GUI Behaviors to Minimize Pointing-based Interaction Interferences

Alice Loizeau, Sylvain Malacria, Mathieu Nancel

GUI Behaviors to Minimize Pointing-based Interaction Interferences

ALICE LOIZEAU, Univ. Lille, Inria Lille - Nord Europe, CRIStAL - UMR 9189, France
SYLVAIN MALACRIA, Univ. Lille, Inria Lille - Nord Europe, CRIStAL - UMR 9189, France
MATHIEU NANCEL, Univ. Lille, Inria Lille - Nord Europe, CRIStAL - UMR 9189, France

Pointing-based interaction interferences are situations wherein GUI elements appear, disappear, or change shortly before being selected, and too late for the user to inhibit their movement. Their cause lays in the design of most GUIs, for which any user event on an interactive element unquestionably reflects the user's intention—even one millisecond after that element has changed. Previous work indicate that interferences can cause frustration and sometimes severe consequences. This paper investigates new default behaviors for GUI elements that aim to prevent the occurrences of interferences or to mitigate their consequences. We present a design space of the advantages and technical requirements of these behaviors, and demonstrate in a controlled study how simple rules can reduce the occurrences of so-called "*Pop-up*-style" interferences, and user frustration. We then discuss their application to various forms of interaction interferences. We conclude by addressing the feasibility and trade-offs of implementing these behaviors in existing systems.

CCS Concepts: • **Human-centered computing** → **Human computer interaction (HCI)**; **Interaction design**; *Empirical studies in interaction design.*

Additional Key Words and Phrases: interferences, pop-ups

1 INTRODUCTION

Typical Graphical User Interfaces (GUIs) behave as if every user input was fully intentional and informed, *i.e.* under the implicit assumptions that (1) input events faithfully reflect the user's intention at the time they were sensed, and (2) that this intention stems from a perfect and instantaneous interpretation of the system's state. These assumptions, however, are not systematically true, and situations that fail to meet them can lead to serious consequences and frustrating experiences.

Interaction interferences [27] are a typical example of such situations. They are changes in the user interface occurring immediately before the user carries out a meaningful atomic input action, but too late for them to inhibit that action—even if they perceived the change—which as a consequence is interpreted differently than the user intended. According to Schmid et al.'s online survey [27], users consider interaction interferences a common issue that can generate from mild annoyance to downright frustration, and occasionally cause serious consequences.

In that same survey, Schmid et al. found that a majority of the interference examples provided by their participants occurred when selecting a target, either with a cursor or with the finger—the

bulk of the rest having to do with text input. In particular, they identified examples wherein the intended target of an action either:

- moved (*e.g.* trying to click a link when the webpage unexpectedly updates),
- changed (*e.g.* the word suggestion above a gesture keyboard changes just before it is tapped),
- became occluded (*e.g.* a notification appeared above the target right before the click),
- or disappeared (*e.g.* a notification disappears right before the user clicks to close it).

Consequences can range from a harmless click, to triggering a critical operation that cannot be cancelled nor undone, *e.g.* sending an unfinished email (not all email clients have an "undo" feature), triggering a half-hour-long update right before giving a keynote, mistakenly clicking "No" (or "Yes") when asked to save before closing an app, answering an identified scam call, etc.

In this paper, we investigate how GUI toolkits could be refined to help prevent or mitigate pointing-based interaction interferences by adapting the display and behavior of their widgets, depending on the extent of information and control that they can access: does the toolkit "know" the cursor location, or the location of upcoming changes in the interface? Can it alter the location, timing, or visual characteristics of that change? Can the next actions of the user be accurately predicted? etc. We present a design space of possible GUI widget behaviors to mitigate the cost of interaction interferences. We also report a controlled experiment in which we compared a selection of characteristic behaviors from this design space in terms of interference avoidance and user preference, using appearing (or "pop-up") windows as a representative type of interferences. Its results suggest that simple systematic behaviors, such as making new windows unresponsive for a few milliseconds or appear away from the cursor, can already prevent most interferences—or their worst consequences—without hindering normal use. Finally, we present design guidelines resulting from this experiment's findings, and discuss how they can generalize to a broader set of pointing-based interferences and to interferences in general, as well as possible next steps in the study and solving of interaction interferences.

2 BACKGROUND: PERCEPTION AND IMPACT OF INTERACTION INTERFERENCES

2.1 Characterization of Interaction Interferences

Schmid et al. assessed the robustness of their initial description of interferences in an online survey [27] in which they enquired whether participants understood the described phenomenon, at which frequency (if ever) it happens to them, and about the frustration and consequences of its occurrences. Participants were also asked to provide their own examples of interaction interferences. Schmid et al. found both generic and example-based interferences to be somewhat uniformly spread in frequency, with more than 66 % of participants reporting it more than once per month and 17 % up to more than once per day, but consistently annoying (87 % of answers between "Moderately" and "Extremely"). Interestingly, the perception of annoyance seemed to be independent of the device or task, suggesting that interaction interferences are a general problem regardless of the context. While the consequences of interaction interferences were judged as minor in general, some were reported to be severe.

The majority of reported exemples of interferences (57 %) occurred when selecting a target, be it via indirect pointing or on tactile surfaces. For this reason we focused our later experimentation on selection-related interferences.

2.2 Causes of Interaction Interferences

On the "user side", interaction interferences stem from neurological limitations of the visuo-motor system that generate narrow time windows within which a user's planned motor action can no longer be inhibited, even when the user perceives a change in the system's state within that window. Movement inhibition in general has been studied extensively in the fields of psychology

and neuroscience (see *e.g.* [20] for a comprehensive state of the art), with the model of the "horse race" (Figure 1) currently prevalent in this research: "*a 'horse race' between two sets of processes, one that generates a response for the primary task and one that responds to the stop signal: If the primary-task process finishes before the stop-signal process, the response is executed; if the stop-signal process finishes before the primary-task process, the response is inhibited*" [15].

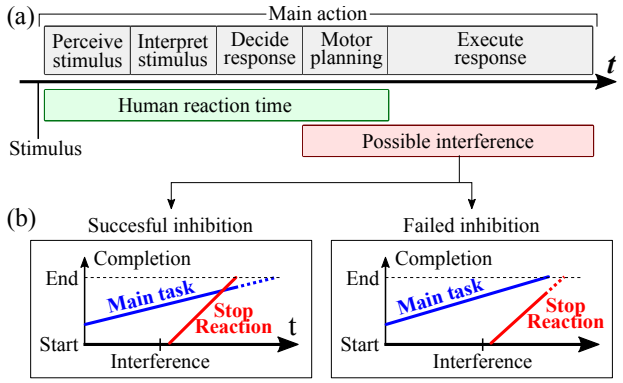


Fig. 1. (a) Simplified diagram of human reaction to a stimulus. Interferences can start before movement onset. (b) The "horse race" model of movement inhibition. The user successfully suppresses his action if the stop-process ends before the main action process.
Reproduced from [27] with the authors' permission.

This phenomenon is studied using variations of the so-called countermanding protocol (see *e.g.* [30] for a detailed and substantiated description) in which participants are asked to perform a simple motor action as soon as a "go-signal" is emitted. In a small proportion of trials, a "stop-signal" is emitted shortly after the go-signal, in which case participants are instructed to refrain from starting the motor action.

Schmid et al. applied a modified version of this countermanding protocol to test in a reciprocal 2D pointing task the time below which a click cannot be inhibited. They found that participants reached a 50% click-inhibition success rate when the stop-signal occurred around 350 ms before the expected time of the click. Interestingly, they found that this "refrain-ability" was not affected by target distance, size, or pointing direction, indicating a process distinct from the reaching motion itself or its planning. Our experiment protocol builds on theirs, but with a simplified stop-signal timing mechanism since we are not attempting to precisely assess the duration between the stop-signal and the expected time of the click itself, or SSCD (Stop-Signal-to-Click Delay). This can be somewhat reminiscent of the protocol used by Faure et al. [8] who, in order to study the effects of target-appearance delay on pointing performance, designed a task in which the participant always *expects* the target to appear, and knows *where* it will appear. Our protocol is substantially different: in our case it is very important that the user be surprised by the interfering window, so we took a number of steps to prevent that expectation (see Section 4 for details).

In the end, the human capacity to inhibit a movement is physiologically limited in time and that limitation cannot be significantly removed with training or experience. This limitation has been known since at least the early 80's, yet GUIs are still designed with a systematic, simplistic handling of user inputs that ignores it. The vast majority of GUI widgets and window managers implicitly consider that as soon as a widget is displayed and active on screen , then the user events it receives are intentional and faithful indications of the user's intention.

2.3 The Singularity of Interferences as Errors

In this section, we refer to an "error" in the general sense of a user action whose effect differs from the user's expectations, which may or may not stem from a user mistake. An interaction interference is an error in this respect since an input (typically from the mouse or keyboard), carrying an intention, does not result in the effect intended at the time it was planned.

Errors happening on a cognitive level (involving decision-making and long-term planning) are well documented in human-machine interaction. GEMS [24], SRK [23], Baber and Stanton's rewritable routines [3], Norman's model of human information processing [22], COCOM [13] and Wickens and Holland's human information processing model [31] all characterize or analyze the errors that can occur from and within this cognitive level. These frameworks show a shift in the conception of error, from blaming human beings and their unreliability, to integrating error as coming from—and being part of—the system's operation. For all of them, nevertheless, input errors involve a wrongly performed motor action or a wrong interpretation of the system's state by the user. Interaction interferences, while arguably a type of error, cannot be modeled by these frameworks: they are entirely unavoidable by the user in a normal course of action; occurring solely at a physiological level, they do not reflect the quality of their decisions, interpretations, or motor accuracy.

When discussing interferences with our fellow researchers, we frequently see them likened to "slip" errors [28]. In HCI, a slip usually occurs in expert use of a tool and consists in a rushed, inattentive motor action caused by misapplied habit or practice. A slip does not only occur at the motor level, but also in the cognitive shortcut established by the repetition of a motor action. This type of error is linked to the cognitive process in action since it can be avoided by procedural and sensory cues [4]. Interaction interferences, on the other hand, occur solely at the perceptuo-motor level.

As a last note, while most of our interference examples and the ones listed in [27] are of "honest mistakes" on the system side, *e.g.* displaying a timed notification or updating a list in real time, recurrent phenomena such as Internet ad pop-ups have primed us to associate pop-up windows with potentially malevolent intent. Pop-ups are often cited as a dark or deceptive UI pattern, *i.e.* "instances where design choices subvert, impair, or distort the ability of a user to make autonomous and informed choices in relation to digital systems regardless of the designer's intent" [12]. Interaction interferences are not in themselves a dark pattern, but one could imagine that some dark patterns could aim at triggering them, *e.g.* to click on an advertisement or to accept downloading a harmful file. However, similar to error taxonomies, we could not find interferences precisely described in the literature on dark patterns. "Interface interferences" in Gray et al.'s classification [11] refers UIs biased towards specific user options, such as pre-selection or false hierarchy. The "Forced Action" category would fit better, but generally refers to trick wording or deceptive bundled consent. Pop-up windows themselves are often mentioned as dark patterns [7, 9, 16, 19] but for their very presence obstructing the interface, not in how they are triggered. Regardless, interaction interferences pose a risk with malevolent design, and systematizing avoidance behaviors in GUIs would prevent their potential use in the future.

2.4 Preventing and Correcting Interferences

The only previous work directly related to interaction interferences [27] focuses on the *problem*: it describes the overall phenomenon in detail, assesses its understandability and some of its qualitative aspects with a survey, and quantifies its timing for point-and-click tasks. The present paper is a first exploration of the space of *solutions* that can be used to reduce their occurrences or consequences.

Interaction interferences are virtually impossible to avoid by the user alone once the motor action is planned and the inhibition threshold crossed. One could formulate naive strategies of being extra careful right before clicking a target or pressing any key, however (1) that would require constant additional attention, for most non-user-initiated interface changes cannot be anticipated; and (2) in a study wherein 15% of trials could feature an interference, *i.e.* much higher than in real life and therefore less likely to be a surprise, participants still failed to inhibit their movement 90% of the time when the change occurred less than 100 ms before the click [27]; Schmid et al. also report that their participants' improvised "tricks" to increase their inhibition success rate, including both slowing down and accelerating before the click, failed to produce any observable result on their inhibition performance.

On a system level, we found no existing tool or technique to prevent, avoid or correct interaction interferences specifically, however that is not surprising considering that their first characterization is recent. Existing interaction mechanisms can be put to use, such as confirmation steps (*e.g.* "Are you sure you want to perform that action?") for consequential actions, or Undo-ing editing [21] or direct manipulation [1, 2] actions. They do not however provide a universal solution to the problem, by a large margin: confirmation steps need to be explicitly implemented for any given action, and the confirmation itself is an action that can be subject to interferences; regarding Undo, command histories do not record every user action, including some among the most consequential such as closing an app or sending an email.

3 SOLUTIONS TO INTERFERENCES

Interaction interferences occur at temporal scales of up to a few hundred of milliseconds [27], too short to be avoided by users reacting to the change. Solving or mitigating them therefore requires adjustments on the GUI side, by taking human capabilities and reaction time into account in the design of interactive systems.

In what follows we introduce a design space of GUI *behaviors* aimed at minimizing the chances or the consequences of interaction interferences, and primarily targeted at interaction designers and software engineers. This work is a first attempt at solving interferences and, as the reader will observe, that often implies interceding or even disrupting the usual flow of input and output events. Some of these adjustments are not trivial to implement in regular GUI frameworks, which were not designed for it, and thus we focus our design space on technical feasibility: which sensing, interpreting, and read/write access is required, at which layer of the architecture. More precisely, our design space includes dimensions on the *interpretative* and *technical* requirements necessary to enable these behaviors. From this space we implemented a set of behaviors specifically designed to prevent or mitigate *pointing-based* interaction interferences, that we evaluated in a controlled experiment.

3.1 A Design Space of Solutions

We define the design space of behaviors along three dimensions:

(1) what it does, *i.e.* the way the behavior would solve or mitigate interferences, which we split into overall **categories** and their respective **types**
(2) its **interpretative requirements**, *i.e.* how "smart" it needs to be in assessing a situation for interferences (behind or ahead of time)
(3) its **technical requirements**, *i.e.* the minimal features that a GUI toolkit needs to expose to allow this behavior

3.1.1 Types of behavior. We distinguish categories of interference-solving behaviors based on when these behaviors occur relative to two main events: the interface change, and the moment the

system responds to the user event (see Figure 2). This results in three categories: ***Prevent*** before the interface change, ***Avoid*** between the interface change and the sensed user event, and ***Correct*** after the system response.

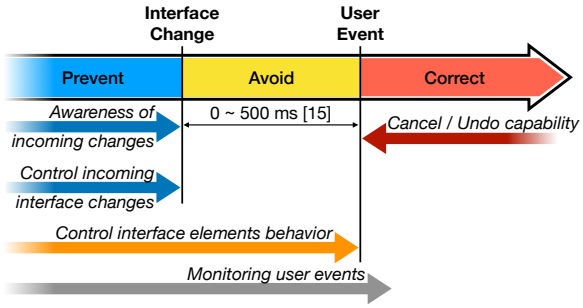


Fig. 2. Time windows for the three main categories of behavior (***Prevent***, ***Avoid***, ***Correct***) and their corresponding technical requirements (in italics).

Ideally, systems ***Prevent*** interferences entirely without delaying or withholding information from the user, by making the change occur in a way that cannot interfere with their incoming actions and with little to no additional latency or visual disturbance. In particular, such behaviors can be implemented by delaying the change until the user's ongoing or incoming action is over (a type of behavior we henceforth dub "Postpone"), or, when the interference is location-dependent, by displaying the change away from the user action ("Displace").

When the problematic GUI change cannot be prevented, it should be possible to ***Avoid*** the resulting interference or its consequences using GUI behaviors that disregard the change, or that do not automatically handle the subsequent events as if they were intentional. Such behaviors can be implemented by Blocking (*i.e.* ignoring) consequential events until enough time has passed, or by Mapping these events to the GUI state prior to the change. For instance, when an application starts automatically it generally grabs the keyboard focus, even when the user was typing in a text field in another app; a Mapping-type solution would be to still send the keyboard events to the text field for a short period of time. In cases where the interference is caused by an element appearing on top of another, this Mapping could be materialized as letting events go Through the foreground elements.

Note that, since interference-***Avoid***ing behaviors occur after the interface change, these strategies can remain problematic to users who perceived the change and might briefly worry about the consequences of their action, before they can observe that these consequences were avoided. ***Prevent***-type behaviors do not have this issue, since they affect the interface change itself.

When nothing can be done to prevent the interfering GUI change or to avoid its wrongful interpretation, post hoc ***Correct***ion should be made available, *e.g.* , by systematically enabling ongoing actions to be cancelled and finished actions to be undone, and by making non-undoable actions to require confirmation before they start. Such corrections could be either Automatic when the system can confidently estimate that a recent action caused an interference, or Prompted by directly asking the user if they wish to correct its consequences, with the respective risks of false positives or annoying the user.

Finally, *a minima*, the system should be able to Inform the user that an interference is suspected to have happened. Note that even this last-resort solution requires dependable means to detect interferences automatically.

The categories and types of behaviors are summarized in Table 1.

Behavior Category	Behavior Type	*Predict*	*Naive*	*Detect*
Prevent: the change occurs in a way that cannot cause an interference.	DISPLACE*: occurs away from incoming user events.		X	
	POSTPONE: occurs after at-risk user events.		X	
Avoid: the change occurs normally but the GUI behavior averts the wrongful interpretation of the action.	BLOCK: ignores user events for a time.		X	
	THROUGH*: events skip the foreground element for a time.		X	
	MAPPING: handles events as if they preceded the change.		↑	
Correct: the system offers means to cancel or rectify the consequences of a (likely) interference.	AUTOMATIC: applies corrections automatically.			
	PROMPT: asks the user whether to apply the corrections.			
	INFORM the user of suspected interferences.			

Table 1. Possible combinations (in blue) of behavior types (rows) ordered by categories (1st column), and interpretative requirements (last three columns) in our design space. Asterisks (*) indicate behavior types that are location-dependent. Cells marked with 'X' indicate behaviors evaluated in the following controlled study. In our study of pop-up interferences below, the condition implementing THROUGH (*Naive*) will also serve as a demonstrator for MAPPING (*Naive*) (↑).

3.1.2 Interpretative requirements. Different types of behaviors also require different degrees of situational interpretation from the system, typically to assess the incoming "threat" of interference, or to confirm that one has just occurred. We distinguish in particular the capacity to *Predict* or *Detect* interaction interferences on the one hand, and *Naive* behaviors that are always active on the other hand.

A peculiarity of interface interferences is that we can only be sure that a GUI change is interfering *after* a user action was interfered with. Altering a specific GUI change in order to prevent an interference therefore implies the ability to *Predict* that it will interfere with incoming events. This is conceivable in situations for which accurate user models exist, *e.g.* slowing down at the end of a unidirectional cursor movement before a click or training a gesture recognizer to detect the *likely* start of a command. In other contexts, such as keyboard shortcuts or touchscreen taps, there might be too little information to go around without additional sensors.

Similarly, *Detect*ing that an interference has just occurred, *i.e.* that a couple {GUI-change ; user-event} constitutes an interference, can be nontrivial without explicit user feedback. Typical examples involve finger, hand, or device gestures for which it can be difficult to pinpoint an exact starting timestamp. For pointing tasks, there exists a grey area after an interface change wherein click or tap events might or might not correspond to an interference [27]. The behavior types in the ***Correct*** category typically all require *Detect* capability, since otherwise the user would be prompted for confirmation or for Undo *after every single action* they take.

In contrast, GUI elements can be assigned "*Naive*," always-on behaviors that may by themselves be enough to prevent some interferences. All three approaches pose risks of false positives and visual disturbance, to a degree that remains to be explored.

In the remainder of this work we focus on solutions that do not require conjecture (as opposed to *Predict* or *Detect*), first because designing and assessing prediction/detection methods is a distinct endeavor that is beyond the scope of this paper, and second because we postulate that many contexts and toolkits today expose too little information to confidently detect interaction interferences, let alone predict them. We therefore focus this paper on "*Naive*" solutions that can be always active, as a first exploration of the design space of interference-solving behaviors.

3.1.3 Technical requirements. The behaviors outlined above can easily be ranked in terms of desirability (for what is better than simply preventing any problematic situation?) but there exists a trade-off with the amount of information and control they each require:

(1) *Awareness of incoming interface changes*, *i.e.* the ability to be notified when an interface change is about to occur, is required for any behavior of the ***Prevent*** category. For instance: a pre-planned calendar reminder is scheduled to appear in a second; a new word suggestion was just computed and is about to be displayed above a gesture keyboard; new emails have been received and will be added to a visual list. Unless there exists native callbacks or hooks for these events, which is at best very rare, this implies access to application- or webpage-specific information that is often only available to its developers.

(2) *Control over incoming interface changes*, *i.e.* where (Displace) or when (Postpone) a scheduled interface change should occur first. This is distinct from *e.g.* translating or hiding an *existing* GUI element, which is usually feasible at the OS or toolkit level. This type of capability is rarely controllable from outside the source code of a program and generally requires access to application- or webpage-specific logic and control.

(3) *Monitoring of user events*, typically recent cursor displacements, the location and timing of mouse clicks, or key presses, is required for any behavior that attempts to *Predict* or *Detect* an interference. Whether this monitoring is system-wide or application-specific (*e.g.* JavaScript `mousemove` events fired from within the web browser window) defines the scope at which the corresponding interference-solving behavior can be used. For instance, the `mousemove` events of a cursor moving towards a web browser window will not be received by the corresponding webpage's scripts until it reaches that window.

(4) *History of recent interface changes* and states, *e.g.* the ability to know when an interface actually changed, what the previous word suggestion was, or the state of the email list prior to its recent update, is necessary to *Detect* recent interferences and to temporally remap input events to prior states of the interface. In most use-cases this history does not need to be complete or long-lasting, since interferences only involve recent interface changes; nevertheless, this is perhaps the least likely capability to have available today since command histories generally don't include interface updates.

(5) *Control over interface element behaviors*, *i.e.* the ability to modify their aspect, their opacity, or directly the handles associated to events occurring upon them or their sub-components, is the minimal requirement for all *Naive* behaviors.

(6) Finally, overall *Control over the resulting commands* that were triggered as a consequence of user events, typically the ability to pause, cancel, or undo a recent command, is a requirement for all behaviors of the ***Correct*** category.

Table 2 indicates the (minimum) technical requirements to implement the eight behavior types defined earlier in this section.

3.2 Naive Behaviors Design for Pointing-triggered GUIs

As stated above, we now focus on the *Naive* area of our design space, *i.e.* on solutions that do not require conjecture about the likelihood of incoming or recent interferences. Designing and assessing prediction or detection methods is a distinct endeavor that is beyond the exploratory goals of this paper, and we expect that *Naive* behaviors can already offer significant improvement over many default GUI behaviors.

A majority of the participant-submitted interaction interference examples gathered by Schmid et al. involved pointing target acquisition as the triggering user event, the rest consisting of typing-related events and device gestures [27]. Even within pointing-based interferences, Schmid et al. identified three categories: *Updates* (an existing GUI element changes right before the click or tap),

Awareness of incoming changes	Controlling interface changes	Monitoring user events	History of interface changes	Controlling element behavior	Controlling resulting commands	*Featured in experiment*	Behavior type
■	■	■					Displace (*Predict*)
■	■	■					Postpone (*Predict*)
	■	■				X	Displace (*Naive*)
	■	■				X	Postpone (*Naive*)
		■	■		■		Prompt (*Detect*)
		■	■		■		Automatic (*Detect*)
		■	■	■			Block (*Detect*)
		■	■	■			Through (*Detect*)
		■	■	■			Mapping (*Detect*)
		■	■	■			Mapping (*Naive*)
		■	■				Inform
				■		X	Block (*Naive*)
				■		X	Through (*Naive*)

Table 2. Types of interference-solving behaviors grouped by their technical requirements (green cells). Cells marked with 'X' indicate behaviors evaluated in the following controlled study.

Focus switch (the user events are suddenly sent to the wrong GUI element), and *Pop-ups* (a new GUI element appears above the targeted one). We aim, ultimately, to design and evaluate behaviors for all possible types of interferences, and this paper presents our first step in this experimental endeavour by focusing on the *Pop-up*[1] category as a representative subset of pointing-based interferences. Based on our design space, we designed four of such naive behaviors to address pop-up-style interferences, that implement the four types of behaviors marked with a "X" in Table 1.

Away$_{\text{Displace}}$ (Fig. 3) is an instance of Displace (*Naive*) that prevents *Pop-up*-style interferences by systematically displaying the appearing GUI element away from the path of the cursor, yet close enough that it can be accessed with minimal traveling.

To do so, we constantly retain the recent 100 ms of cursor coordinates (in our case the last 10 cursor events at a frame rate of 1000 Hz, assumed constant for simplicity) and we display the new GUI element so that (1) its center is on the line perpendicular to the latest cursor displacement and (2) the distance between its closest edge and the last cursor location is the same as between the last and 10^{th} last cursor location (see Fig. 4). As a result, the faster the cursor, the further away the new GUI element from its path to reduce the risk of accidentally reaching it. The appearing GUI element's position was corrected when necessary to remain within the screen. This systematic approach is designed to be simple with minimal assumptions about the user's behavior or the positions of targets of interest, and to solve the common case of interferences occurring while or right after reaching a target. It is currently target-agnostic for generalizability, but the display coordinates could easily be adapted to avoid known target areas altogether.

[1]We wish to clarify that the scope of the term *Pop-up* used by Schmid et al. is not limited to "traditional" pop-up windows *per se*. It designates any interaction interference caused by a GUI element appearing above the one currently targeted by the user, *e.g.* when a notification appears in the screen's corner or when a starting app appears in the foreground.

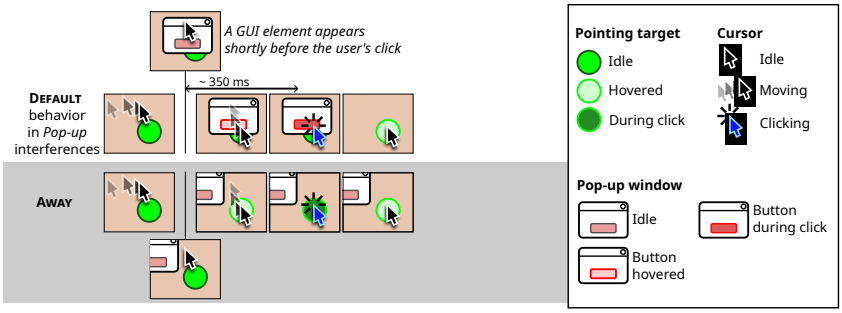


Fig. 3. The **AWAY**$_{\text{DISPLACE}}$ BEHAVIOR makes the window appear at the same time as initially planned, but some distance away from the cursor's path. Clicking as planned selects the target.

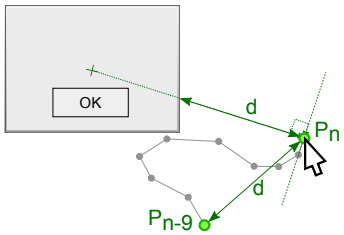


Fig. 4. **AWAY** displays the new GUI element so that its center is on the line perpendicular to the latest cursor displacement, and the distance between its closest edge and the last cursor location P_n is the same as between the last and 10$^{\text{th}}$ last cursor location P_{n-9}.

To simplify, we assume a scenario wherein the system does not know in advance where and when the new GUI element was initially scheduled to appear (*i.e.* no *Awareness of incoming changes*). One notable downside of **AWAY**$_{\text{DISPLACE}}$ is therefore that it overrides that planned position regardless of the cursor's location. Assessing whether that brings performance advantages (the new GUI element always appears near the cursor, yet out of its path) or usability issues (the new GUI element appears in unrelated locations) is left for future work.

DELAY$_{\text{POSTPONE}}$ (Fig. 5) is an instance of POSTPONE (*Naive*) that avoids pointing-based interferences by systematically waiting for a pause in the stream of cursor movements. In our implementation, the behavior waits for a pause of at least 200 ms in `mousemove` events to display the new GUI element. This delay was calibrated in pilot studies and is on the safe side of the reaction times measured in [27]. Note that this mechanism does not guarantee systematic interference avoidance: a pause in cursor movement could still be followed by a late click, which could itself trigger an interference.

Another possible risk with this behavior is that the appearance of a new GUI element could be delayed for an extended period of time during intense cursor activity. To circumvent this possibility, we implemented the additional rule that if a click occurs while a pending GUI element is awaiting a pause in cursor events, then that element is immediately displayed, based on the hypothesis that an interference is less likely to occur immediately after a click[2].

[2]This creates the risk of missing a double-click, but double-clicks were not part of the input vocabulary of our experiment. This rule could be lifted in double-click intensive interfaces, or a delay could be added to the rule corresponding to the system's double-click threshold.

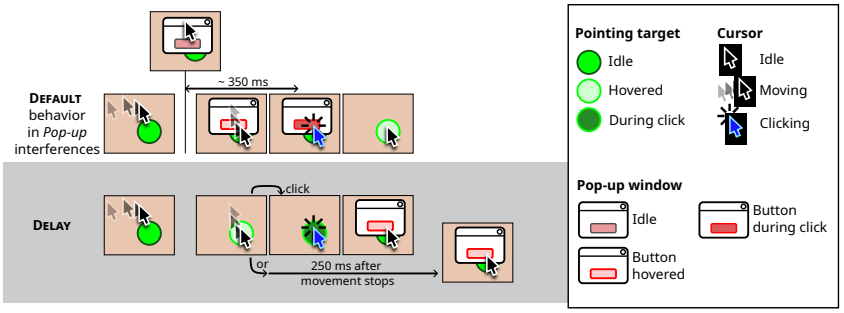


Fig. 5. The **DELAY** BEHAVIOR postpones the appearance of the window until 250 ms have passed, the cursor stops moving, or until a click. Clicking as planned selects the target and immediately triggers the appearance of the window. Moving the cursor around postpones that appearance even further.

HOLEOVER$_{\text{THROUGH}}$ (Fig. 6) is an instance of THROUGH (*Naive*) that lets click events pass through new GUI elements for a fixed duration of one second after they appear, calibrated in pilot testing. As feed-forward, a 'hole' in the new GUI element follows the cursor for the duration of the behavior. Arguably a feed-forward cannot help in situations of true interaction interference in which, by definition, the user cannot take visual cues into account before acting: we added it for situations in which the user knows in advance that a GUI element will appear (*e.g.* out of habit) and needs to know when the **HOLEOVER**$_{\text{THROUGH}}$ behavior times out.

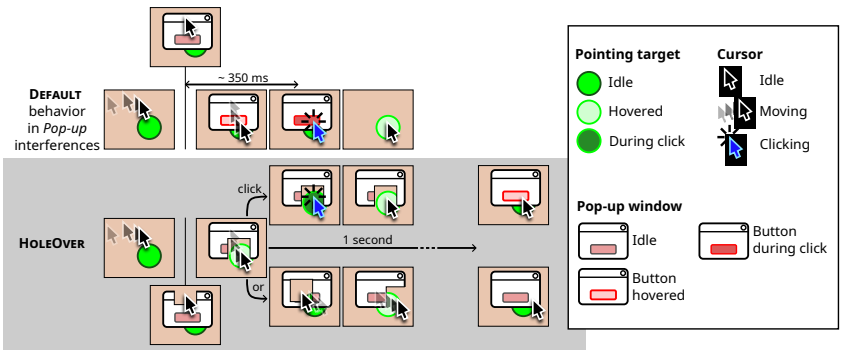


Fig. 6. With the **HOLEOVER** BEHAVIOR the window appears at the planned time and place, but a hole in the pop-up follows the cursor and sends input events through for 1 second. Within that second, clicking as planned selects the target. Clicking after more than 1 second affects the window normally.

FREEZE$_{\text{BLOCK}}$ (Fig. 7) is an instance of BLOCK (*Naive*) that renders new GUI elements unresponsive for a fixed duration after they appear, ensuring that a pre-planned click will not trigger their contents. However, contrary to **HOLEOVER**$_{\text{THROUGH}}$, the initial target does not receive the event. This arguably "safe" approach trades a potentially consequential interference (*e.g.* closing a notification without having read it, or unintentionally accepting an operation) for an inconsequential one (the user's click does nothing). Through pilot experiments we chose a freeze duration of one second to prevent *non*-interfering GUI elements to remain unresponsive for too long after they appear, which could impede the user's workflow. Visually, we gave the frozen GUI element the common visual characteristics of an inactive window, *i.e.* grayed-out background, text and buttons. Finally, to avoid

Default	Popups will now appear without any special behaviour. We will call this behaviour "Default".
Away	Popups will now appear in a different manner called "Away". They will spawn away from the current cursor position. You can try moving your cursor while waiting for the popup to spawn.
Delay	Popups will now appear in a different manner called "Delay". They will spawn either when the cursor is not moving or just after a click. You can try moving your cursor and immobilize when you want the popup to appear. You can also move your cursor around and click.
Freeze	Popups will now appear in a different manner called "Freeze". For a little time after it has spawned, the popup freezes and blocks the clicks. If it is clicked, the freeze effect ends. You can try clicking anywhere on the popup when it has just appeared or wait for the freezing effect to go away.
HoleOver	Popups will now appear in a different manner called "Hole". For a little time after it has spawned, the popup lets the clicks pass through a hole around the cursor. You will see the hole as you move your cursor over the popup. You can try clicking through the popup when it has just appeared to get the target. You can also wait for this effect to end by hovering over the popup.

Table 3. Interference-avoiding behaviors as they are implemented and explained to the participants in the experiment. As opposed to the rest of this paper, the behavior names presented to the participants did not include the category they belong to, *e.g.* **Away** instead of $\textbf{Away}_{\text{Displace}}$.

frozen GUI elements becoming a frustrating delay *after* they have prevented an interference, an element becomes responsive again after the first click event it receives regardless of the one-second threshold. This relies on the hypothesis that an interference is less likely to occur immediately after a click, already used in the design of $\textbf{Delay}_{\text{Postpone}}$.

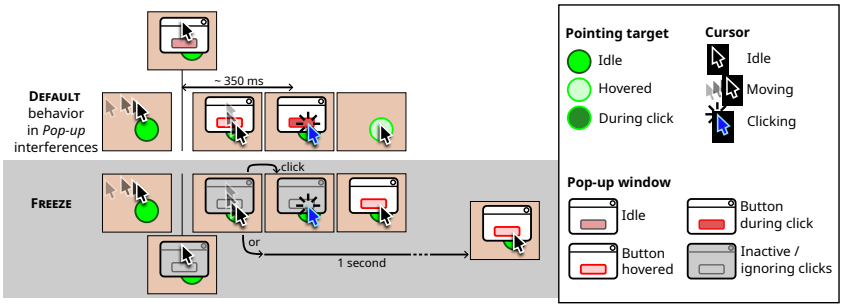


Fig. 7. The **Freeze** Behavior makes the appearing window unresponsive for 1 second or until a click. Clicking as planned deactivates the "freeze", after which any action on the window is treated normally.

A number of these behaviors could be applied as-is to other types of interaction interferences, pointing-based or not. We discuss this further at the end of the paper.

4 CONTROLLED STUDY

We conducted a controlled experiment to evaluate the GUI Behaviors defined above in terms of interference avoidance, user feedback, and overall user experience. Our main objectives were to assess whether, and how, the different *Naive* behaviors defined above can successfully prevent

Pop-up-style interaction interferences or mitigate their effects without deteriorating user experience, and to better understand the trade-offs at hand when designing systems to address them. We later discuss how these results could generalize to other forms of interaction interferences, and what remaining issues could be solved with non-*Naive* GUI behaviors.

4.1 Participants

20 participants took part in the study (5 F, 15 M, ages 21 to 48, mean 31.5). All participants had normal or corrected-to-normal vision. None of the participants reported current dyslexia or dyscalculia, as the task involved character recognition (see below). Participants were of multiple nationalities and replied either in English or in [left blank for anonymization]. In the latter case, quotes of their feedback are translated in what follows and marked with an asterisk.

4.2 Apparatus

The experiment was conducted on a 2020 13-inch Apple MacBook Pro running under macOS Monterey version 12.2.1. Cursor control was performed via a Dell M-BAC-DEL5 MY897 832205-0000 USB Wired 5 Button Optical Scroll Wheel mouse controller plugged via usb, set to 1000 dpi and with the system tracking speed setting set to 6 out of 10 levels. The experiment was displayed on the native built-in retina display 13 inches monitor with a resolution of 2560 × 1600 pixels (227 ppi). The experiment software was implemented as a web application using JavaScript.

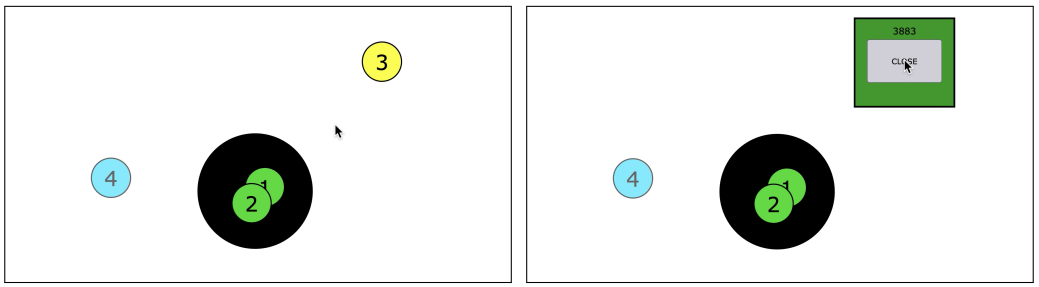
4.3 Tasks

Each trial in the experiment featured a primary task of target acquisition and manipulation. In 40% of trials a pop-up window appeared during the primary task, in which case a secondary task consisted in accurately reporting that window's contents.

4.3.1 Primary task. The primary task consists in bringing numbered visual targets onto a central drop-zone via *pick-and-drop* [25]: clicking a target *picks* it and makes it follow the cursor's positions; clicking again *drops* the target at its current position. The drop-zone is a black disc (3.9 cm diameter) in the background, always visible at the center of the screen. The visual targets are colored discs (1.3 cm diameter) that appear at semi-random locations (see details below) around the drop-zone, and numbered incrementally to indicate their ordering (see Figure 9). Targets appear progressively as the participant completes the trials, always one target ahead of the current task. The current target to pick is displayed in yellow, the next one in cyan. If the current (yellow) target is not entirely contained by the drop-zone after it is dropped, the participant has to pick it and drop it again until it is so. The dropped target (numbered n) then becomes green, the new current target ($n+1$) switches color from cyan to yellow, and a new target (in cyan and numbered $n+2$) appears at a semi-random location—unless $n+1$ is the last target in this block. Figure 8 shows the experiment's appearance.

Target locations were pre-generated to ensure that every block for every participant had a uniform distribution of distances and orientations from the drop-zone. Participants were instructed to complete the primary task quickly and accurately in each trial.

4.3.2 Secondary task. At the beginning of a trial, before the first click on a target (*pick*), a small window could appear, which the participants could not move around. It contained a 4-character *label* and a large `[CLOSE]` button which closes the window when pressed (see Figure 9). The timing, location, and Behavior of these windows' appearance depend on the independent variables of the experiment, detailed below. By default, windows are scheduled to appear with the `[CLOSE]` button on top of the ongoing target (see figure 8-a). Trials with and without pop-ups are henceforth referred to as *Pop-up* and *Normal* trials, respectively. In *Pop-up* trials, after the participant successfully completes



(a) (b)

Fig. 8. Experiment task (a) before and (b) after a popup is displayed (to scale, cropped)

the pick-and-drop task, a multiple-choice question appears above the drop-zone displaying the pop-up window's *label* and three similar-looking strings, in random order, as well as an "I don't know" option and a `[Submit]` button (Figure 9). Participants are instructed to select the option that was displayed on the window and press 'Submit'. If they select any of the four-letter options, a last question inquires whether they are sure of their answer (yes/no). In our results a participant's answer is categorized as *Correct* if they selected the string corresponding to the window's label, *Incorrect* if they selected one of the three other strings, and *Unknown* if they selected the "I don't know" option. After that, the new trial begins.

The three alternative answers were designed to look similar to the *Pop-up* window's (actual) label, so the participant could not guess the correct answer after only a glimpse at the window before it was closed. These alternative answers are composed of the same characters as the actual label, themselves chosen to look similar to each other. Labels and wrong answers could be composed of letters (*e.g.* "NMMN", "MNNM", "MNMN", and "NMNM"), digits (*e.g.* "3883", "8338", "8383", and "3838"), or both (*e.g.* "6GG6", "6G6G", "G6G6", and "G66G"). For this reason, we specified during our participant recruitment process that we were looking for people who do not suffer from dyslexia or dyscalculia.

Unbeknownst to the participant, there were two types of *Pop-up* trials: *Interfering* pop-ups (20% of all trials) are scheduled to appear exactly 100 ms after the cursor enters the ongoing target, to let the cursor move inside the target while maximizing the chances of triggering an interference[3]; *Non-Interfering* pop-ups (20% of all trials) are scheduled to appear early enough that the participant can refrain from clicking them, using a delay defined in an initial, per-participant calibration procedure (see 4.4.1 below) and previous findings [27]. *Non-Interfering* pop-ups were introduced to simulate "regular" pop-ups that do not necessarily conflict with user interaction, and to steer participants' attention away from interferences, our main focus. In doing so, it also allows us to (1) assess whether the differences between *Normal* and *Pop-up* trials are due to interferences or to the mere presence of pop-ups, (2) assess the possible negative consequences of our behaviors on normally-appearing GUI elements, and (3) obtain a baseline for some of our comparison metrics.

We acknowledge that appearing GUI elements cause interaction interferences less than half the time in real life. These 20-20-60 proportions were chosen through pilot tests to satisfy several constraints at once: *Pop-up* trials had to be sufficiently numerous to collect enough data for analysis (especially since *planned Interfering* trials do not always end up causing interferences), but also rare enough to keep the participants' focus on the main task, all while keeping manageable study

[3]Note that this 100 ms threshold does not correspond to a 90 % chance of interference, as [27] Fig. 7a could be read; 100 ms after entering the target does not equate to an SSCD of 100 ms.

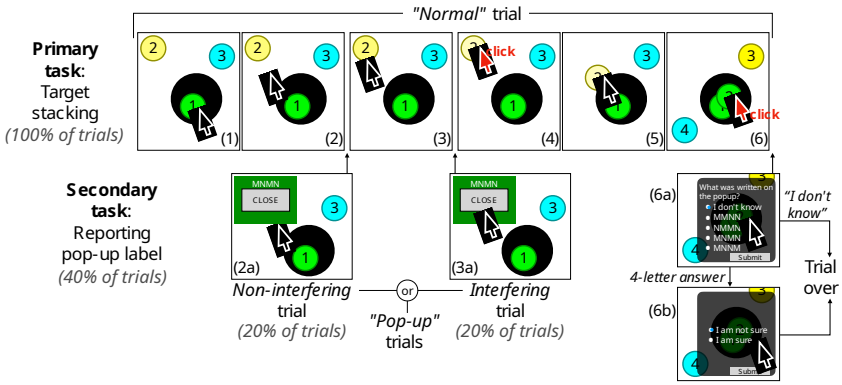


Fig. 9. A trial in our study. (1-2) The participant moves their cursor towards the current target in yellow. (2a) A pop-up window may appear if the trial is *Non-Interfering*, or (3-3a) 100 ms after the user enters the target in *Interfering* trials. (4-6) In any case the participant completes the primary task. The dropped target turns green, the new current target turns yellow, and a new target appears in cyan. (6a-6b) In *Pop-up* trials, the participant is asked what is/was written on the pop-up window. Sizes and distances were adjusted for figure readability; the cursor did not actually turn red on clicks.

duration. In particular, participants should not wait for interferences at every turn but instead maintain a pointing behavior as normal as possible in every trial.

The question about the window's label in *Pop-up* trials (Fig. 9-6a) appears as soon as the target is correctly dropped. With *Non-Interfering* trials, and with some of the tested Behaviors, this means that the pop-up window can still be visible when the question is displayed. We did not give specific instructions about whether to close pop-up windows systematically before answering the questions. This, in turn, also means that the time between the closing of the popup and the appearance/answering of the question is variable. A trial's pop-up window closes automatically once its question(s) are answered.

Occurrences of *Pop-up* trials are semi-random: their order was pre-calculated for each participant in order to maintain the proportions of 60% - 20% - 20% for respectively *Normal*, *Non-Interfering*, and *Interfering* trials within each Behavior condition; like in [27], we also ensured that there could be no consecutive *Pop-up* trials. In the event that a participant is able to click the target before the scheduled appearance of a pop-up window (*Interfering* or not), the trial is automatically reclassified as *Normal* in our logs. If it was an *Interfering* trial, a new *Interfering* trial with the same Behavior and with the same target position is inserted at the end of the block. If the last trial of that block was already scheduled to contain a pop-up, another *Normal* trial is inserted in between, making it two additional trials.

While both Schmid et al.'s [27] and our protocol are designed to evaluate phenomena related to *pointing-based* interferences, our task differs because we are interested in different information and phenomena. First, Schmid et al.'s conditions are either classic target acquisition ("No-stop" trials in [27]) or countermanding ("Stop" trials); it lacks the intentional *Non-Interfering* trials that we use to compare our Behaviors' performance to *Interfering* trials, and our Behaviors' side-effects to *Normal* trials. Second, Schmid et al. focused on precisely measuring the inhibition capability of their participants, making precise variation of SSCD a significant aspect of their protocol; this is unnecessary in our case, and would have made our study longer. Third, we implemented a pick-and-drop task instead of a simple target acquisition because we wanted participants to have a

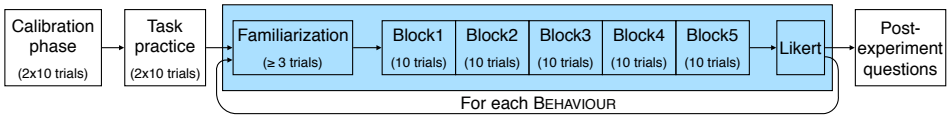


Fig. 10. Main phases and blocks of our experiment protocol.

goal past their first click, even in *Normal* trials. Sequences of target acquisitions are common in real use of a computing device, and it allowed us to further divert the participants' attention away from the prospect of appearing windows. More pragmatically, it also allowed us to control the distance parameter of our trials by bringing the participant's cursor near the center of the screen at the end of every trial. Schmid et al.'s task always started from the location of the previous target, but doing so in a 2D task poses the risk of non-uniform distributions of distances or orientations.

4.4 Procedure

Participants were first invited to sit at a desk in front of the experiment computer and to sign a consent form. They were then explained that the goal of the study was to *"evaluate different appearing behaviors for pop-up windows"*. They were also encouraged to take breaks between blocks or conditions, as often as they needed. The phases of the experiment are illustrated in Figure 10.

4.4.1 Calibration phase. In order to estimate a safe apparition time for the pop-up windows in *Non-Interfering* trials, and to detect whether participants started slowing down as a response to the interfering pop-ups overall [27], we started the experiment with an initial performance calibration phase that also served as training for the primary task.

Participants were instructed to perform 2 blocks of 10 *Normal* trials, *i.e.* with only the primary pick-and-drop task—and without knowing about the secondary task yet. For each participant P we collected the movement times between the start of the trial and the first click on the current target[4], and calculated the regression parameters (slope and intercept) of the "Shannon formulation" [18] of Fitts' Law [10]. These parameters were used in two ways.

First, previous work showed that participants of countermanding-like studies tend to slow their response down after a while [27, 30]. We therefore defined a threshold function that calculates, for any Fitts' Index of Difficulty (ID) within the tested range, the maximum duration above which we consider that participant P is moving significantly slower than during the calibration phase. This was calculated using the following formula, appraised in pilot tests:

$$T_{\max,p} = i_p + ID \times (s_p + 1.96 * SD(MT_p)) \tag{1}$$

with i_p and s_p respectively the intercept and slope of the Fitts' Law regression from participant P's calibration, and $SD(MT_p)$ the standard deviation of their measured Movement Times during that calibration.

When the participant took longer than $T_{\max}$ to click the target, a full-screen message informed them that they had slowed down too much and prevented them to complete the trial for 4 seconds as deterrent. The goal was to ensure that participants could not avoid interferences by altering their speed—consciously or not. We used a conservative threshold (orange line in Figure 11) to avoid interrupting the tasks too frequently while still penalizing obvious slowdowns.

Second, we used the Fitts' Law regression parameters to calculate the display time of *Non-Interfering* pop-up windows. For each *Non-Interfering* trial, the pop-up window is scheduled to appear after a duration chosen at random (uniform) between the beginning of the trial and 100 ms

[4]Pop-up interferences could only happen before the first click of the trial.

before the *expected* click on the target (green area in Figure 11). If the model predicts a picking action shorter than 100 ms, the pop-up window is set to appear as soon as the trial begins. This 100-ms offset was selected during pilot tests, both to avoid too many out-of-script interferences and to allow some variation in the window-spawning times, in order to maintain the illusion that our scripted *Interfering* pop-ups were simply "bad luck".

The calibration phase occurred only once for each participant at the beginning of the experiment (see Figure 10).

4.4.2 Task practice phase. Participants were then informed that from that time on, pop-up windows could appear at random during a trial and that they were to report their contents after successfully dropping the target. No further information regarding pop-ups was provided. Before introducing the new Behaviors—the main independent variable of this study—we wanted participants to become familiar with the secondary task. We therefore introduced a task practice phase in which they would encounter *Pop-up* trials, both *Interfering* and *Not*, but without Behaviors other than **Default**. This phase consisted in 2 block of 10 trials with the same ratios of *Interfering* and *Non-Interfering* trials as the main part of the study. After this began the main data collection.

4.4.3 Behaviors phase. The rest of the study is split in five parts, one per tested Behavior (see Table 3 for a reminder), plus one "**Default**" behavior corresponding to standard appearing windows identical to the *Task practice phase* above. Each Behavior starts with a *familiarization phase* (Figure 12) wherein participants are shown the drop-zone, a single target, a concise explanation of the current Behavior, and a `[Show Behaviour]` button. Clicking the button reveals a pop-up window implementing the current Behavior, complete with a label and a `[CLOSE]` button. Pick-and-drop is active, because some aspects of our Behaviors depend on user actions such as moving or clicking. With **Delay**$_{\text{Postpone}}$ and **HoleOver**$_{\text{Through}}$, for instance, participants might observe by themselves what happens when the cursor moves, but they need a reason to click (here, picking the target) to observe the behavior-cancelling effect of clicking. Dropping the target in or out of the drop-zone does not end a trial or begin a new one. If the target is dropped in the drop-zone, it

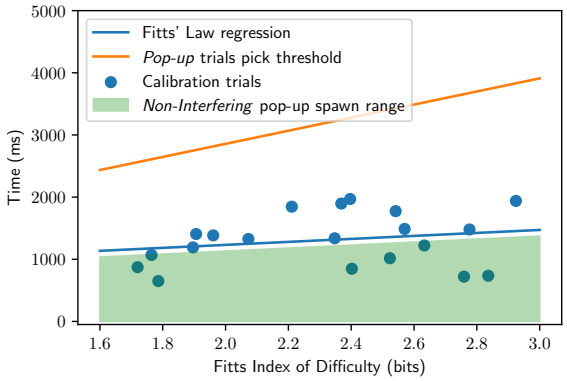


Fig. 11. Fitts Law and time thresholds for Participant 17: the Fitts law regression (blue line) is calculated from the durations of the calibration trials (blue dots). The time range within which the *Non-Interfering* windows spawn (green area) spans from the beginning of the trial (0 ms) to 100 ms before the Fitts regression, depending on the Fitts Index of Difficulty of the target. The threshold to trigger the 4-second warning message also depends on the Fitts law regression and on the Index of Difficulty.

is simply relocated at a random location within the screen. The goal of this *familiarization phase* is to try out the Behavior's response to timing, user input, or other. Participants can 'spawn' as many new pop-up windows as they want until they feel familiar with the Behavior. After three clicks on the `[Show Behaviour]` button, a new button labeled `[Next]` appears at the bottom-right of the screen; clicking it allows the participant to start the first block of this Behavior.

The next 5 consecutive blocks of 10 trials follow the **Tasks** descriptions in subsection 4.3. In particular, each block has a uniform distribution of target distance from the center of the screen (between $d_{\text{min}} = \text{radius}_{\text{dropzone}} + \text{radius}_{\text{target}}$ and $d_{\text{max}} = 0.9 \times \frac{\text{display\_diagonal}}{2}$) and a mostly-uniform[5] distribution of orientation from the drop-zone, and a 60%-20%-20% odds of respectively *Normal*, *Non-Interfering*, and *Interfering* trials.

After the fifth block of each condition, participants were asked to answer several questions on 7-point Likert scales regarding their impression with this Behavior (listed Table 4) before moving on to the *familiarization* phase of the next Behavior.

At the end of the experiment, after the last Behavior, participants were asked to rank the different Behaviors by order of preference. Finally, the experimenter invited the participants to comment on their ranking and on the experiment in general.

4.5 Design

Our study used a within-subject experiment design with pop-up window Behavior (**Default**, **Freeze**$_{\text{Block}}$, **HoleOver**$_{\text{Through}}$, **Away**$_{\text{Displace}}$, **Delay**$_{\text{Postpone}}$) and TrialType (*Normal*, *Interfering*, or *Non-Interfering*) as independent variables. Order of presentation of Behavior was counterbalanced across participants using a balanced Latin square design. TrialType (*Non-*/*Interfering*) was randomized within each block following the proportions and rules described earlier. On-screen target location was pseudo-randomized in order to maintain a uniform distribution of directions and distance

[5]Accounting for the screen's aspect-ratio: not all angles are usable at distances larger than half the display's smallest dimension.

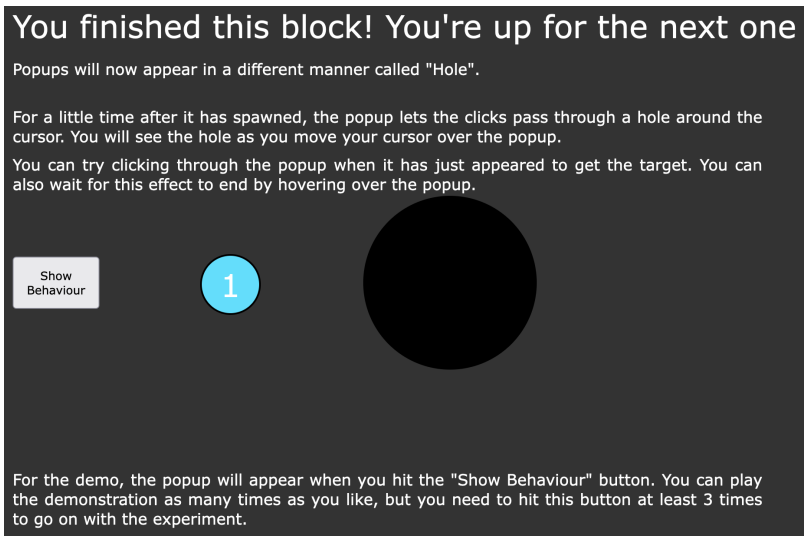


Fig. 12. Screenshot of our experimental interface during the familiarization phase.

Questions about the Behaviors	
I think I would like my popup windows to behave like this in general	
I found the popup behaviour unecessarily complex	
I thought the popup windows were easy to interact with	
I think most people would understand this behaviour very quickly	
I understood this behaviour very quickly	
I found the popup behaviour cumbersome	
I felt very confident that I would not close a popup by accident	
I felt very confident closing the popups on purpose	
Questions the pick-and-drop (primary) task	NA option
I felt in control of the situation in the task	
I felt I had time to change my course of action before clicking on a popup that just appeared	X
I felt surprised by popups	
I felt disturbed in my task by popups	
I was annoyed by popups	
If I could not read the popup content before closing it:	
- I was calm	X
- I was frustrated	X
- I felt responsible	X
- I blamed the system, the popup or my mouse	X
- I felt angry at myself	X
- I felt angry at the system, the popup or my mouse	X

Table 4. Questions asked after each Behavior. NA stands for "Non-Applicable".

to the drop-zone, and to guarantee that all participants had the same target layout for the same Behavior and blocks.

In total, we collected 5 (Behaviors) × 5 (blocks) × 10 (trials) = 250 trials at least[6] per participant. Calibration and task practice trials were not considered in the study analysis. The experiment lasted 57 minutes per participant on average.

5 RESULTS

We collected 5043 trials in total. The initial study plan contained 5000 trials, 2000 of which were planned to feature pop-ups: 1000 aimed to create an interference, 1000 did not. In 192 (9.6%) of these 2000 planned pop-ups trials, the participant clicked the target fast enough to prevent the pop-up's appearance. Only 32 of these trials were aimed to create an interference, thus 32 new *Interfering* trials were added at the end of the same block, as described in subsection 4.3.2, in order to obtain the 1000 interfering pop-up trials that we initially aimed for. In situations where the block already ended with a *Pop-up* trial (11 out of these 32), and since we decided not to have consecutive pop-up trials, we also inserted *Normal* trials between the planned last trial of the block and the added pop-up trial. This adds up to 5000 + 32 + 11 = 5043 trials.

The number of "missed" pop-up trials is strongly unbalanced between planned *Interfering* (32) and *Non-Interfering* (160) trials. We posit that this is because the window of opportunity to click the target before an interfering pop-up is much shorter (100 ms), making it less likely to occur.

[6]Extra trials could be added at the end of blocks when a click occurred faster than a planned pop-up, see subsection 4.3.2.

5.1 Data Pre-processing and Analysis Approach

In what follows and unless specified otherwise, time measurements are aggregated as geometric means per participant and condition for statistical analysis or plotting purposes, as recommended for pointing studies and skewed datasets in general [26]. Other types of numerical data, such as numbers or proportions of correct/incorrect/unknown answers in the secondary tasks, are aggregated as arithmetic means per participant and condition.

Reported significant effects are obtained using Repeated-Measure (RM) ANOVAs performed on block aggregates. Sphericity is controlled using Mauchly's test, and degrees of freedom are corrected using the Greenhouse-Geisser correction when necessary. Post-hoc tests are pairwise t-test comparisons with Bonferroni p-value adjustment method and Tukey's HSD tests when the independent variable had more than two levels. Error bars in bar charts and all confidence intervals (CIs) represent the 95% CIs from bootstrap with 999 resamples over participant aggregates, using the adjusted bootstrap percentile (BCa) method of the `boot.ci`[7] statistical package.

Likert-scale answers and Behavior rankings are analyzed using Friedman tests with Behaviors as groups and participants as blocks. We ran post-hoc pairwise comparisons using Wilcoxon rank sum test with continuity correction, using Bonferroni p-value adjustment.

Finally, qualitative feedback was coded by one of the authors in an iterative process of deductive thematic analysis [5], focusing on themes related to (1) the Behaviors themselves, (2) emotional responses to the interferences and to the pop-up Behaviors, and (3) recurrent themes that participants talked about such as notifications or the influence of the task on the appreciation of a pop-up Behavior. This feedback is reported with examples and descriptive statistics.

In what follows we explore our results by theme rather than by data type (*e.g.* time then errors then quantitative-subjective then...), applying the principle of triangulation [17] to strengthen our findings and conclusions.

5.2 Triggering Interferences

Our experimental protocol displayed *Interfering* pop-ups 100 ms after the mouse pointer entered the target, which differs from Schmid et al.'s procedure [27] that relied on a threshold value continuously updated from each participant's behavior throughout the experiment. Therefore, before concluding anything about the effectiveness of the tested Behaviors, we need to ensure that our conditions reasonably triggered the desired phenomenon. In particular, we need to ensure that (1) trials scheduled to be *Interfering* (often) contained interferences, (2) trials scheduled to be *Non-Interfering* did not trigger (too many) interferences, and (3) participant behavior during the initial reaching phase of the primary task did not significantly change as a result of either experiment progression or pop-up Behavior.

5.2.1 From user interactions. Note that, as opposed to the countermanding protocols used in [27, 30] there is no definite way for us to label some trials as "successfully interfered with" when the click occurs, *e.g.* X ms and more after the window popped up. Indeed, some participants can be more alert, or faster readers than others. Our main guideline comes from Schmid et al.'s finding of 350 ms as the 50% mark for successful click inhibition in a standard 1-D pointing task [27].

Figure 13 shows the distribution of delays between pop-up appearance and the first `mousedown` event of the trial for each TrialType and for all Behaviors (negative times thus correspond to a first click happening before the pop-up spawned). We observe that more than 60% of all intended *Interfering* trials had their first click *earlier* than 350 ms after the pop-up appearance, and more than 67% of all *Non-Interfering* trials had their first click *later* than 350 ms after the pop-up appearance.

[7] https://www.rdocumentation.org/packages/boot/versions/1.3-28/topics/boot.ci

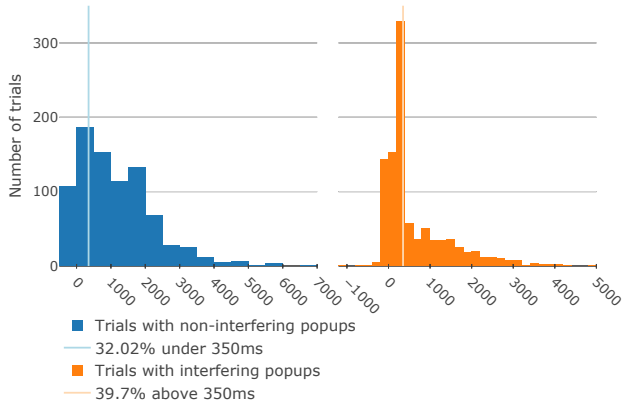


Fig. 13. Distribution of the time between the appearance of a window and the first click of the trial, for all Behaviors. Vertical lines indicate the 350 ms mark.

If we remove **Delay**$_{\text{Postpone}}$ from the set, since it alters the *timing* of the window appearance[8], we obtain an "interfered-with" ratio of 53 %.

While a 53-60 % ratio of successfully triggering interferences may seem quite low, we stress that this phenomenon is probabilistic in nature. Schmid et al.'s results correlate the Stop-Signal-to-Click Delay (SSCD) to the chances of successfully inhibiting a planned click, so there is no "sure-fire" way to trigger it. Plus, precisely implementing an SSCD implies to have extremely accurate estimates of the user's pointing performance and behavior, in all situations rather than on average, which is easier to achieve in a 1D, single-task protocol. Instead, we chose to use target entry as a reference frame because it is a mandatory step in selecting the target, and under the hypothesis that selection occurs shortly after it. This is of course an imperfect approximation: in case of *e.g.* fast-approaching ballistic movements that result in target overshoot, the window might pop up before the final phase of the pointing action, *i.e.* before the participant "sent the (motor) order" for the click. For reference, a 55-60 % success rate corresponds to a Stop-Signal-to-Click Delay of approximately 300-350 ms (Fig. 7a and 9 in [27]).

We also highlight that our 4-string labels were designed to be difficult to parse, being composed of similar-looking characters in an order that matters for the post-drop question. The average reading speed is estimated around 330 ms per word in English [6], but with known words. The combined durations of noticing the window, inhibiting one's planned click, reading the precise sequence of characters, and committing it to memory (including distinctions such as "G6G6" vs. "6GG6") likely took longer. We could therefore use a higher threshold—assuming that participants were dedicated to completing the tasks as well as possible—but we decided against it because instrumenting participants to measure precise reading phases would have added more time and discomfort to the study. We thus maintain our 350 ms as a conservative threshold, and its corresponding 53-60 % probability, while keeping open the possibility that a number of interferences might have occurred after longer delays.

[8]All values below 0 in Fig. 13 are **Delay**$_{\text{Postpone}}$ since it is the only Behavior in which a first click occurring after the *scripted* pop-up time can still occur before the window appears.

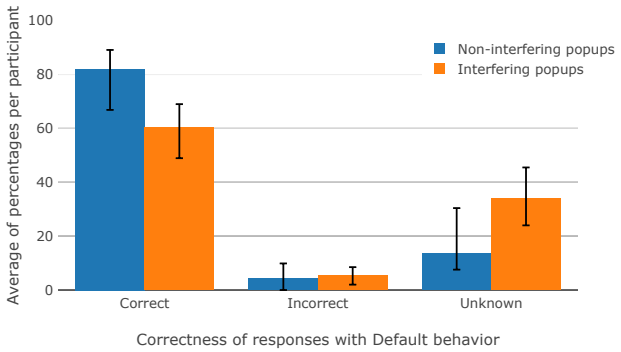


Fig. 14. Proportions of *Correct*, *Incorrect*, and *Unknown* answers to the secondary task with the **Default** behavior, as a function of TrialType.

5.2.2 From secondary task answers. The secondary task consisted in accurately reporting the label of the pop-up window, which is (by design) harder to do if that window was closed prematurely. Figure 14 shows the percentages of *Correct*, *Incorrect*, and *Unknown* answers gathered in the **Default** condition—the other Behaviors are designed to facilitate the secondary task—and separated by TrialType. We observe a noticeable drop in correct answers in the *Interfering* trials, that matches an increase of *Unknown* answers.

A RM-ANOVA found a significant effect of TrialType on the percentage of correct answers in **Default** trials ($F_{1,19} = 15.985, p < .001$), with *Non-Interfering* trials resulting in significantly more correct answers (82.04%, $CI = [67.64, 89.16]$) than *Interfering* ones (60.5%, $CI = [48.39, 69.5]$). Conversely, a significant effect of trial type was also found on the percentage of *Unknown* answers ($F_{1,19} = 14.905, p < .01$) with *Non-Interfering* trials resulting in fewer "I don't know" answers (13.68%, $CI = [7.44, 35.38]$) than *Interfering* ones (34%, $CI = [24.5, 46.95]$). The difference between the percentages of wrong answers with *Non-Interfering* (4.28%, $CI = [1.09, 9.75]$) and *Interfering* (5.5%, $CI = [2.5, 9]$) was not found significant.

The amount of *Unknown* answers is a good indication that participants "played along" and did not try to guess at least some of the labels that they did not see. In the worst case, assuming a participant who purposefully selects a label without any idea of what was displayed on the pop-up window, we can expect a 75 % probability of selecting the wrong answer; considering that there are only about 4.9 % of *Incorrect* answers in our results, we can deduct that up to roughly 1.6 % of *Correct* answers could be caused by luck—if all participants intently ignored our instructions all the time.

Participants in our baseline condition, **Default**, provided significantly fewer correct answers and more "I don't know" answers in *Interfering* trials, which corroborates with our goal of triggering interaction interferences. These values will serve as comparison baselines later in this section.

5.2.3 From subjective feedback. Participant feedback about the task in general, and about the **Default** condition in particular, expressed issues that we confidently assign to interaction interferences. 9 participants out of 20 reported frustrations referring to interferences, for instance complaining of being "*annoyed by* [pop-ups] *appearing*" [P8], that in daily life they "*sometimes*

appear suddenly like in the first exercise" [P6, referring to task practice with **Default**], or that "*as soon as you click it is over*"*[9] [P11].

2 participants explicitly referred to the unpredictability of pop-ups in itself, notably that they "*did not clearly understand the pattern of their appearance, and as a result it was more disturbing and frustrating*" [P6] and that it was "*very frustrating when you do not know when one might appear and disappear instantly*" [P17].

As a side-note on pop-up interferences "in the wild", 7 participants volunteered that they try as hard as they can to forbid all pop-up windows in their daily lives, deactivating them everywhere possible and using pop-up blockers online; three of them even deactivated every notification on their smartphones, and one simply doesn't have internet on their phone to avoid having too many notifications.

5.3 Addressing Interferences Overall

We then investigated whether the proposed pop-up window Behaviors successfully helped reduce the occurrences of interaction interferences or mitigate their effects.

5.3.1 From secondary task results. Part of the answer can be found by observing the proportion of *Interfering* pop-ups that were closed, as illustrated in Figure 15. We found a significant effect of Behavior on the percentage of popups closed in *Interfering* trials ($F_{1.49,28.29} = 0.81, p < .001$). Both **Default** and $\textbf{Freeze}_{\text{Block}}$ (average pop-ups closed 100%, $CI = [100, 100]$) always required participants to close the pop-up windows before accessing the targets, and both were found significantly different from $\textbf{Away}_{\text{Displace}}$ (33.5%, $CI = [17.5, 53]$), $\textbf{Delay}_{\text{Postpone}}$ (36%, $CI = [19.5, 55]$) and $\textbf{HoleOver}_{\text{Through}}$ (42.5%, $CI = [26.61, 60]$). No significant difference was found between **Default** and $\textbf{Freeze}_{\text{Block}}$, nor between $\textbf{HoleOver}_{\text{Through}}$, $\textbf{Delay}_{\text{Postpone}}$, and $\textbf{Away}_{\text{Displace}}$.

These results are consistent with the nature of each Behaviors, highlighting two clear groups: a first group wherein every Behavior allows would-be interfered clicks to reach the target: by keeping the pop-up away ($\textbf{Delay}_{\text{Postpone}}$ and $\textbf{Away}_{\text{Displace}}$) or by passing the click event through ($\textbf{HoleOver}_{\text{Through}}$); and a second group that capture and interpret that click, either normally (**Default**) or with a tolerance for interferences ($\textbf{Freeze}_{\text{Block}}$). Note that while a $\textbf{Freeze}_{\text{Block}}$ window does need to be closed to access the underlying target, the $\textbf{Freeze}_{\text{Block}}$ behavior can effectively prevent it to be closed (or interacted with) by accident.

We did not analyze the *time* it took participants to close pop-up windows in *Interfering* trials, because whether a pop-up even needs to be closed is strongly dependent on the tested Behavior, as demonstrated above. Participants closed the pop-ups less often with $\textbf{Away}_{\text{Displace}}$, $\textbf{Delay}_{\text{Postpone}}$, and $\textbf{HoleOver}_{\text{Through}}$ than with **Default** and $\textbf{Freeze}_{\text{Block}}$, so the analysis would be run on strongly unbalanced population sizes. Plus, these data-points could be biased towards unusual or user-specific behaviors (for instance, some participants felt like closing pop-up windows before answering the four-choice question even when they knew the answer with confidence). We present in Subsection 5.4 an analysis of overall trial completion time.

5.3.2 From secondary task answers. Success in answering the secondary task, in comparison with the **Default** condition, is another indicator of interference-mitigation success.

We found significant effects of Behavior ($F_{1.69,32.04} = 13.81, p < .001$), TrialType ($F_{1,19} = 6.55$, $p < .05$), and Behavior × TrialType ($F_{1.92,36.55} = 11.51, p < .001$) on the average percentage of **correct answers**, illustrated in Figure 16. TrialType only affects that percentage with **Default** ($p < 0.001$), with *Interfering* trials (60.5%, $CI = [48.39, 69.5]$) having fewer correct answers on average than *Non-Interfering* (82.04%, $CI = [67.56, 89.03]$).

[9]Quotations with an asterisk were translated from [left blank for anonymization].

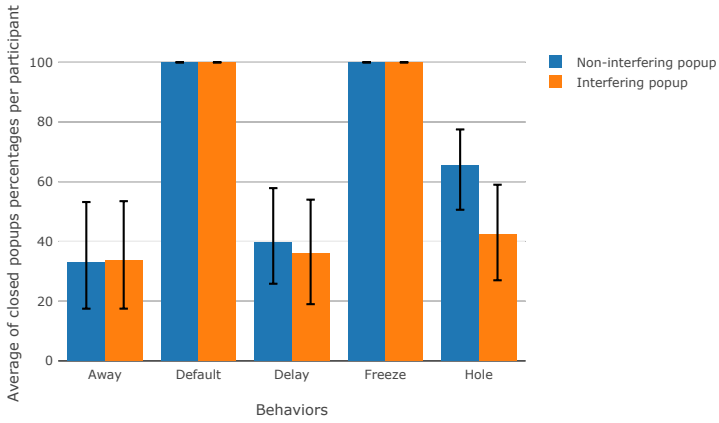


Fig. 15. Percentages of closed pop-up windows as a function of Behavior and TrialType.

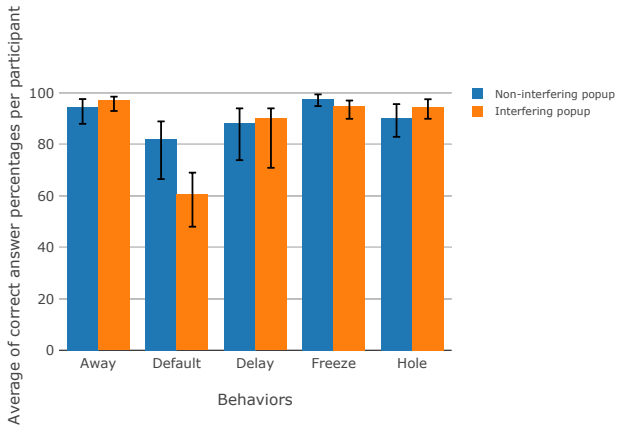


Fig. 16. Percentages of correct answers to the secondary task as a function of Behavior and TrialType.

Regarding Behaviors alone, post-hoc tests revealed that participants selected significantly fewer correct answers with **Default** (70.17%, $CI = [54.95, 76.16]$) than with $\textbf{Away}_{\text{Displace}}$ (95.97%, $CI = [90.67, 98.11]$), $\textbf{Freeze}_{\text{Block}}$ (95.95%, $CI = [93.75, 97.7]$), $\textbf{HoleOver}_{\text{Through}}$ (92.69%, $CI = [87.78, 95.86]$), and $\textbf{Delay}_{\text{Postpone}}$ (89.26%, $CI = [74.34, 95.16]$), all $p < 0.05$.

These significant differences are maintained when we only consider *Interfering* trials ($F_{2.04,38.7} = 20.97$, $p < .001$), with **Default** having significantly fewer correct answers (60.5%, $CI = [48, 69]$) than $\textbf{Away}_{\text{Displace}}$ (97%, $CI = [93.06, 98.5]$), $\textbf{Freeze}_{\text{Block}}$ (95%, $CI = [90.5, 97.5]$), $\textbf{HoleOver}_{\text{Through}}$ (94.5%, $CI = [88.5, 97]$), and $\textbf{Delay}_{\text{Postpone}}$ (90%, $CI = [70.5, 96]$), all $p < 0.05$.

Conversely, when only considering *Non-Interfering* trials, and even though the analysis of variance revealed a significant effect of Behavior ($F_{1.81,34.4} = 3.898$, $p < .05$), the post-hoc Tukey tests found no significant difference between any of the individual Behaviors.

The opposite effects (**Default** ≫ other Behaviors) were found for the proportions of "**I don't know**" answers. No significant effect of Behavior nor TrialType was found on the percentage of **wrong answers**.

Participants provided at least about 30 pp[10] more correct answers in *Interfering* trials with the candidate Behaviors (see Figure 16), suggesting that they successfully mitigated the functional impact of interferences overall.

5.3.3 From subjective feedback. Friedman tests revealed no significant effect of Behavior on any of our Likert-scale questions. This is interesting in contrast to the results of the secondary task, which would tend to indicate that pop-up window Behaviors helped participants avoid closing the *Interfering* pop-up windows too quickly. This can have several explanations: considering the relative rarity of *Interfering* pop-up trials, the Likert responses could reflect the overall tasks operationalization more than the Behaviors themselves. Another possibility is that the interference-related advantage brought by the Behaviors is counterbalanced by negative side-effects when they operate. We however take note that our Behaviors do not seem, as a whole, to worsen user experience.

While we did not find a statistical difference between participants' responses to these statements, open comments suggest on the other hand that some participants still felt differences compared to **Default**. More precisely, 20 comments from 9 participants mentioned interference-like issues, 8 among which, written by 4 participants, compared specific Behaviors to occurrences or consequences of interaction interferences.

For instance, regarding **Away**$_{\text{Displace}}$, P11 reported that the pop-up window had "*a lot less chances of being* ***closed by accident***. [They] *have seen all the popups without being disturbed, except one that* [they] *noticed only once* [they were] *asked for the content.*" P17 stated that "*When popups appeared far from the mouse it was not that bad, but other times* [they] *would* ***close the popup too fast.***"

Regarding **Delay**$_{\text{Postpone}}$, P4 reported they were "*consistently able to read the content of the popups.* [They] *never* ***closed them by mistake****; rather, they were closed automatically by the system after telling what the popup said.* [P4] *could read the popup every time, because it was still open.*" P6 stated that "*having a delay is interesting, it feels less aggressive and* [they] *feel more comfortable dealing with* [pop-up windows].".

With regard to **Freeze**$_{\text{Block}}$, P6 reported that "*Having the freeze saved* [them] *from* ***clicking too fast and not seeing the pop up****, but* [they] *would have* [clicked it] *without it.*".

P6 also concluded that, overall, "*Except "****Default****"* [they] *did not have any interference.*".

5.4 Side-effects of Behaviors

Interaction interferences can be extremely frustrating, but remain relatively infrequent [27]. While it is desirable to minimize their occurrences or consequences, it should not be at the expense of cumbersome or frustrating default GUI behavior. We therefore investigated their impact on *Non-Interfering* trials specifically, using *Normal* trials as a baseline for standard interaction.

5.4.1 From user inputs. Figure 17 illustrates overall trial duration in blue, and target picking duration (*i.e.* between the beginning of the trial and the first click on the primary target) in orange, per Behavior, for *Normal* and *Non-Interfering* trials.

With *Normal* trials, we found no significant effect of Behavior on either total trial time or target picking time (all $p < 0.05$). This is a sanity check that pop-up-less trials were not affected by pop-up behaviors.

[10]"pp" is short for "Percentage Point", *i.e.* the arithmetic difference of two percentages.

In *Non-Interfering* trials, however, we found a significant effect of Behavior on total trial time ($F_{2.99,56.79} = 9.03$, $p < 0.001$), with significant differences between **Delay**$_{\text{Postpone}}$ (mean 2385 ms, $CI = [2088, 2710]$) and both **Default** (3179 ms, $CI = [2856, 3539]$) and **Freeze**$_{\text{Block}}$ (3224 ms, $CI = [3014, 3450]$). We also found a significant effect of Behavior on target picking time ($F_{2.87,54.51} = 29.74$, $p < 0.001$), with significant differences between, on the one hand, **Default** (2385 ms, $CI = [2088, 2679]$) and **Freeze**$_{\text{Block}}$ (2438 ms, $CI = [2242, 2597]$), and on the other hand, **Away**$_{\text{Displace}}$ (1627 ms, $CI = [1350, 2034]$), **Delay**$_{\text{Postpone}}$ (1252 ms, $CI = [1124, 1511]$), and **HoleOver**$_{\text{Through}}$ (1744 ms, $CI = [1533, 2002]$).

These results indicate neutral to beneficial impacts of the candidate Behaviors on the completion times of the primary task when compared to the **Default** condition in *Non-Interfering* trials, *i.e.* when pop-up windows had little chance to trigger an interference. Even though the candidate Behaviors are not designed to improve *Non-Interfering* situations, **HoleOver**$_{\text{Through}}$, **Away**$_{\text{Displace}}$, and **Delay**$_{\text{Postpone}}$ made the primary task faster, probably because they eliminated the need to close the pop-up windows. This is likely a consequence of our task operationalization, however, and we do not conclude that these Behaviors could improve overall time performance in real-life situations.

Nevertheless, these results do indicate that our pop-window Behaviors do not impair time performance.

5.4.2 From subjective feedback. Participants' preference ranking of the five Behaviors are illustrated in Figure 18. A Friedman test found a significant effect of Behavior on these rankings ($\chi^2(4) = 15.56$, $p < 0.005$), and post-hoc tests identified significant differences between **Default** (median rank 4.5) and **Away**$_{\text{Displace}}$ (2), **Delay**$_{\text{Postpone}}$ (2.5), and **Freeze**$_{\text{Block}}$ (2), as well as between **HoleOver**$_{\text{Through}}$ (4) and **Away**$_{\text{Displace}}$ and **Freeze**$_{\text{Block}}$ (all $p < 0.05$ or less). The open comments that we collected draw a similar picture, with more detail about which aspect of each Behavior was appreciated, which one was disliked, and what can be done to improve some of them.

Despite having found no clear effect of Behaviors on any of our Likert questionnaires, we identified a number of usability and design insights from participants' numerous open comments regarding what they did or did not appreciate with each technique.

***Delay**$_{Postpone}$*. **Delay**$_{\text{Postpone}}$ was ranked consistently across the range (first and last twice, median 2.5) and obtained mixed comments. Five participants expressed an appreciation for its working principle, for instance reporting "*never closing pop-ups by mistake*" [P4] or finding it "*more comfortable*" [P6]. However, we also identified two consistent caveats.

First, displaying the pop-up window immediately after the first click could give participants the impression that its appearance was somehow triggered by their action, or that they clicked the pop-up instead of the target by a few milliseconds, sometimes even prompting them to quickly re-click the pop-up window (*e.g.* "***Delay**$_{Postpone}$*[was] *closed a lot by accident*"* [P14], "*it does not interrupt your task at the beginning, but it creates doubts if it is an error you did; maybe you did not* [click] *the target and left it behind, or it is the pop up showing up*" [P3]). As a consequence, some participants reported an overall sentiment of unpredictability [P16], or split-second impressions of having closed a pop-up by accident even though they had not. This was however not reflected in the amount of closed pop-up windows (Figure 15).

Second, a number of participants (3) explicitly specified that their positive opinion of **Delay**$_{\text{Postpone}}$ could be conditional to their activity, depending in particular on the relative importance of the task at hand vs. the information on the pop-up window (*e.g.* "[**Delay**$_{\text{Postpone}}$] *becomes irritating when we have a task to do, but if we have to read any information, this is the best*" [P10]).

These comments indicate that users could benefit from additional delay between the click and the pop-up's appearance, for overall usability if not directly for performance purposes. That would

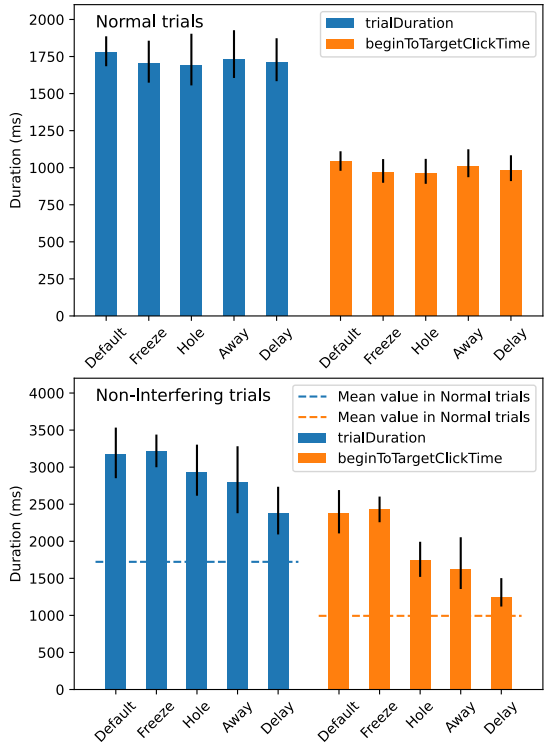


Fig. 17. Total trial duration (in blue) and target acquisition duration (in orange) for each Behavior, in the *Normal* trials (top graph) and in the *Non-Interfering* trials (bottom).

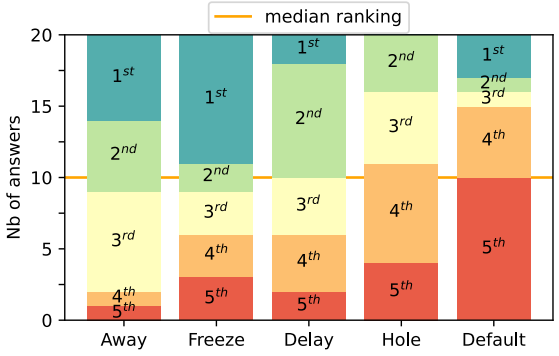


Fig. 18. Stacked histogram of participant rankings per Behavior.

however postpone the display of important information even further, which some participants hypothesized could become problematic depending on the task at hand. Another possibility would be to design **Delay**$_{\text{Postpone}}$-specific appearance feedback (*e.g.* fading-in) that would indicate that a given pop-up was waiting to be displayed. These design adjustments need to be assessed in further studies.

***Away**$_{Displace}$.* **Away**$_{\text{Displace}}$ received among the best rankings overall: more than half the participants ranked it first or second, and only two participants ranked it 4th or 5th (median 2). It also received a number of positive feedback (21 comments, 14 participants), praising *e.g.* that "*one can stay focused on what they are doing, while seeing what else is happening on the screen*"* [P5], that **Away**$_{\text{Displace}}$ "*has a lot less chances of being closed by accident*" [P11], or that **Away**$_{\text{Displace}}$ was less/least annoying Behavior [P1, P5, P16]. As downsides, 3 participants found that the location of the pop-up felt random and unpredictable; and 3 participants mentioned that in a real setting it could become tedious to close them if they ended up too far (*e.g.* "*if I had to actually close each popup I would be annoyed*" [P17]).

Overall the open comments about **Away**$_{\text{Displace}}$ were primarily positive, the main issue being related to the unpredictable location of the pop-up window. This is very encouraging considering that our pop-up-locating algorithm was a first version and likely improvable, *e.g.* making the pop-up appear closer to the cursor (although at the risk of triggering more interferences) or at a designated location, as some participants suggested.

***Freeze**$_{Block}$.* Despite its longer additional delay (Figure 17) and the obligation to close the pop-ups, **Freeze**$_{\text{Block}}$ received numerous positive comments (23, from 13 participants) and received among the best rankings (median 2) with the highest number of 'first' (9 participants). Among its appreciated features, it increased confidence and control (*e.g.* "*I knew I wasn't going to miss the information*"* [P19], "*Way more managable* [than **Default**] [...] *I could be fearless to try to do* [the task] *fast*" [P3]) and was "*very easy to follow*" [P10]. We collected 3 negative comments from 3 participants, two of them referring to the inherently blocking nature of **Freeze**$_{\text{Block}}$'s behavior (the last one did not specify what they disliked).

In terms of both positive and negative comments, **Freeze**$_{\text{Block}}$ was a participants' favorite even though its interrupting principle and some of its quantitative effects could have hinted otherwise. One participant even mentioned that they preferred **Freeze**$_{\text{Block}}$ "*because it changes color*" [P8], which adds to our primary conclusions that richer visual feedback could improve our Behaviors.

***HoleOver**$_{Through}$.* Comments about **HoleOver**$_{\text{Through}}$ were mostly critical, in line with its ranking which was the worst among the candidate Behaviors (median 4): more than half the participants ranked it 4th or 5th, and no-one ranked it first. Even its working principle was confusing to some (*e.g.* "*It felt slightly odd* [...] *to click through the popup*" [P17], "*I thought of my mom she wouldn't have* [understood]" [P8], "[it felt] *unnatural to click through the windows*"* [P11]) or even disturbing ("*This* [behavior] *seemed very disturbing to me, it took me a long time to get used to it*"* [P5]). 2 participants felt that it was overly complicated and in particular required more focus than the other Behaviors (*e.g.* "*Was hard at the beginning to deal with the pop up content, the number to grab and the hole animation at the same location on the screen*" [P6]). 2 participants specifically complained that they felt not in control of the Behavior's duration, making them "*doubt in every click if* [they were] *clicking the target through the pop-up or clicking the pop-up*" [P2]. Finally, even its core goal was criticized, as 3 participants mentioned that **HoleOver**$_{\text{Through}}$ made them click pop-up windows when they did not intend to.

We intended the **HoleOver**$_{\text{Through}}$ technique as a post-hoc indication that a potentially interfered click had been sent to the target underneath, not as a "mode" during which users should interact

while they can. However, during our post-experiment discussions we realized that some participants tried to actively aim at the target through the hole while the Behavior was still active, which would explain some comments about having insufficient control over its duration and about adding to the tasks' complexity.

Default aside, **HoleOver**$_{\text{Through}}$ was the least appreciated Behavior evaluated in our study. There seems to have been a miscommunication when we explained what the behavior does and to what purpose. Similar to the previously discussed Behaviors, we consider it possible that adjusting its visual feedback may better convey what its purpose and meaning are (and aren't!).

Default. Our experiment's baseline condition received the worst ratings of all Behaviors (median 4.5). It was ranked 5$^{\text{th}}$ by half the participants and 4$^{\text{th}}$ by five more. Being the condition "without any special behaviour" (Table 3), the comments about **Default** were more about pop-ups and interferences in general, or used as a reference to explain why a given candidate technique was preferred. Three participants felt like **Default** was the most understandable or predictable behavior. On the other hand, the simplicity of **Default** also has its upsides, as one participant noted that "[With **Default**] *you don't feel guilty. You cannot undo what the system is doing*" [P18].

6 DISCUSSION

In the following, we further discuss these results and elaborate on the limitations of our work.

6.1 Overall Utility of *Naive* Behaviors

The results of our experiment suggest that the candidate behaviors can already reduce the occurrences of (some) pointing-based interferences without impeding the user's ability to perceive the content of the GUI change. Three of them, **Freeze**$_{\text{Block}}$, **Away**$_{\text{Displace}}$, and **Delay**$_{\text{Postpone}}$, were consistently preferred over the **Default** behavior (Figure 18), and the comments we gathered from our post-experiment questionnaires hint at a number of simple feedback refinements that could improve user satisfaction even further.

However, we acknowledge that a one-hour experiment can be a limited platform to study such an un-systematic phenomenon. First, despite our efforts to make them as infrequent as possible during the study, there is a risk that the unusually high rate of interferences affected the participants' subjective answers—positively or negatively. Second, even though we tried to design *Naive* behaviors that operate as innocuously as possible, there exists a risk that their systematic nature may cause more harm in day-to-day use than in the confines of an experiment clearly designed to characterize them[11]. For these reason, despite rather positive results with *Naive* behaviors, we still plan to design and evaluate more "intelligent" approaches that aim to *Detect* or even *Predict* interferences in order to make unusual behaviors even rarer.

6.2 Feasibility and Technical Requirements

The four candidate behaviors evaluated in our study have technical requirements that we consider among the easiest to fulfill. **Freeze**$_{\text{Block}}$ and **HoleOver**$_{\text{Through}}$ require the ability to change the behavior of graphics elements, both in terms of visual appearance and of events handling. This can be implemented relatively easily, *e.g.* on webpages with the `MutationObserver`[12], or as an additional default behavior in most window managers. **Freeze**$_{\text{Block}}$ can even benefit from existing window states such as Qt's `enabled` property or JavaFX's `disabled` property, which can

[11]This in fact belongs to a more general paradox when investigating methods to prevent unwanted GUI behaviors: a perfect solution might prove imperceptible to users, because it is difficult to notice the absence of an already rare, unwanted event.
[12]https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver

be changed with simple method calls. The specific visual feedback of **HoleOver**$_{\text{Through}}$ could demand some extra work, but its value is anyway put to question by our participants' feedback; a quieter implementation of the Mapping behavior type where events are sent to the element below without visual feedback would be even simpler to implement, but at the possible cost of confusion as to whether the event was interpreted as intended.

Away$_{\text{Displace}}$ and **Delay**$_{\text{Postpone}}$ require the ability to monitor user events, which can already be done at system level with most operating systems via global mouse listeners, but also to control the default time or position at which the change occurs. The latter two can be more challenging to implement on top of an existing application since GUI toolkits rarely expose event handlers for GUI elements that are *about* to change, and waiting for a change to occur before immediately altering it can cause problematic visual flicker. Such solutions need to be investigated on a toolkit-by-toolkit basis for feasibility.

Connecting these requirements to our study results, we expected to find a correlation between the satisfaction with a Behavior and its *Technical Requirement*s, *i.e.* the most appreciated behaviors would be the most technically demanding. We were therefore surprised that **Freeze**$_{\text{Block}}$, which we considered our most basic Behavior (and the only one unable to prevent an interference, only its worst consequences) ended up being ranked first the most. Even though our participant comments offer promising leads to improve **Delay**$_{\text{Postpone}}$, **Away**$_{\text{Displace}}$ and **HoleOver**$_{\text{Through}}$, this highlights the interesting possibility that a simple and predictable solution could be preferred even when its best-case consequences are worse. Then again, user feedback may differ in day-to-day use when users are not specifically directed to think about these behaviors, and where interfering pop-ups (and pop-ups in general) are less of a constant threat.

Design adjustments could improve the tested behaviors, but must however be done carefully. Too little feedback can disrupt the user – as with **Delay**$_{\text{Postpone}}$ – but too much feedback can also be overwhelming – as with **HoleOver**$_{\text{Through}}$. To the user, realizing that an interface change has occurred at the very moment that they have generated an input is a mental effort in itself, whether the whole situation triggered an interference or not. When deploying interference countermeasures, we potentially add to this effort the need to understand *where* the click went, since the system's interpretation no longer follows the chronological order of events. This may explain the success of **Away**$_{\text{Displace}}$: by eliminating the co-location of events, the effect of the input is no longer temporally ambivalent.

More work is needed to single out a set of "best" behaviors, but our study demonstrated that simple and arguably achievable behaviors can already decrease the occurrences of pointing-based interferences without harming GUI readability or user experience. Future work should investigate the implementation of these Behaviors in real systems, and its implications on existing window managers, notifications, and input events handlers.

As a concluding note, we had to define many of our Behaviors' parameters by hand using pilot studies: their durations, their various triggers, the location of the appearing window with **Away**$_{\text{Displace}}$, etc.; and we could only test *Naive* behaviors, which leaves many interesting possibilities for all of these parameters. Our controlled experiment was designed primarily to establish the feasibility, performance, and limitations of the principles behind our behavior types; as such, there remains a lot to do before we can present a definitive set of techniques that can adapt to any task, context, device, etc. Finer tuning of these numerous relevant parameters would have generated too many conditions to test in this exploratory work, but it is a very promising next step in our investigation of this problem space.

6.3 Applicability to Other Types of Interferences

Schmid et al. reported a number of examples of real-life interferences provided by the participants of an online survey [27]. Most of these examples involved a pointing action as triggering event, followed in frequency by keyboard events, and a minority of gesture events. We devised our design space of solutions to be input-agnostic, but had to settle on a single type of interference-triggering GUI events in our experiment to keep its duration manageable. We chose pop-up windows as they are among the most common and recognizable interferences around [27]. We also expect that it is as good a stimulus as any other GUI event to trigger *pointing-based* interferences, since the physiological ability to inhibit a click likely does not depend on the nature of the change. However, that nature does affect the GUI behaviors that can be applied to avoid it or to alleviate its consequences, and some of the resulting Behaviors would arguably only apply to that category of interaction interferences: what could it mean to **HoleOver** an app taking the focus, when the target element is not hidden as a result? or to apply **Away** to word suggestion updates of a gesture keyboard?

We however see value in many of the design space's categories of behaviors and requirements for other types of interaction interferences. We will now discuss how our results and *Pop-up*-specific Behaviors can generalize to other forms of pointing-based interferences, structured around the four examples given in the introduction. In them, the intended target of a pointing action either:

- became **occluded** (*e.g.* a notification appeared above the target right before the click),
- **moved** (*e.g.* trying to click a link when the webpage unexpectedly updates),
- **changed** (*e.g.* the word suggestion above a gesture keyboard changes just before the tap),
- or **disappeared** (*e.g.* a notification disappears right before the user clicks to close it).

6.3.1 Displace. The Displace behavior type, of which $\textbf{Away}_{\text{Displace}}$ is derived, can apply to any interference whose trigger is location-dependent. Its implementation needs to be case-specific, however, compared to *Pop-up*-style interferences. The "**occluded** target" situation was operationalized as a pop-up window in our study, but other examples such as notification panels can also be displaced when an interference is expected to occur. In situations when whole applications take the foreground and focus, *e.g.* as a result of a planned prompt, it may be complicated to move a full-screen app out of the way of the cursor.

When the intended click target **moves**, it would seem strange to want to Displace it some more. But it can make sense in situations where the viewport of the target can itself be manipulated. Take the example of an emails list that is updated by the arrival of a new email, right as the user was about to click another email. The update typically shifts all emails down by one slot, resulting in the user clicking the email (previously) above their target. **Away** cannot be directly applied here, but a Displace-type of behavior could consist in appending the new email to the top of the list without automatically shifting/scrolling its contents.

Displace behaviors make less sense for situations where the intended target is **updated**, since the click will still likely occur *somewhere* unintended. The same goes for **disappearing** targets.

6.3.2 Postpone. The Postpone behavior type, of which $\textbf{Delay}_{\text{Postpone}}$ is derived, is rather universal in its application because interaction interferences themselves are essentially temporal phenomena. Any potentially interfering interface change (**occlusion**, **move**, **change**, **disappearance**) can be postponed to a safer time, assuming that interference risks can be accurately estimated—and that the initial timing of said change is not important. The real challenge here lies in estimating *when* and *how long* to postpone an update, as our study already showed issues with the simplest approach to apply the change immediately after we estimate the risk gone: participants felt like their click caused the window appearance. In the other extreme, a too-cautious approach can cause extended

delays. This begs the question of how to provide visual feedback to the postponing behavior, or whether the system should provide visual feedback at all.

6.3.3 Block. Similarly, the principle of the Block behavior type (of which **Freeze**$_{\text{Block}}$ is derived) can be applied in seemingly all situations (**occlusion**, **move**, **change**, **disappearance**), because purposefully ignoring input events that are possibly interfered with does not depend on the nature of the interference. It however requires to accurately assess the risks of imminent interferences lest GUI elements be too-frequently greyed out, which may appear random in the user's perspective. Typically, applying the *Naive* **Freeze**$_{\text{Block}}$ behavior to every updated element in an interface could prove disturbing if we maintain the greyed-out visual feedback. In the "**disappear**" case, Blocking all risks of interference might mean freezing the entire interface unless a precise scene graph is exposed.

6.3.4 Mapping. The Mapping behavior type, which is the superset of Through from which **HoleOver**$_{\text{Through}}$ was derived, can be seen as the equivalent of Postpone when the interfering GUI change could not be prevented: by sending the interfered-with event to its (estimated) original target, it simulates a situation in which the interface change had not yet occurred, despite what the user perceives. Like Postpone, its principle is not location-dependent and therefore can be applied to seemingly all categories of interferences (**occlusion**, **move**, **change**, **disappearance**)—provided that there exists a history of recent interface states, which is a significantly constraining requirement in today's systems.

Its subset, Through, simplifies that technical constraint by assuming that the interference was location-dependent but also in the foreground of the intended target. While simpler, it almost exclusively apply when the intended click target is **occluded** by another visual element. It can however remain complex to implement, *e.g.* when multi-window apps take the foreground and focus, and the "hole" needs to reach down several layers. In all other cases, **move**, **change**, and **disappearance**, the **Hole** technique cannot be applied as-is but might be extended as "worm-holes" showing the recent past around the cursor to indicate that the event was sent to its initial target.

6.3.5 ***Correct.*** Finally, every behavior type in the ***Correct*** category, *i.e.* Automatic, Prompt and Inform, are independent on the type of interference since they can only be applied after both the interface change and the user event (Figure 2). Their interpretative and technical constraints remain the same regardless of interference type.

Part of our future work will consist in investigating the existence of other behavior types that we could have missed in our design space.

6.4 Overriding the Behaviors

While we focused on minimizing the occurrences of pop-up interferences and mitigating their effects, there exist very specific situations where one may prefer the interference *not* to be handled. This can be the case when the user knows that a certain event will happen at a certain on-screen location, for instance that a notification window will appear, and barrage-click on that area in anticipation.

Our current point of view is that such situations remain exceptional – even compared to interferences themselves – and do not justify the current state of things where GUI toolkits implicitly assume that every user input is fully informed and faithfully reflects the user's intentions. Moreover, it is interesting to note that the proposed Behaviors are not all affected in the same way in this situation. Typically, while **Delay**$_{\text{Postpone}}$ might purely hold the notification spawn until the end of the click barrage, other like **Freeze**$_{\text{Block}}$ might easily cope with these situations since the notification would be activated after the second click. In any case, the negative side-effects of

anti-interference GUI behaviors will need to be thoroughly explored in real, long-term field studies in which participants can experience them realistically. This is part of our future works, using or adapting monitoring tools such as [14, 29].

7 SUMMARY AND CONCLUSION

In this paper we investigate how Graphical User Interfaces, old and new, could minimize the occurrences and effects of interaction interferences. We describe a design space of possible solutions that differ in terms of types of behaviors, *i.e.* whether they ***Prevent***, ***Avoid***, or ***Correct*** the interferences, and in terms of their interpretative and technical requirements. From this space, we designed and instantiated four behaviors for appearing (or *Pop-up*-style) GUI element that we compared to the default window behavior in a controlled experiment with 20 participants. Our results indicate that simple, low-requirement behaviors can already prevent or mitigate most *Pop-up*-style interferences without harming the readability of the appearing element. Participants majoritarily rated the default condition (*i.e.* no mitigation behavior at all) the worst, and their feedback offered valuable hints on how to improve existing behaviors. They also hint at a correlation between user preference and the predictability of the GUI behavior, sometimes even at the cost of best-case outcomes. We then discuss how our design space and behaviors can generalize from appearing GUI elements to all types of interaction interferences.

Our next steps will involve tweaking the feedbacks of our existing behaviors, exploring methods to detect and predict interferences in order to only apply behaviors in "risky" situations, and exploring the technical feasibility of these solutions in real systems and GUI toolkits. In particular, we will propose updated models of common UI patterns and models (MVC, ECS, finite state machines, etc.) that can expose and allow the *Technical requirements* that we list as a dimension of our design space.

REFERENCES

[1] Caroline Appert, Olivier Chapuis, and Emmanuel Pietriga. 2012. Dwell-and-Spring: Undo for Direct Manipulation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin, Texas, USA) *(CHI '12)*. Association for Computing Machinery, New York, NY, USA, 1957–1966. https://doi.org/10.1145/2207676.2208339

[2] Caroline Appert, Olivier Chapuis, Emmanuel Pietriga, and María-Jesús Lobo. 2015. Reciprocal Drag-and-Drop. *ACM Trans. Comput.-Hum. Interact.* 22, 6, Article 29 (sep 2015), 36 pages. https://doi.org/10.1145/2785670

[3] C Baber and NA Stanton. 1997. Rewritable routines in human interaction with public technology. *International Journal of Cognitive Ergonomics* 1, 4 (1997), 237–249.

[4] Jonathan Back, Ann Blandford, and Paul Curzon. 2007. Slip Errors and Cue Salience. In *Proceedings of the 14th European Conference on Cognitive Ergonomics: Invent! Explore!* (London, United Kingdom) *(ECCE '07)*. Association for Computing Machinery, New York, NY, USA, 221–224. https://doi.org/10.1145/1362550.1362595

[5] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (2006), 77–101. https://doi.org/10.1191/1478088706qp063oa

[6] Marc Brysbaert. 2019. How many words do we read per minute? A review and meta-analysis of reading rate. *Journal of Memory and Language* 109 (2019), 104047. https://doi.org/10.1016/j.jml.2019.104047

[7] Linda Di Geronimo, Larissa Braz, Enrico Fregnan, Fabio Palomba, and Alberto Bacchelli. 2020. UI Dark Patterns and Where to Find Them: A Study on Mobile Applications and User Perception. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3313831.3376600

[8] Guillaume Faure, Olivier Chapuis, and Michel Beaudouin-Lafon. 2009. Acquisition of Animated and Pop-Up Targets. In *Human-Computer Interaction – INTERACT 2009*, Tom Gross, Jan Gulliksen, Paula Kotzé, Lars Oestreicher, Philippe Palanque, Raquel Oliveira Prates, and Marco Winckler (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 372–385.

[9] Dan Fitton and Janet C. Read. 2019. Creating a Framework to Support the Critical Consideration of Dark Design Aspects in Free-to-Play Apps. In *Proceedings of the 18th ACM International Conference on Interaction Design and Children* (Boise, ID, USA) *(IDC '19)*. Association for Computing Machinery, New York, NY, USA, 407–418. https://doi.org/10.1145/3311927.3323136

[10] Paul M Fitts. 1954. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of experimental psychology* 47, 6 (1954), 381. https://doi.org/10.1037/h0055392

[11] Colin M. Gray, Yubo Kou, Bryan Battles, Joseph Hoggatt, and Austin L. Toombs. 2018. The Dark (Patterns) Side of UX Design. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3173574.3174108

[12] Colin M. Gray, Cristiana Santos, and Nataliia Bielova. 2023. Towards a Preliminary Ontology of Dark Patterns Knowledge. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI EA '23)*. Association for Computing Machinery, New York, NY, USA, Article 286, 9 pages. https://doi.org/10.1145/3544549.3585676

[13] Erik Hollnagel. 1993. *Human Reliability Analysis: Context and Control*. London: Academic Press.

[14] Dugald Ralph Hutchings, Greg Smith, Brian Meyers, Mary Czerwinski, and George Robertson. 2004. Display Space Usage and Window Management Operation Comparisons between Single Monitor and Multiple Monitor Users. In *Proceedings of the Working Conference on Advanced Visual Interfaces* (Gallipoli, Italy) *(AVI '04)*. Association for Computing Machinery, New York, NY, USA, 32–39. https://doi.org/10.1145/989863.989867

[15] Gordon Logan, Trisha Van Zandt, Frederick Verbruggen, and Eric-Jan Wagenmakers. 2014. On the Ability to Inhibit Thought and Action: General and Special Theories of an Act of Control. *Psychological review* 121 (02 2014), 66–95. https://doi.org/10.1037/a0035230

[16] Aditi M. Bhoot, Mayuri A. Shinde, and Wricha P. Mishra. 2021. Towards the Identification of Dark Patterns: An Analysis Based on End-User Reactions. In *Proceedings of the 11th Indian Conference on Human-Computer Interaction* (Online, India) *(IndiaHCI '20)*. Association for Computing Machinery, New York, NY, USA, 24–33. https://doi.org/10.1145/3429290.3429293

[17] Wendy E. Mackay and Anne-Laure Fayard. 1997. HCI, Natural Science and Design: A Framework for Triangulation across Disciplines. In *Proceedings of the 2nd Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques* (Amsterdam, The Netherlands) *(DIS '97)*. Association for Computing Machinery, New York, NY, USA, 223–234. https://doi.org/10.1145/263552.263612

[18] I. Scott MacKenzie. 1992. Fitts' Law as a Research and Design Tool in Human-Computer Interaction. *Human–Computer Interaction* 7, 1 (1992), 91–139. https://doi.org/10.1207/s15327051hci0701_3 arXiv:https://doi.org/10.1207/s15327051hci0701_3

[19] Arunesh Mathur, Mihir Kshirsagar, and Jonathan Mayer. 2021. What Makes a Dark Pattern... Dark? Design Attributes, Normative Considerations, and Measurement Methods. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 360, 18 pages. https://doi.org/10.1145/3411764.3445610

[20] Giovanni Mirabella, Pierpaolo Pani, Martin Paré, and Stefano Ferraina. 2006. Inhibitory control of reaching movements in humans. *Experimental Brain Research* 174, 2 (Sep 2006), 240–255. https://doi.org/10.1007/s00221-006-0456-0

[21] Mathieu Nancel and Andy Cockburn. 2014. Causality: A Conceptual Model of Interaction History. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. ACM, New York, NY, USA, 1777–1786. https://doi.org/10.1145/2556288.2556990

[22] Donald A Norman. 1988. *The psychology of everyday things*. Basic books.

[23] Jens Rasmussen. 1982. Human errors. A taxonomy for describing human malfunction in industrial installations. *Journal of occupational accidents* 4, 2-4 (1982), 311–333.

[24] James Reason. 1990. *Human error*. Cambridge university press.

[25] Jun Rekimoto. 1997. Pick-and-Drop: A Direct Manipulation Technique for Multiple Computer Environments. In *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology* (Banff, Alberta, Canada) *(UIST '97)*. Association for Computing Machinery, New York, NY, USA, 31–39. https://doi.org/10.1145/263407.263505

[26] Jeff Sauro and James R. Lewis. 2010. Average Task Times in Usability Tests: What to Report?. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. Association for Computing Machinery, New York, NY, USA, 2347–2350. https://doi.org/10.1145/1753326.1753679

[27] Philippe Schmid, Sylvain Malacria, Andy Cockburn, and Mathieu Nancel. 2020. Interaction Interferences: Implications of Last-Instant System State Changes. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20)*. Association for Computing Machinery, New York, NY, USA, 516–528. https://doi.org/10.1145/3379337.3415883

[28] Abigail J. Sellen and Donald A. Norman. 1992. *The Psychology of Slips*. Springer US, Boston, MA, 317–339. https://doi.org/10.1007/978-1-4899-1164-3_13

[29] Susanne Tak and Andy Cockburn. 2009. Window Watcher: A Visualisation Tool for Understanding Windowing Activities. In *Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group: Design: Open 24/7* (Melbourne, Australia) *(OZCHI '09)*. Association for Computing Machinery, New York, NY, USA, 241–248. https://doi.org/10.1145/1738826.1738865

[30] Frederick Verbruggen, Adam R Aron, Guido Ph Band, Christian Beste, Patrick G Bissett, Adam T Brockett, Joshua W Brown, Samuel R Chamberlain, Christopher D Chambers, Hans Colonius, Lorenza S Colzato, Brian D Corneil, James P Coxon, Annie Dupuis, Dawn M Eagle, Hugh Garavan, Ian Greenhouse, Andrew Heathcote, René J Huster, Sara Jahfari, J Leon Kenemans, Inge Leunissen, Chiang-Shan R Li, Gordon D Logan, Dora Matzke, Sharon Morein-Zamir, Aditya Murthy, Martin Paré, Russell A Poldrack, K Richard Ridderinkhof, Trevor W Robbins, Matthew Roesch, Katya Rubia, Russell J Schachar, Jeffrey D Schall, Ann-Kathrin Stock, Nicole C Swann, Katharine N Thakkar, Maurits W van der Molen, Luc Vermeylen, Matthijs Vink, Jan R Wessel, Robert Whelan, Bram B Zandbelt, and C Nico Boehler. 2019. A consensus guide to capturing the ability to inhibit actions and impulsive behaviors in the stop-signal task. *eLife* 8 (Apr 2019). https://doi.org/10.7554/eLife.46323

[31] Christopher D Wickens, JG Hollands, S Banbury, and R Parasuraman. 1992. Attention in perception and display space. *Engineering Psychology and Human Performance (2 nd ed.). New York: HarperCollins* (1992).

AUTHORS STATEMENT

This paper is a direct follow-up to a previous publication of some of the authors:

[27] Philippe Schmid, Sylvain Malacria, Andy Cockburn, and Mathieu Nancel. 2020. Interaction Interferences: Implications of Last-Instant System State Changes. In UIST '20. https://doi.org/10.1145/3379337.3415883

The 2020 paper contributes a first description and categorization of the overall problem of interaction interferences, and quantifies some aspects of their timing. The paper under submission (1) proposes a design space of solutions, grounded in different aspects of feasibility so they can be implemented by interaction designers, and (2) evaluates some of the most feasible of these solutions as of today, resulting in design guidelines and new directions for this little-explored research topic.