計算できないが、予想としては「動き出せばよい」のだから、目標アイコンまでの距離に依存せずに済み、なおかつ通常のマウス操作で時間が掛かるのは目標位置に正確に合わせるための微調整であるので、それがない分大幅に速くなるのではないか、と考えた。

3.2 実験による「投げる」インタフェースの評価

実際にアイコン投げを実現する簡単なデモプログラムを X-Window 上で作成して操作してみたところ、簡単な測定の結果では通常のドラグ & ドロップの半分近くの操作時間で済み、主観的にも操作しやすいという結果を得た。そこで、このプログラムをもとに、メニューやドラグ & ドロップなど通常の操作方式も行える実験用プログラムを用意し、筆者らが被験者となって各種選択方式の比較を行う予備的実験を行った。

実験には IBM-PC 互換機 (飯山電機製、486DX66、メモリ 8MB、ISA バス、17 インチカラーディスプレイ MF-8217J、機械式 3 ボタンマウス) を使用し、OS は BSD/386 1.1、X サーバは BSD/386 付属の X11R5 サーバを解像度 1152 × 900 ドットとして使用した。実験プログラムは C++ で約 900 行あり、API は Xlib を使用し、BSD/386 付属の g++ により翻訳した。

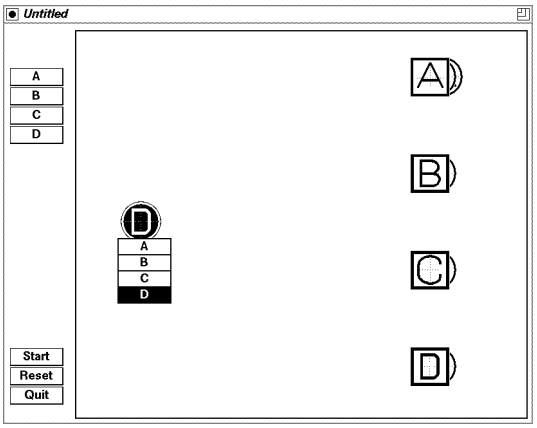

図 1: 実験用プログラムの画面

実験の枠組みとしては、次のようなものにした。

- 「まず操作対象となるアイコンをマウスにより指定し、続いてそのアイコンに対する操作を選択する」という動作を 1 サイクルと考え、その各種の操作選択方式を用いた場合の所要時間を比較する。
- 選択肢の数は人間にとってひと目で認識できる数として、今回は 4 を採用した。より多くの選択肢については後で扱う。
- 比較する操作方式としては、ドラグ & ドロップ、パレット、ポップアップメニュー、キーボード、および「投げる」インタフェースの 5 種類とした。マウスボタンによる選択は用途が限られているとの考えから含めなかった。また、プルダウンメニューについては今回は選択肢が 4 つのため、パレットにくらべたメリットがないと判断し除外した。

図 1 に実験用プログラムの画面の様子を示す (ポップアップメニューを出しているところ)。

ここでメニューのすぐ上にある丸いものが対象アイコンである。選択肢は 4 つなので、各アイコンには A ～ D までの文字が描かれ、被験者はそれと同じ操作 (つまり操作名も A ～ D となっている) を選ぶことになる。従って、操作を現す目的アイコンにも、パレットにも、ポップアップメニューにも A ～ D の文字が記してある。具体的な操作は次の通り。

- 投げる: 対象アイコンの上でマウスボタンを押

表 1: 各方式による操作 1 ラウンドの所用時間

被験者	投げる	ドラグ	パレット	メニュー	キー
A	21.57 (1.05)	34.78 (3.26)	42.45 (4.81)	37.67 (5.53)	15.09 (1.65)
	1.00	1.61	1.97	1.75	0.70
B	15.41 (0.71)	27.45 (2.49)	35.48 (3.49)	27.29 (4.50)	13.97 (0.75)
	1.00	1.78	2.30	1.77	0.91
C	20.41 (1.85)	33.42 (1.98)	35.00 (3.14)	27.68 (3.77)	14.92 (1.85)
	1.00	1.64	1.71	1.36	0.73
D	18.57 (0.77)	33.92 (2.05)	36.60 (1.28)	26.89 (2.83)	15.04 (0.91)
	1.00	1.83	1.97	1.45	0.81

単位は秒数、かっこ内は標準偏差
下段は「投げる」時間を 1 とした時の比率

し、そのまま同名の目的アイコンに向かって動かし、動いた状態でボタンを離す。

- ドラグ & ドロップ: 上と同様だが、途中で離すのではなく、目的アイコンの上まで持って行って重ねた状態で離す。(重なると目的アイコンの色が変わるようになっている。)
- パレット: 対象アイコンをクリックして選択した後、マウスカーソルをパレットの上に持って行って A ～ D のいずれかの上でボタンをクリックする。
- メニュー: 対象アイコンをクリックして選択した後、コントロールキーを押しながらマウスボタンを押すとメニューが表示されるので、その状態でカーソルを動かし A ～ D のいずれかが反転した状態でボタンを離す。
- キー: 対象アイコンをクリックして選択した後、キーボードの A ～ D のキーを押す。

対象アイコンは半径 30 ドット (実験に用いた画面では 1 ドット = 約 0.27mm) の円、目的アイコンは 1 辺 60 ドットの正方形、パレットとメニューの 1 項目は高さ 25 ドット、幅 80 ドットの長方形であった。

なお、「投げる」場合には対象アイコンが目的アイコンに接しなくても、対象アイコンの中心が目的アイコンの中心から 90 ドット以内に入った時には「引力」が働いて引き付けられるようにした。さらに、投げたアイコンが壁 (窓の四辺) にぶつかった場合には停止するのではなく、ぶつかった壁に沿って「ころがる」ようになっている。このようにしたのは、投げた以上「何もない」ところをめざして投げることはないはずなので、多少それた方向に投げてもそばにある目的アイコンを選択したものと見なせるようにするためである (これらの点についても後で議論する)。

実験は 20 個の対象アイコンが同一位置に重なったものを 1 セットとし、5 種類の選択方法を各 9 セット、合計 45 セット実行した。実験に際しては、できる範囲ですみやかに操作することとし、誤りに気が付いた場合に訂正するかどうかは被験者に任せた。

表 1 に各方式毎の 1 セットの所用時間 (最初の対象アイコンの上でボタンを押してから、最後にウィンドウの終了ボタンを押すまでの時間) を被験者別に集計したもの、および「投げる」操作を 1 としたときの比率を示す。これを見ると、どの被験者においてもキーによる選択が最も早いが、それに続いて「投げる」操作が早く、残る 3 つはあまり差はなくどれも遅いという結果になっている。

また、表 2 に誤り数を集計したものを示す。これは実験で採取した事象の系列から時刻、マウス位置、位置報告イベントなどの情報を削除した操作系列を作り、それと予め用意した標準的な操作系列とで異なっている箇所を数えたものである。(複数の連続した差異は 1 個に数え、ただしサイクルをまたがる場合は分けて数えた。) さらに、これらのうちメニューの項目番号や目標アイコン番号のみが違っている場合を選択誤り (下段)、それ以外を操作の逸脱 (上段) として分類した。

なお、全部の操作系列を検討するのは大変だったので、9 ラウンドの実験のうち最初の 2 ラウンド (40 操作) のみについて行い、誤り件数が多くないことから全被験者の誤りを合計して示した。これを見ると、「投げる」操作における選択誤り (「投げ間違い」) の比率は予想通りドラグ & ドロップよりは多いが、メニュー

の選択し損ないよりはだいぶ少ないことがわかる。

表 2: 各方式による操作の逸脱 (上) と選択誤り (下)

投げる	ドラグ	パレット	メニュー	キー
1	5	13	11	5
0.63%	3.13%	8.13%	6.88%	3.13%
5	1	0	22	0
3.13%	0.63%	0%	13.75%	0%

3.3 操作時間の検討

次に、より各操作のより細かい分析を行うため、各実験で対象アイコンを選択し、操作してから次の対象アイコンを選択するまでを 1 サイクルと考え、1 サイクルぶんの操作系列を抽出してタイミングチャートを作成したものを図 2〜6に示す。抽出に際してはエラーの分析を行った第 2 ラウンドのデータを用い、「次の対象アイコン」がない最終サイクル、エラーを含むサイクル、および他のサイクルよりはっきりと長いサイクルを除外して残ったものの平均を取っている。

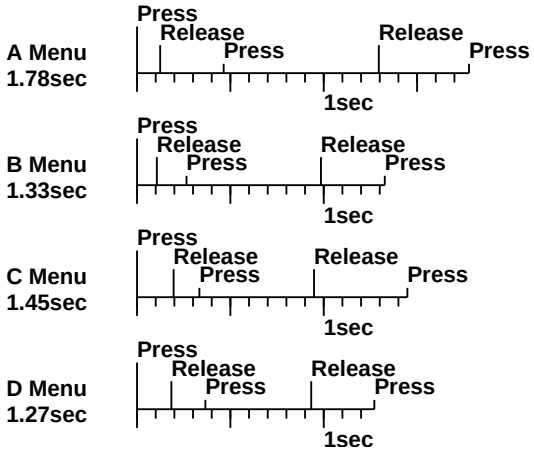

図 2: メニュー操作のタイミング

まずメニュー操作 (図 2) について見ると、最初に対象アイコンを選択するためボタンを押してすぐに離し、次にメニューを出すため再度 (コントロールキーを押しながら) 押している。ここで被験者 A だけ時間が掛かっているが、これは被験者 A は対象アイコンの真上にメニューが出るのを好まず少しマウスカーソルを移動してからメニューを出す操作を行っていたからだと思われる。そこから押したままマウスを移動してメニューを選択するのに比較的時間が掛かるが、終わった後次の対象アイコン選択のため元の位置までカーソルを戻すのはさほど掛かっていない。

次に一連の操作の所用時間を KL モデルにより予測してみる。操作系列は

$$KKP_{10.5,7}KP_{10.5,6}K$$

となる。最初の 2 つの K は、メニューを出すための Control キーとマウスボタンの押し下げに対応している。次の $P_{d,s}$ は距離 d 離れた大きさ s の目標への移動を現す (距離の単位は mm)。KL モデルではマウスボタンを押して離すのを 1 つの K としているが、この実験の場合はボタンを押したまま別の操作をし、改めて離すことからマウスボタンを離す操作も独立した K(3 番目のもの) として考えた。次の P は次の対象アイコンまで戻る操作、最後の K はその対象アイコンを選択するものである。

ここでポインティングに要する時間は 3.1 節に挙げた式により、打鍵時間は均等に 200msec とすると、予測される時間は 200 + 200 + 300 + 200 + 320 = 1220msec であり、実測値とよく合っているといえる (被験者 A については、最初に「よける」ぶんの移動 $P_{16,16}$=358msec を加えるので、これもおおむね合っている)。

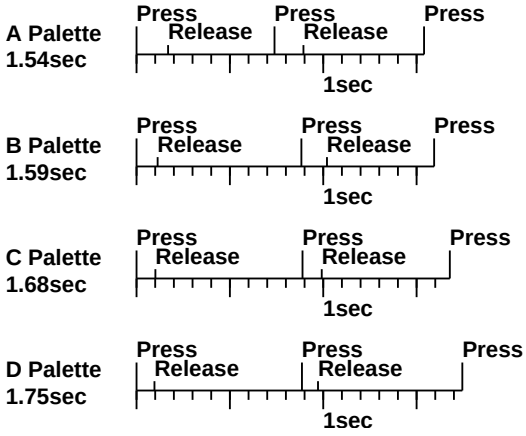

図 3: パレット操作のタイミング

パレット操作 (図 3) の操作系列は、まず対象アイコンの上でマウスボタンを押して離し、続いてパレットの上に移動してそこでマウスボタンを押して離し、その後元の位置に戻って次の対象アイコンの上でマウス

ボタンを押すというものである。これはメニューに比べて単純なため有利であると先に予想したが、実際にはメニューよりやや遅くなっている。これは操作系列としては単純でもアイコンとパレットの間の移動に時間が掛かるためである。KL モデルによる操作系列では

$$P_{78,15}K\,P_{78,16}K$$

であり、予測される時間は 550 + 200 + 540 + 200 = 1490msec となり、これもよく合っているといえる。

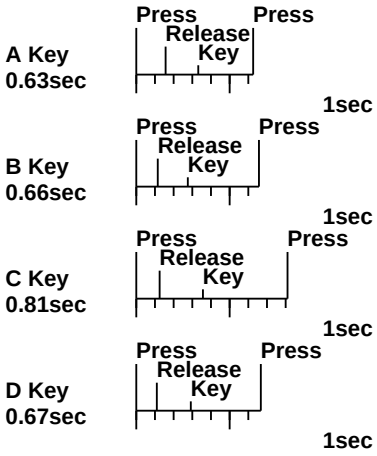

図 4: キー操作のタイミング

キー操作の操作系列(図 4)は対象アイコンの上でマウスボタンを押して離し、続いてキーを押し、再び次の対象アイコンの上でマウスボタンを押すというものである。KL モデルによる操作系列は

$$K\,K$$

だけなので 400msec となり、過少である。

これは思考時間に関係しているものと思われる。すなわち、今回の実験では設定が「A を A に重ねる」ようなものなので思考は不要として一切思考時間を考慮していない。(KL モデルでは思考時間として一律に 1.35 秒を採用しているので、もし含めたとすればすべての見積もりが過大になるのは明らかである。) しかし、実際にはこの思考時間はマウス移動操作とオーバラップして隠されているだけであり、$K\,K$ のような短い系列になるともはや隠れていることができなくて現れて来るからと考えればつじつまが合う。それでも 1.35 秒は過大であり、例えば木村らによる「少数の選択肢からの選択」オペレータ [?] のようなものをあてはめるのがよいかも知れない。

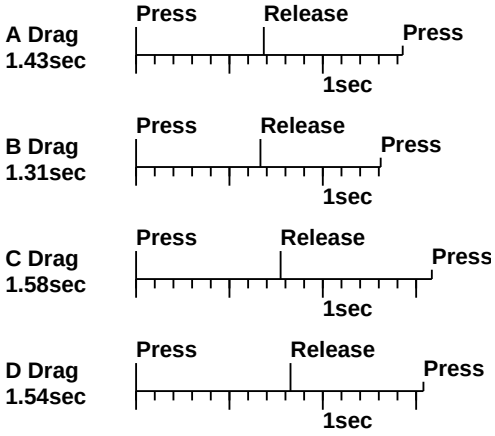

図 5: ドラグ & ドロップ操作のタイミング

ドラグ & ドロップは図 5 のように操作系列は単純で、対象アイコンの上でマウスボタンを押し、そのまま移動して目標アイコンの上で離すだけである。KL モデルによる操作系列は

$$P_{128,16}K\,P_{128,16}K$$

であり、予測時間は 610 + 200 + 610 + 200 = 1,620msec となり、やや大き目になるがまあ合っているといえる。

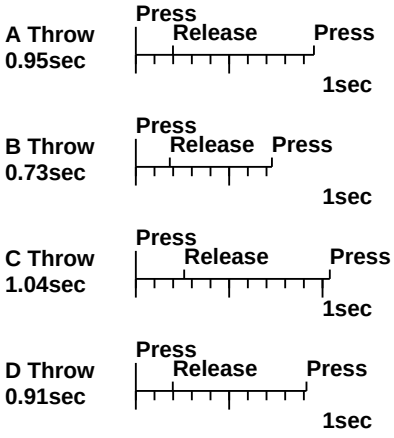

図 6: 「投げる」操作のタイミング

そして、「投げる」操作(図 6)の系列はドラグ & ドロップと同じであるが、ただし途中でボタンを離すため、ボタンを押してから離すまでの時間がずっと短い。

「投げる」操作はKLモデルに含まれないためKLモデルでは予測できないが、これをとりあえず一定の時間を要するTオペレータなるものであると考えるとすると、操作系列は

$$TP_{d,16}K$$

となる。ここでTの時間はボタンが押されてから離されるまでの時間だと考えると、図6からは200～250msecであると読み取れる。この値はキーボードの打鍵時間とさほど変わらない値であり、「投げる」操作の効率性を表しているものと考える。

3.4 議論

本節では前掲の実験結果に基づき、「投げる」インタフェースが全体としてどれくらい有望であるかを検討する。

まず、表1の全体としての所要時間を単純に考えた場合、「投げる」方式はキーによる選択に続いて高速であり、十分有望だと考えられる。とくに今回の実験設定ではキー操作はマウスを全く動かす必要がなく、そのため非常に有利になっている。実際のGUIではある対象アイコンを操作し終わった後次の対象アイコンまで移動することは避けられず、その場合にはキー操作と「投げる」操作はもっと近付くことになる。

さらに、今回の実験設定では避けてあるキーの弱点として、ニモニックの割り当てが容易でないこと、右手の受け持ちキーの場合には手の移動が必要になることが挙げられ、これらを加味するとむしろ「投げる」方が高速になる可能性もある。

他の方式についてはいずれも「投げる」方式より明らかに所要時間が長く不利であったが、このうちメニューとパレットについてはこれ以上の時間短縮は難しいと思われる。(メニューについていえば、今回の実験では次の対象を選ぶための移動距離がメニュー選択のため動いた距離と同じでごく短いため時間的に有利であり、現実にはもっと時間が掛かるはずである。)

もう1つ興味深かったのは、「投げる」時にボタンを離す位置までの移動距離が被験者によって異なり、被験者A、C、Dで60mm程度、被験者Bで25mm程度であったことである。このことと被験者Bの所要時間が他の被験者と短いことは何らかの関連があると予想できるし、また利用者が短い移動距離で投げられるように修練を積むことで所要時間も短縮できる可能性が大きい。

また別の問題として、投げた後のマウスの位置は必ずしもボタンを離した位置ではないといことも考慮すべきである。つまり、「投げる」場合にはマウスを動かした状態でボタンを離すので、投げ終わった時にマウスのある位置はもっと最初の位置から遠く、従ってTの正味の所要時間は上記の値よりもう少し長いと考えるのがよいかも知れない。

このことは、まったく初めての人に「投げる」インタフェースを試してもらうとうまく投げられず、止まってしまってからボタンを離すのでアイコンも止まってしまったり、マウスを戻し初めてからボタンを離すのでアイコンがあらぬ方向に飛んだりすることからもうなずける。

なお、今回の実験ではマウスボタンを押していない状態でのマウス位置は記録しなかったため、実際にどこまでマウスが行ったかは後からは判らなかった。反面、今回の実験では投げた後必ずマウスは元の方向に戻すことになっていたが、実用の場面では投げた後「向こうの方に」次の操作対象のアイコンがある可能性もあり、その場合にはむしろ次の操作の時間が短縮できる可能性すらある。

次に誤り率につては、表2を見る限りメニューの選択間違いよりも投げ損ないの方が少ないことになる。ただしメニューは間違えては困ると思ったらゆっくり慎重に操作することができるが、投げる場合にはかなり熟練しても投げる限りは一定の投げ損ないが避けられないと思われる。これについては基本的にundo機能を用意し、投げ損なった場合にはすぐ元に戻してやり直せるようにするべきであろう。なお、投げるインタフェースはそれ自体がアニメーションなので、メニューやボタンのように選択したものを点滅させるなどの工夫を設けなくても間違いの認識は容易である。また、どうしても間違えたくない場合にはドラグ & ドロップによることもできる。

今回提案した方式は純粋にソフトウェア的な実現を前提としていたが、文献[6]に提案されている慣性マウスのように、マウスのハードウェアを拡張することで「投げる」動作を実現しているものもある。これはマウスの下面にスイッチを設け、マウスを物理的に持ち上げると、持ち上げている間ずっと持ち上げる直前の方向/速度の移動が続いているかのような信号を送出するものである。この方式の利点はソフトウェアの変更が不要で任意のソフトウェアに仕える点であるが、「引力」や後述する「摩擦」「カーブ/シュート」な

どのアイデアは適用できない。

一方、文献 [6] によれば慣性マウスの利点は

- 遠くの対象を指示するときでもマウスを大きく移動しなくて済む
- マウスをマウスパッドの中央付近に置いておける

という点から来ているとのことだが、これらはアイコン投げにおいても同様であるといえる。ただし、今回の枠組みではアイコンは投げられるがマウスカーソルは投げられないので、対象アイコンを選択する時には通常のポインティングになってしまう。この点を改良してカーソルまで投げられるようにすべきかどうかも今後の検討課題である。

3.5 実験の課題と拡張の可能性

今回の実験ではとりあえず他の選択方式との優劣を検討することが目的だったので、「投げる」操作自体の持つ性質については十分調べることができていない。たとえば被験者 B と他の被験者の差に見られた「投げ方のうまさ」の分析や、投げる時間が目標の遠さに依存しないかどうかなど、調べてみたいことは多数ある。

また、今回の実験で選択肢を 4 としたのは、あまり選択肢が多いと選択肢を探すための思考時間が無視できず、KL モデルによる検討が難しくなるからであった。しかし、投げ損ないの率は目標アイコンの数や距離といった「投げ易さ」に影響されることも明らかであり、選択肢が 4 つという比較的投げ易い (と思われる) 場合だけの実験では問題がある。そこで、盤面の大きさは変えず、アイコンの数を倍の 8 にして被験者 C のみで実験を行ってみた。

この実験では X サーバが XFree-86 2.0 に代わっており、実験も「投げる」もののみ連続して行ったので必ずしも表 1 と同等に比べられないが、めやすにはなる。その結果は 1 ラウンド所要時間の平均が 23.82sec、標準偏差 0.67 であり、時間的には「やや遅くなった」程度である。その遅延が思考時間によるかどうかの検討は今後の課題である。また、選択誤り率は 15.6% で、4 個の場合の 3 倍に増えているが、予想された結果であり、この程度であれば実用に耐えると考える。

実用に使う場合であれば、常に目標アイコンが投げる位置から見てうまく横に並んでくれるとは限らない。そのため、手前のアイコンを「よけて」向こうにある目標アイコンに到達させる方法も必要であろう。そのためのアイデアとしては、

- 投げたアイコンが窓の周囲で「はね返る」ようにして、クッションを利用して到達させる。
- 投げる時の軌跡の方向ベクトルの角速度を記憶しておき、それにしたがって投げられたアイコンもカーブやシュートさせる。
- 画面上にも地上と同様の重力を仮定し、投げたアイコンが放物線を描いて飛ぶようにする。
- 飛んで行ったアイコンが目標アイコンに「つかまる」ための操作を明示的に (キーボードなどで) 行う。ただし、これを行うとオーバラップして投げることは難しくなる。
- 「摩擦」を導入して、投げたアイコンがだんだん遅くなるようにし、ある程度以上の速度であれば目標アイコンにつかまらずに通過するようにする。

などが考えられる。今回は実験する余裕がなかったが、今後試してみたいと考えている。

ただし、実用的には「投げにくい」場合の不利さはそれほど問題にならない可能性もある。というのは、先にも述べたようにドラグ & ドロップではアイコンの配置を変更すること自体がカスタマイズであり、従って特によく使う操作に対応する目標アイコンを少数選んで投げやすい位置に置くという状況を想定すれば、あえて投げにくい時に投げる必要はあまりないかも知れないからである。

4 まとめ

GUI において複数の選択肢からの選択を行うのに、メニューと並んで有力な方法であるドラグ & ドロップの操作時間を改善するため、「アイコン投げ」と呼ばれる操作方式を提案した。その操作時間を評価するための実験の結果、「アイコン投げ」方式は条件によってはメニュー、パレット、ドラグ & ドロップより高速であり、誤り率も少なく、実用に耐えるとの感触を得た。

今後の課題としては、「アイコン投げ」の操作時間を説明するようなもっと細密なモデルを組み立て検証すること、および 3.4 節で挙げたような操作性を改善するアイデアを実現し評価することが挙げられる。最終的には「アイコン投げ」を利用した GUI ベースのコマンドインタプリタで、(少なくとも複雑な制御構造やパターンを必要としない場合には) 通常のコマンド方式のインタプリタと同等以上の使い易さを持つものの開発を目指してみたい。

参考文献

[1] Graphical User Interfaces: The Next Generaiton, Special issue, CACM, vol. 36, no. 4, 1993.

[2] 久野　靖, 角田博保, 流れて行かない Unix 環境, 情報処理学会論文誌, vol. 29, no. 9, pp. 854-861, 1988.

[3] 佐藤直樹, 久野　靖, 鈴木友峰, 中村秀男, 二瓶勝敏, 明石　修, 関　啓一, 第 29 回プログラミングシンポジウム報告集, pp. 13-22, 1988.

[4] Stuart K. Card, Thomas P. Moran, Allen Newell, The Psychology of Human-Computer Interaction, Lawrence Erlbaum, 1983.

[5] 木村　泉, 粕川正充, ワープロ利用者の思考時間に関する統計的模型の精密化, 情報処理学会文書処理とヒューマンインタフェース研究会 14-4, 1987.

[6] 野中秀俊, 伊達　淳, 慣性機能を持つポインティング装置の開発, 情報処理学会論文誌, vol. 31, no. 2, pp. 268-274, 1990.

付録: モールス符号による選択

本文で述べたように、マウスボタンや修飾キーによる区分は所要時間の面では有利だが、どの手順がどの操作に対応するか記憶するの難しいという問題を持つ。その理由の 1 つとして、この種の操作手順はキーボードのキーなどと異なり「自然なニモニック」を持たないことが挙げられる。そこで、例えばマウスボタンでモールス信号を打ち込むようなことを考えれば、モールス信号をニモニックとして認識する訓練ができている利用者にとっては自然で使いやすいインタフェースになるかも知れない。これも「初心者に厳しい」インタフェースの 1 つの候補であるといえる。

筆者らのなかでは被験者 A だけがモールス信号の使用経験を持っていたため、今回の実験用プログラムを改造してモールス符号による選択の実験を行ってみた。プログラムでは標準な短点 (トン) の時間を t とおき、マウスボタンの押しから離しまでの時間が $2t$ 未満を短点、それ以上 $5t$ 未満を長点 (ツー)、それを越えた場合は 1 字終わりとしている。t としては被験者 A に繰り返し試させた結果 70msec を採用している。

実験データは表 1 のデータと同じものを用い、9 ラウンド連続して実施した。その結果、1 ラウンドの平均所要時間は 42.3 秒、標準偏差は 11.61 秒となった。ただし、そのうちエラーによるやり直しを含まないものだけを取ると 35 秒程度で、表 1 のドラグ & ドロップ、パレット、メニューなどと同程度になる。

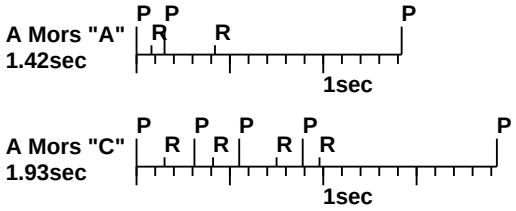

図 7: モールス信号による選択 (上:A、下:C)

典型的な 1 サイクルのタイミングチャートを図?? に示す。上が文字 A(トンツー)、下が文字 C(ツートツート) に対応している。(Press、Release をそれぞれ P、R と略してある。) これを見ると、モールス符号の系列が長いとそれに呼応して時間が掛かること、および「1 字おわり」の空白を置かないと選択が完了しないため最後の Release から次の対象アイコンの Press までが非常に長いことが判る。さらに、長点と短点の打ち間違いがあった場合、そのことが判るのは $5t$ 待った後で、その後やり直しとなるのでかなり時間が掛かる。

このような問題にも関わらず、被験者 A はキーボードのみの使う選択よりも快適であるとの感想を述べているが、これは右手だけで済むことも 1 つの理由だとのことであった。全体としてモールス符号による選択は時間的には不利であるが、打ち間違いを避けるようにもっと賢い t の自動調整を行い、エラーを少なくすればメニュー等と同程度以上の効率にはなるのではないかとの感想を持った。