

Evaluating a Gradual Detection Meter as a Difficulty Control for User-System Interaction in a Stealth Game

Cristian-Raul Moldovan

Technical University of Cluj-Napoca
Cluj-Napoca, Romania
raulcristian.moldovan@yahoo.com

Dorian Gorgan

Technical University of Cluj-Napoca
Cluj-Napoca, Romania
dorian.gorgan@campus.utcluj.ro

ABSTRACT

In stealth games, player experience depends on how enemies perceive and react to the player. Shadow Protocol uses a gradual detection meter: guards accumulate awareness over time instead of instantly detecting the player, giving visible feedback and a chance to recover. We ask whether the fill rate r of that meter is a usable and predictable difficulty control. Replacing the human player with a deterministic agent, we ran 180 trials over nine fill rates, with the guards' close-range proximity failsafe both enabled and disabled. Where the meter alone governs detection, the measured detection time equals $t_0 + 1/r$ to within four milliseconds across seven fill rates spanning an eightfold range, so the parameter is not merely monotonic but predictable in closed form. Chase time rises from 0 to 25.4 percent of guard time while completion time never moves, and below a predicted critical rate of 0.40 per second the meter cannot fill before the encounter ends, so the player is never detected. The meter is a continuous difficulty parameter with a computable range of usefulness.

AUTHOR KEYWORDS

user-system interaction; game artificial intelligence; stealth games; finite state machine; perception modelling; difficulty tuning; experimental evaluation; Unity

DOI: 10.37789/icusi.2026.30

INTRODUCTION

Computer games are interactive systems in which the user and the system observe and react to each other in a tight loop. Stealth games make this loop explicit: enemy agents perceive the player's actions, the game communicates the agents' growing awareness back to the player, and the player adapts their behaviour in response. The quality of this exchange shapes the entire experience: if guards detect the player too quickly the interaction feels unfair, and if they react too slowly the challenge disappears. Tuning how the system responds to the user is therefore a central concern of stealth game design, and it is closely related to the broader problem of difficulty adjustment in games [1] and to models of player enjoyment such as GameFlow [8].

Shadow Protocol is a top-down 3D stealth game developed in Unity. Its guards combine three artificial intelligence components: a vision cone implemented in FieldOfView, a hearing sensor implemented in AudioSensor, and an eight-state finite state machine implemented in EnemyBrain that drives the states Patrol, Suspicious, Investigate, Chase, Attack, Search, Retreat and Return.

A common design choice in stealth games is whether detection is instantaneous or gradual. Instantaneous detection flips the guard from unaware to alert the moment a clear line of

sight exists. Gradual detection instead accumulates awareness over time, giving the player a short window to break the line of sight before the guard commits. From an interaction point of view the two designs differ in the feedback and the recovery options they offer the user: the gradual variant turns detection from a binary system event into a continuous, observable negotiation between the user's movement and the system's perception. In Shadow Protocol the gradual approach is implemented as a detection meter inside EnemyBrain. While the player is visible, a value $d \in [0, 1]$ rises at a configurable rate $r = \text{detectionFillPerSecond}$, and when the player is not visible it decays at a separate rate. When d crosses the suspicious threshold the guard begins to investigate, and when d reaches the chase threshold the guard locks on and pursues.

This paper evaluates that one technique. The central question is whether the single parameter r acts as a usable and predictable control over the difficulty of the user-system interaction. We state and test the following hypothesis.

H1: The gradual detection meter yields more controllable difficulty than instant line-of-sight detection. A higher fill rate r lowers the time-to-detect and raises the proportion of guard time spent chasing, and these effects are non-linear; at a low fill rate a forgiveness window appears in which a moving player can cross a guard's view without ever being fully detected.

Scope

The study isolates the detection meter. All other perception parameters (view radius, view angle, hearing radius) and all level geometry are held fixed. Audio detection triggers investigation but does not fill the meter, so the hearing channel does not confound the vision-based measurement.

One mechanism does compete with the meter and is therefore treated as a second experimental factor rather than left as a confound. Guards carry a proximity failsafe: a player who comes within melee range is detected immediately, whatever the meter reads. In the shipped game this is desirable, since it stops a guard from being walked into blindly, but it also means the meter is not always what decides an encounter. Every condition is therefore run twice, once with the failsafe active, which is the shipped behaviour, and once with it disabled, which isolates the meter completely. Comparing the two is what separates the parameter's effect from the route's geometry.

RELATED WORK

Detection modelling has a long history in commercial stealth games. Early titles such as the Thief series [4] and the Metal Gear Solid series [3] exposed guard awareness as discrete alert states (unaware, suspicious, alarmed), with transitions triggered the moment a sight or sound test passed. Later

games, including the Splinter Cell [11] and Assassin's Creed [12] series, moved toward a visible, gradually filling awareness indicator that gives the player feedback and a recovery window before full detection. The detection meter studied here belongs to this second family.

On the academic side, game agent decision making is commonly built on finite state machines or behaviour trees. Finite state machines, as used in EnemyBrain, are valued for being transparent and easy to author and debug, at the cost of scaling poorly as the number of states and transitions grows; behaviour trees trade some transparency for better modularity. Millington's survey of game AI [6] treats perception, state-based decision making and steering as separate layers, which mirrors the layered design of the Shadow Protocol guards described in the section on Shadow Protocol. Isla's account of the Halo 2 combat AI [2] describes how staged awareness states are used in a shipped commercial title. Steering and flocking, used here in the squad layer, follow Reynolds' classic boids model of separation, alignment and cohesion [7].

Closer to this paper's method, Tremblay et al. [9, 10] analyse stealth level design by simulating player paths against guard perception models, quantifying risk rather than relying on designer intuition. On the interaction side, Hunicke [1] argues for treating difficulty as an adjustable system parameter, and Sweetser and Wyeth [8] place appropriate challenge at the centre of player enjoyment.

Most of this prior work describes detection systems as design artifacts rather than measuring them. This paper treats one such system as an experimental object instead.

THE GAME: SHADOW PROTOCOL

Shadow Protocol is a top-down 3D stealth game in which the player guides a single infiltrator through guarded interiors to an extraction zone without being caught. The camera looks down on the scene, so the player sees the full room layout but the guards do not: a guard only knows what its own senses report. This split between what the user knows and what each agent knows is the heart of the stealth loop.

Player toolkit

The player moves with a Rigidbody at a fixed speed and is unarmed; survival depends on avoiding sight rather than on fighting. Movement is not silent: the PlayerController exposes a noise radius that scales with motion, up to 7 units when moving and zero when standing still, so a careless sprint can alert a guard that never saw the player. The player also carries a thrown decoy (GrenadeThrow and Grenade) that emits a loud noise event at its landing point, pulling guards toward a false location. Health is finite (100 points); enough melee hits from a guard in the Attack state end the run.

Levels

The game ships three hand-built levels placed by an in-editor LevelBuilder that also bakes the navigation mesh. Level 1, used throughout the experiment, is a two-room layout with a single door-gated guard; the later levels add rooms, doors and guards. Doors are interactive, and a closed door carries a NavMesh obstacle that blocks pathing until it is opened, so geometry and navigation stay consistent for everyone in the scene.

Guard artificial intelligence

The guards are the system side of the user-system interaction and the focus of the game's AI. Each guard stacks several techniques:

- **Perception.** A vision cone (FieldOfView) with a ray-cast line-of-sight check, and a hearing sensor (AudioSensor) that reacts to the player's movement noise and to thrown decoys.
- **Decision making.** An eight-state finite state machine (EnemyBrain) cycling through Patrol, Suspicious, Investigate, Chase, Attack, Search, Retreat and Return, mediated by the gradual detection meter that this paper evaluates. Figure 1 shows the full machine and its transition conditions.
- **Navigation.** NavMesh pathfinding along patrol waypoint sets that change with the guard's state and with which doors are open.
- **Group behaviour.** A squad layer (CaptainBrain, SquadManeuver and BoidsSteering) lets multiple guards coordinate under a captain directing flocking-style movement.

Among these techniques the gradual detection meter is the one chosen for formal evaluation. It sits at the seam between perception and decision making, it is governed by a single continuous parameter, and it shapes the moment-to-moment fairness of every encounter, which makes it both the most measurable and the most design-relevant knob in the stack.

TASKS AND SCENARIOS

The task in every trial is identical and corresponds to the core game objective on Level 1: starting from a fixed spawn point, the player must reach the extraction zone (WinZone) while one guard patrols the room that contains the exit. The level was chosen deliberately as the simplest layout in the game, which minimises confounds from branching routes and multiple guards.

The scenario unfolds as follows. The guard spawns inside the left room and patrols an indoor waypoint set while a door, gated by a NavMesh obstacle, separates that room from the hall. The player must approach, open the door and move to the extraction zone. Because the exit and the guard share the same room, the player is forced through the guard's field of view, which is exactly the condition that exercises the detection meter.

Each trial ends in one of three outcomes:

- **Win:** the player reaches the extraction zone.
- **Caught:** the guard reaches the player and the player health reaches zero.
- **Timeout:** neither event occurs within 120 seconds.

The full experiment is a sweep: nine values of the fill rate in each of the two proximity-failsafe conditions, ten trials per cell, for a total of one hundred and eighty trials, all on the same level with the same starting conditions.

PARTICIPANTS

A note on the heading first: this section fills the Participants slot of the standard evaluation structure, but the role a participant plays in the design, that of the entity performing the task under each condition, is here filled by a program. The heading is kept so that the paper can be read against the usual structure.

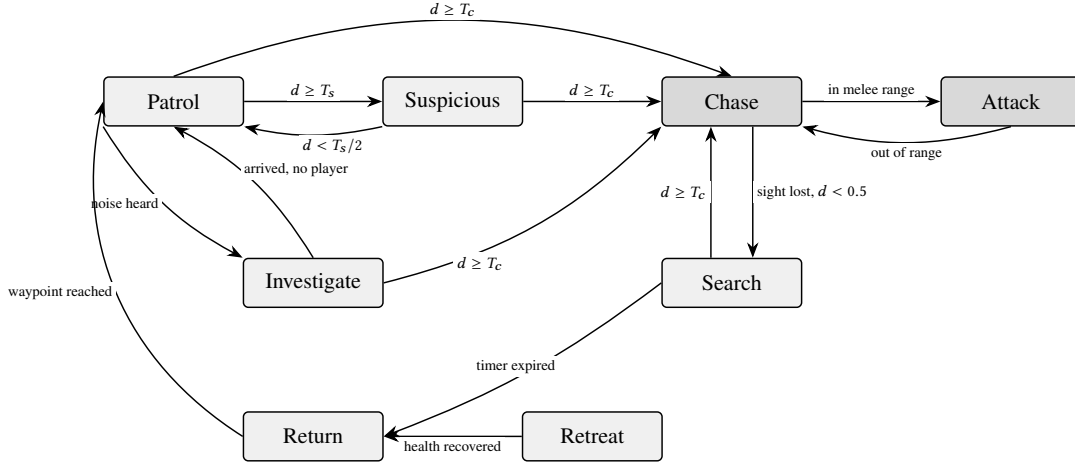


Figure 1: The eight-state guard machine in EnemyBrain with its transition conditions ($T_s = 0.4$, $T_c = 1.0$). Three global rules are omitted for readability: low health forces Retreat, a player inside melee range snaps the meter to T_c and forces Chase, and an active decoy wipes the meter and forces Investigate.

Human participants were deliberately not used. In a stealth game the dominant source of variation between runs is player skill: route choice, timing and reaction speed differ from person to person and from attempt to attempt, and that variation would mask the effect of the single parameter under study.

The participant is therefore a deterministic scripted agent, ScriptedPlayer. It disables the normal PlayerController input, makes the player rigid body kinematic, and drives a Unity NavMeshAgent that paths to the extraction zone along the baked navigation mesh. It opens door-gated exits automatically, since it has no key input, and it emits the same movement noise profile a human would by setting the noise radius from its current velocity. Movement speed is matched to the human controller value of 4.5 units per second. Because the agent follows the same path at the same speed every time, any difference in the measured outcomes can be attributed to the independent variable rather than to player behaviour.

SYSTEM CONFIGURATION

The game runs in Unity [13]. The relevant subsystems and their fixed settings are summarised below.

- **Vision (FieldOfView):** view radius 8 units, view angle 90 degrees, with a raycast line-of-sight check so that walls block vision. These values are constant across all trials.
- **Hearing (AudioSensor):** hearing radius 10 units. A guard hears the player when the player’s current noise radius reaches the guard and the guard is within its own hearing radius. Audio events push the guard toward the Investigate state but do not contribute to the detection meter.
- **Decision making (EnemyBrain):** an eight-state finite state machine. The detection meter $d \in [0, 1]$ fills at rate r while the player is visible and decays at the fixed rate $k = 0.3$ per second otherwise. The suspicious threshold is $T_s = 0.4$ and the chase threshold is $T_c = 1.0$. The melee range that arms the proximity failsafe is 2.5 units. Only the fill rate r is varied.
- **Navigation:** Unity NavMesh, baked at build time. Closed doors carry NavMesh obstacles that carve the

mesh, so a closed door genuinely blocks the agent’s path until opened.

- **Player:** movement speed 4.5 units per second, maximum noise radius 7 units, maximum health 100.

Two custom tools support the experiment. ExperimentLogger is an in-scene component that owns the sweep: it sets the fill rate and the proximity-failsafe flag on every guard once spawned, detects the trial end condition, writes one CSV row per trial, and reloads the scene with the next configuration. The sweep position is stored in PlayerPrefs so it survives the scene reloads. Each row records the timestamp, fill rate, failsafe condition, trial index, outcome, time-to-detect, completion time, guard count and the percentage of guard time spent in each state, so each of the one hundred and eighty trials is recoverable individually. The second tool is the ScriptedPlayer described in the section on participants. Data analysis is performed offline by a Python script (pandas [5] and matplotlib) that groups the rows by fill rate and produces the summary table and figures used in the analysis of the measured data.

Availability. The raw logger output (one CSV row per trial, for every trial reported here) and the analysis script are available from the authors on request, so that every aggregate in this paper can be recomputed from the per-trial rows.

Formal model of the detection meter

The detection meter can be written as a simple piecewise-linear accumulator. Let $v(t) \in \{0, 1\}$ be the visibility flag, equal to 1 when the player is inside the guard’s view cone with a clear line of sight and 0 otherwise. The awareness value $d(t) \in [0, 1]$ evolves as

$$\frac{dd}{dt} = \begin{cases} +r & \text{if } v(t) = 1, \\ -k & \text{if } v(t) = 0, \end{cases} \quad d(t) \in [0, 1], \quad (1)$$

where r is the fill rate (the independent variable) and $k = 0.3$ per second is the fixed decay rate. The guard enters the Investigate behaviour when d crosses the suspicious threshold $T_s = 0.4$ and commits to a chase when d reaches the chase threshold $T_c = 1.0$. One rule sits outside (1): whenever the player is within the melee range of 2.5 units, the guard snaps d to T_c and enters Chase irrespective of r . This proximity failsafe is what bounds the slow end of the sweep, and the

analysis below separates the regime in which the meter decides the encounter from the regime in which the failsafe does.

For the idealised case of continuous visibility from a standing start ($d(0) = 0, v \equiv 1$), the meter fills linearly and the time to reach the chase threshold is

$$t_{\text{detect}} = \frac{T_c}{r} = \frac{1}{r}. \quad (2)$$

This bound is the source of the non-linearity observed in the measured data: as r grows, $1/r$ shrinks fast at first and then flattens. The measured times are larger than $1/r$ because the player is not continuously visible from $t = 0$; it must first travel into the guard's cone, and intermittent occlusion lets the decay term k claw the meter back. The gap between t_{detect} from (2) and the measured mean is exactly the forgiveness the gradual design buys the player, and at low r it can be large enough that d never reaches T_c before the player breaks line of sight, which is the forgiveness window of H1.

VARIABLES

Independent variables

The primary independent variable is the detection meter fill rate $r = \text{detectionFillPerSecond}$ in EnemyBrain. It is swept over nine levels:

$$r \in \{0.3, 0.6, 1.0, 1.5, 2.0, 3.0, 4.0, 6.0, 8.0\}.$$

The largest value, $r = 8.0$, fills the meter from 0 to 1 in roughly an eighth of a second and therefore acts as a near-instant line-of-sight baseline against which the gradual settings are compared. The smallest value, $r = 0.3$, requires several seconds of continuous visibility before a lock-on. The levels are spaced so that the upper half of the range, where the reciprocal $1/r$ flattens, is sampled at 1.5, 3.0, 4.0 and 6.0 rather than jumping from 2.0 straight to the baseline, since a coarse grid there would leave the shape of the curve underdetermined.

The secondary independent variable is the proximity failsafe, with two levels, active and disabled. It is not a parameter a designer would ship as a choice; it is manipulated here to separate detection caused by the meter from detection caused by contact.

Dependent (measurable) variables

- **Time-to-detect** (seconds): the time from trial start to the first instant any guard's meter reaches the chase threshold. Blank when the player is never detected.
- **Detection rate**: the fraction of trials in which the player was detected at all.
- **Capture rate**: the fraction of trials ending in Caught.
- **Win rate** and **timeout rate**: the fractions ending in Win and Timeout.
- **Completion time** (seconds): trial duration.
- **Chase share** (percent): the percentage of total guard time spent in the Chase state, used as a proxy for how much pressure the guard applies.

Chase share needs one caveat stated at the point of definition rather than left for the reader to infer. It is a share of guard time within a trial whose duration is itself a measured variable, so in general a change in the denominator could masquerade as a change in pressure. That risk does not materialise here: completion time turns out to be essentially constant across the sweep at 5.32 seconds for every trial in which the player was

r	TTD (s)	Median (s)	Det. rate	Chase (%)	Compl. (s)
<i>Proximity failsafe on (shipped behaviour)</i>					
0.3	4.013	4.005	1.00	3.67	5.323
0.6	4.005	4.005	1.00	3.61	5.314
1.0	3.865	3.871	1.00	6.32	5.317
1.5	3.531	3.531	1.00	12.46	5.311
2.0	3.364	3.364	1.00	15.53	5.312
3.0	3.190	3.189	1.00	18.85	5.314
4.0	3.116	3.122	1.00	20.29	5.315
6.0	3.057	3.038	1.00	21.41	5.319
8.0	2.986	2.988	1.00	22.61	5.313
<i>Proximity failsafe off (meter isolated)</i>					
0.3	n/a	n/a	0.00	0.00	5.320
0.6	4.505	4.487	1.00	0.08	5.318
1.0	3.819	3.819	1.00	8.96	5.324
1.5	3.486	3.486	1.00	15.33	5.326
2.0	3.322	3.322	1.00	18.36	5.327
3.0	3.152	3.152	1.00	21.54	5.325
4.0	3.069	3.069	1.00	23.08	5.326
6.0	2.988	2.986	1.00	24.61	5.323
8.0	2.946	2.946	1.00	25.41	5.325

Table 1: Per fill-rate summary, ten trials per cell. Time-to-detect and its median are conditional on detection; the detection rate gives the number of trials they are computed over. At $r = 0.3$ with the failsafe off no trial produced a detection, so no time is defined.

detected, so the denominator is fixed in practice and the chase share can be read directly as pressure. Where the denominator does move, at $r = 0.3$, it moves only because two trials ended early without any detection, and those trials are identified explicitly in the results.

Controlled variables

The level, the player path and speed, the view radius and angle, the hearing radius, the decay rate, the melee range and both detection thresholds are all held constant. Ten trials per cell smooth out the small residual non-determinism in NavMesh timing.

ANALYSIS OF MEASURED DATA

Table 1 reports the per-setting aggregates over the one hundred and eighty trials, nine fill rates in each of the two failsafe conditions, ten trials per cell. Time-to-detect is averaged only over trials in which a detection actually occurred, and the detection rate printed beside it says how many trials that was. The subsections below take the dependent variables in turn, one subsection each, so that every variable defined in the previous section is reported explicitly rather than left to be inferred.

Time-to-detect

In the shipped condition, time-to-detect falls from 4.013 seconds at $r = 0.3$ to 2.986 seconds at $r = 8.0$; Figure 2 plots the curve for both conditions. The standard deviations are at most 55 milliseconds and usually under 12, so the scripted player removed essentially all run-to-run noise and the differences are attributable to r alone.

The relationship is non-linear, but the qualification matters more than the headline. The fall is not spread evenly across the sweep, and it is worth being precise about where it happens rather than quoting the endpoints. Between $r = 0.3$ and $r = 1.0$ the mean moves by only 0.148 seconds, while between $r = 1.0$ and $r = 8.0$ it moves by 0.879 seconds, six times as much.

Below $r \approx 1$ the curve is flat because the meter is no longer what ends the encounter, as the next subsection shows. Any statement that time-to-detect ranges from 4.01 to 2.99 seconds across the sweep therefore describes the route as much as the parameter, and the range should not be quoted without saying where the parameter is actually in control.

Measured detection time versus the model bound

The formal model above predicts a detection time of $1/r$ under continuous visibility, so the gap between the measured mean and this bound quantifies how much slack the real encounter adds. Figure 2 plots both conditions against $1/r$ and against the shifted bound $t_0 + 1/r$.

In the isolated condition the fit is essentially exact. The offset $t_{\text{measured}} - 1/r$ takes the values 2.8192, 2.8190, 2.8220, 2.8183, 2.8190, 2.8214 and 2.8208 seconds at $r = 1.0, 1.5, 2.0, 3.0, 4.0, 6.0$ and 8.0. That is a spread of 3.7 milliseconds across seven fill rates spanning an eightfold range, and adding $r = 0.6$ at 2.8386 widens it only to 20 milliseconds. The measured mean is therefore $t_0 + 1/r$ with $t_0 = 2.820$ seconds, where t_0 is the fixed time the scripted route needs before it first enters the guard's view cone with a clear line of sight. Once the player is visible the meter fills linearly and the idealised model of (2) holds without correction. This is a direct empirical validation of the piecewise-linear model across the whole range in which detection occurs at all, and it is the most precisely determined result in the paper.

The same fit holds in the shipped condition for $r \geq 1.0$, with $t_0 = 2.867$ seconds and a spread of 33 milliseconds. The 47 millisecond difference in t_0 between the two conditions is a between-sweep offset rather than an effect of the manipulation; the same drift of a few tens of milliseconds appears between any two sweeps run at different times, and it is discussed under conclusion validity.

Below $r = 1$ the two conditions separate, and the separation is the central result. With the failsafe on, the shifted bound predicts detection at 6.20 and 4.49 seconds for $r = 0.3$ and $r = 0.6$, but the measurements plateau at 4.013 and 4.005 seconds, two values that differ by 8 milliseconds despite a twofold change in r . A plateau this flat is not a slow meter; it is a different mechanism. With the failsafe off, the plateau disappears: $r = 0.6$ moves to 4.505 seconds, within 18 milliseconds of the 4.487 the shifted bound predicts, and $r = 0.3$ produces no detection whatsoever.

That last point is predicted quantitatively rather than merely observed. The encounter lasts until the player reaches the extraction zone at about $T_{\text{end}} = 5.32$ seconds. Detection requires $t_0 + 1/r \leq T_{\text{end}}$, which rearranges to

$$r \geq r^* = \frac{1}{T_{\text{end}} - t_0} = \frac{1}{5.32 - 2.82} = 0.40 \text{ per second.} \quad (3)$$

At $r = 0.3$ the meter needs 6.20 seconds inside an encounter lasting 5.32, so it can never fire, and across ten trials it never did. At $r = 0.6$ it needs 4.49 seconds, so it can, and across ten trials it always did. The model predicts the boundary and both sides of it behave as predicted.

Detection rate and the forgiveness window

With the failsafe on, the detection rate is 1.00 at every fill rate: on this route a player who walks past the guard is always caught by something. With the failsafe off it is 1.00 everywhere except at $r = 0.3$, where it is 0.00 across all ten trials.

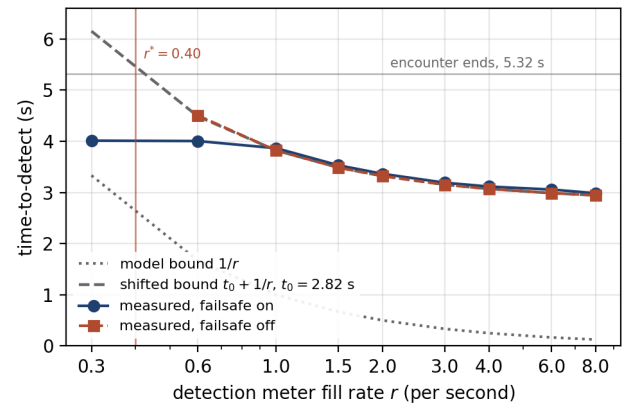


Figure 2: Measured mean time-to-detect against the model bound $1/r$ and the shifted bound $t_0 + 1/r$ with $t_0 = 2.82$ s. With the failsafe off the measurements sit on the shifted bound wherever detection is possible at all. The vertical line marks $r^* = 0.40$ from (3), below which the meter cannot fill before the encounter ends.

The guard spent 45.4 percent of those trials patrolling and 54.6 percent investigating the noise it heard, and never once entered Chase or Attack.

This is the forgiveness window of H1, and the two conditions together say precisely what it is. It is not a probabilistic near-miss at the slow end of the meter. It is a deterministic threshold at $r^* = 0.40$ below which a moving, goal-directed player cannot be detected by the meter at all on this route, and above which they always are. What the shipped configuration adds is the failsafe, which converts that window into a guaranteed detection at 4.01 seconds by contact rather than by sight. A designer choosing $r < r^*$ is therefore not making guards slightly more forgiving; they are turning vision-based detection off on routes of this length and leaving contact as the only remaining mechanism.

Capture rate

The capture rate is 0.00 in all one hundred and eighty trials, in both conditions and at every fill rate. No trial ended with the player's health reduced to zero. This is a direct consequence of the participant design rather than a property of the meter: the scripted player is a committed, goal-directed mover that always reaches the exit before a guard can land enough melee hits to kill it. Raising r makes the guard reach the player sooner and stay on them longer, but the encounter is still over in about 5.32 seconds, which is not long enough for 100 health at 20 damage per hit and a one second cooldown to be exhausted. The meaningful result is therefore that the fill rate tunes detection pressure and not the binary outcome, for a player who never hesitates.

Win rate and timeout rate

The win rate is correspondingly 1.00 in all one hundred and eighty trials and the timeout rate is 0.00; no trial approached the 120 second limit. The flat win rate is what makes the detection rate and the chase share the informative variables here: outcome is saturated, so all of the effect of r has to be read from how the encounter unfolds rather than from how it ends.

Completion time

Completion time is insensitive to everything the experiment varies. Over all one hundred and eighty trials it stays between 5.257 and 5.382 seconds, with per-cell means between 5.311 and 5.327. It does not move with r , it does not move with the failsafe condition, and, at $r = 0.3$ with the failsafe off, it does not move with whether the player was detected at all: those ten undetected trials averaged 5.320 seconds against 5.324 for the detected trials at $r = 1.0$.

Being detected therefore costs the committed agent essentially nothing in transit time. A chasing and attacking guard damages the player and crowds the route, but it does not delay a kinematic agent pathing to the exit. This sharpens the interpretation of the fill rate considerably: for a committed, goal-directed player it scales the pressure of the encounter, when the chase starts and how long it lasts, without changing the outcome, the duration of the task, or the pacing of the level. The interaction becomes more aggressive, not longer or harder to complete. It also justifies reading the chase share as pressure, since the denominator it is normalised by is fixed in practice.

Chase share

The chase share grows monotonically with r in both conditions. With the failsafe on it rises from 3.67 percent at $r = 0.3$ to 22.61 percent at $r = 8.0$, a factor of six. With it off the range is wider still, from 0.00 percent at $r = 0.3$, where no chase ever begins, to 25.41 percent at $r = 8.0$. A faster meter means the guard reaches and stays in the Chase state sooner and longer, so the same fixed encounter feels more aggressive. This is the practical sense in which the fill rate behaves as a difficulty knob: it continuously scales the pressure the system applies to the user without any change to geometry or to the number of guards.

Full FSM state distribution

The logger records the share of guard time spent in every state, not just Chase. Table 2 gives the full breakdown for the isolated condition, where the meter alone drives the machine. Four of the eight states recorded zero time in all one hundred and eighty trials and are omitted.

Two patterns stand out. First, the Patrol share does not respond to r at all: it sits between 44.5 and 45.6 percent at every fill rate in both conditions. The guard leaves its patrol route at a time set by the route and the level, not by how fast its meter fills. Second, the Investigate share falls monotonically as r rises, from 54.6 percent at $r = 0.3$ to 10.8 percent at $r = 8.0$, and the Chase share rises to meet it: at high fill rates the meter races past the suspicious threshold and the guard goes more or less straight to Chase. The fill rate does not simply speed up detection, then; it reallocates the guard's time from cautious investigation to direct pursuit, holding the patrol budget fixed. Suspicious stays at zero throughout because noise reaches the guard before the meter crosses T_s on this route, so the machine enters Investigate through the audio channel first, and Search, Retreat and Return are never exercised by a single committed encounter against an undamaged guard.

The Attack share is constant at 19.3 percent for every $r \geq 1.0$: once the guard reaches the player, the melee exchange lasts about the same time regardless of how quickly it began. The exception is instructive. At $r = 0.6$ the Attack share is 15.3 percent while the Chase share is only 0.08 percent, because

r	Patrol	Investigate	Chase	Attack
0.3	45.38	54.62	0.00	0.00
0.6	44.73	39.92	0.08	15.27
1.0	44.53	27.15	8.96	19.35
1.5	44.50	20.88	15.33	19.29
2.0	44.52	17.79	18.36	19.33
3.0	44.49	14.64	21.54	19.33
4.0	44.49	13.08	23.08	19.35
6.0	44.53	11.52	24.61	19.34
8.0	44.51	10.76	25.41	19.32

Table 2: Guard time per FSM state (percent, mean over ten trials) with the proximity failsafe off, so that the meter alone drives the machine. The Suspicious, Search, Retreat and Return states were zero in every trial and are omitted.

detection arrives at 4.505 seconds into an encounter that ends at 5.32. The guard locks on with less than a second to spare and spends what is left in Attack. Near r^* the meter therefore does not produce a shorter chase; it produces an encounter that skips the chase and arrives at contact.

DESIGN IMPLICATIONS

The measurements say something usable about interface elements that sit on top of a gradual detection meter. This section names the widgets that benefit and states how the results change the way each should be built.

The awareness indicator. The widget that depends most directly on the meter is the awareness indicator itself: the arc, pip ring or directional wedge that renders d back to the player while a guard is filling. Instant detection cannot support such a widget at all, because there is no intermediate value to show. The gradual meter can, but the results put a number on how much it can show. Once the player is inside the cone the warning lasts $1/r$ seconds, and that collapses fast: 3.33 seconds at $r = 0.3$, 1.00 second at $r = 1.0$, 0.50 at $r = 2.0$ and 0.125 at $r = 8.0$. Beyond roughly $r = 2$ the indicator degenerates into a binary alarm that flips faster than a player can act on it, and the feedback contract the gradual design was chosen for is lost even though the meter is still technically gradual. A designer who wants the indicator to be actionable rather than decorative should treat $r \approx 2$ as an upper bound, not a midpoint, of the usable range.

The difficulty slider. The natural way to expose this parameter in an options menu is a slider interpolating r linearly between a low and a high value, and the measurements show that this is the wrong mapping. Detection time follows $t_0 + 1/r$, so difficulty in the units the player experiences, seconds of grace, is linear in $1/r$ and not in r . A slider linear in r spends most of its travel where nothing perceptible changes: moving from $r = 2.0$ to $r = 8.0$ is four sixths of such a slider and buys only 0.375 seconds, while the first fifth covers more than two seconds. The fix is to expose the slider in units of time and derive the parameter, $r = 1/T$, where T is the grace window in seconds, so that each notch changes the reaction window by the same amount.

Enemy archetypes. Because the parameter is a single float on each guard rather than a property of the level, distinct enemy types can be authored by varying it alone: a distracted rookie at $r = 0.5$, a regular patrol at $r = 1.0$, an alert veteran at $r = 2.0$ and a static camera at $r = 8.0$, all sharing one finite state machine, one vision cone and one set of thresholds. The state distribution supports this reading directly. A guard

at $r = 1.0$ spends 27 percent of its time investigating and 9 percent chasing, while a guard at $r = 8.0$ inverts that to 11 and 25 percent, so the archetypes differ in visible behaviour and not only in a hidden number.

Dynamic difficulty adjustment. The completion time result makes the fill rate unusually well behaved as a target for automatic adjustment. Completion time did not move with r anywhere in the sweep, nor with whether the player was detected at all, so a system that raises or lowers r between encounters changes how much pressure the player is under without changing how long the level takes. Pressure and pacing are therefore separable, which is not true of the other obvious knobs: adding guards or enlarging patrol routes moves both at once. A dynamic difficulty system in the sense of Hunnicke [1] can act on r and leave the level's rhythm intact.

Accessibility presets, and the floor beneath them. A low fill rate is an attractive assist mode because it weakens detection without removing the stealth loop: guards, patrols, cones and consequences all remain, and a player who keeps moving gets a real chance to slip past. The results also give the preset a hard floor, which is the more useful half of the finding. Equation (3) says detection by sight is possible only when $r \geq 1/(T_{\text{end}} - t_0)$, where $T_{\text{end}} - t_0$ is how long the player remains exposed. Set r below that and vision-based detection does not become gentle, it stops working: no trial at $r = 0.3$ was ever detected by the meter. An assist preset should therefore be derived from the exposure time of the levels it applies to rather than picked as a small-looking number, and a designer who wants forgiveness rather than blindness should sit just above r^* .

One caveat for all of them. Every widget above assumes the meter is what decides the encounter, and the results show a regime where it is not. Inside the melee range of 2.5 units the failsafe snaps d to T_c irrespective of r , so an indicator that renders d faithfully will jump from partial to full with no intervening animation and appear to glitch at close range. The honest options are to give the snap its own presentation, or to accept that the feedback contract holds at distance and not in contact.

THREATS TO VALIDITY

Internal validity. The largest threat is also the experiment's greatest strength. By replacing the human with a deterministic agent we removed skill noise, but the agent's single fixed route means the result describes one path through one encounter, not the space of player behaviour. The near-zero standard deviations confirm the route is repeatable, not that it is representative.

The one confound that could have been mistaken for a property of the meter, the proximity failsafe, is measured rather than argued away: the route passes within melee range, and removing the failsafe recovers the model's prediction at $r = 0.6$ to within 18 milliseconds, which is what licenses the claim that the plateau below $r = 1$ belongs to the geometry. Two caveats remain. Disabling the failsafe is a different manipulation from routing the player around the guard, and only the latter would show what a distant observer experiences at low r . And r^* in (3) depends on the exposure time of this route, so its value transfers to other levels only through $T_{\text{end}} - t_0$, not as the constant 0.40.

Construct validity. We use Chase share and time-to-detect as proxies for difficulty. They capture detection pressure well but

not the full felt difficulty, which also depends on player stress and on whether detection actually leads to a loss. The flat zero capture rate shows this gap directly: more pressure did not translate into more losses for a player that never hesitates.

External validity. All trials ran on Level 1 with a single guard. Levels with more guards, the squad layer active, or different geometry could interact with the fill rate in ways this study does not observe. The formal model of (2) is level-independent, but the measured offsets from it are not.

Conclusion validity. Ten trials per cell are enough here only because the deterministic agent makes variance tiny. Within a cell the standard deviation of time-to-detect is typically a few milliseconds. The same trial count with human players would be far too small to support the same claims.

Determinism holds within a sweep but not perfectly across sweeps. Runs of the same configuration executed at different times differ by a few tens of milliseconds, which is why t_0 is 2.820 seconds in one condition and 2.867 in the other. The offset is systematic across all fill rates within each sweep, so it behaves like a per-sweep constant rather than noise, and it is an order of magnitude smaller than any effect the paper claims. All comparisons in this paper are therefore made within a sweep, and figures quoted to the millisecond should be read as exact for the run reported and reproducible to about 50 milliseconds across runs.

For the same reason all trials reported here come from a single re-run of the experiment, carried out at a finer resolution in r and with the proximity-failsafe condition added. These sweeps supersede an earlier and coarser five-rate sweep of the same configuration; the two datasets are not pooled, and every figure quoted in this paper comes from the one hundred and eighty trials described above.

CONCLUSION

We evaluated the gradual detection meter used by the guards in Shadow Protocol with a controlled one hundred and eighty trial sweep driven by a deterministic scripted player, at nine fill rates with the guards' proximity failsafe both active and disabled.

The data support H1 with one qualification and one sharpening. Increasing the fill rate lowers time-to-detect and raises the chase share of guard time, from 3.7 to 22.6 percent with the failsafe active and from 0 to 25.4 percent without it. The qualification is that almost all of the movement in time-to-detect lies above $r = 1$, and the flat region below it is produced by the route and the failsafe rather than by the meter, so the full range should not be quoted as an effect of the parameter.

The sharpening is that the meter is predictable in closed form and not merely monotonic. Where it alone governs detection the measured time equals $t_0 + 1/r$ with $t_0 = 2.820$ seconds, the residual varying by under four milliseconds across seven fill rates spanning an eightfold range. The same expression locates where the mechanism stops working: detection by sight requires $r \geq r^* = 1/(T_{\text{end}} - t_0)$, here 0.40 per second, and across ten isolated trials the meter never fired at $r = 0.3$ and always fired at $r = 0.6$. The forgiveness window is thus a threshold rather than a probabilistic softening, and its position is computable from the exposure time of the level. Completion time moved by less than 20 milliseconds across all trials whether or not the player was detected, so the parameter scales pressure independently of pacing.

The main limitation is the source of the experiment's cleanliness: the deterministic, always-successful player produced a flat win rate and zero captures, so the study measures detection pressure and not felt difficulty. The natural next step is a human study, and these results make one cheap to design. Rather than sweeping the parameter blindly it can place conditions where the data says something changes, on either side of $r^* = 0.40$ and around $r \approx 2$ where the awareness indicator stops giving a reaction window a person can use; any difference in perceived difficulty is then attributable to perception, because the system behaviour at each point is already characterised to the millisecond. Two further extensions are worth naming: a route that keeps the player beyond melee range throughout, and applying the same harness to view angle, view radius and hearing radius.

ACKNOWLEDGMENTS

The research presented in this article has been funded by the national project 135PHE / 2026, through the PN IV - European and International Cooperation Programe.

References

- [1] Robin Hunicke. 2005. The Case for Dynamic Difficulty Adjustment in Games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE '05)*. ACM, New York, NY, 429–433.
- [2] Damian Isla. 2005. Handling Complexity in the Halo 2 AI. *Game Developers Conference (GDC)*.
- [3] Konami Computer Entertainment Japan. 1998. Metal Gear Solid. Video game, published by Konami. PlayStation.
- [4] Looking Glass Studios. 1998. Thief: The Dark Project. Video game, published by Eidos Interactive. Windows.
- [5] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*. 56–61.
- [6] Ian Millington. 2019. *AI for Games* (3rd ed.). CRC Press, Boca Raton, FL.
- [7] Craig W. Reynolds. 1987. Flocks, Herds and Schools: A Distributed Behavioral Model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*. ACM, New York, NY, 25–34.
- [8] Penelope Sweetser and Peta Wyeth. 2005. GameFlow: A Model for Evaluating Player Enjoyment in Games. *Computers in Entertainment* 3, 3 (2005).
- [9] Jonathan Tremblay, Pedro Andrade Torres, Nir Rikovitch, and Clark Verbrugge. 2013. An Exploration Tool for Predicting Stealthy Behaviour. In *Proceedings of the 2013 AIIDE Workshop on Artificial Intelligence in the Game Design Process (IDP 2013)*. AAAI Press.
- [10] Jonathan Tremblay, Pedro Andrade Torres, and Clark Verbrugge. 2014. Measuring Risk in Stealth Games. In *Proceedings of the 9th International Conference on the Foundations of Digital Games (FDG 2014)*.
- [11] Ubisoft Montreal. 2002. Tom Clancy's Splinter Cell. Video game, published by Ubisoft. Xbox.
- [12] Ubisoft Montreal. 2007. Assassin's Creed. Video game, published by Ubisoft. PlayStation 3, Xbox 360, Windows.
- [13] Unity Technologies. 2026. Unity User Manual: NavMesh and NavMeshAgent. Accessed July 2026. <https://docs.unity3d.com/Manual/nav-Introduction.html>

[1] Robin Hunicke. 2005. The Case for Dynamic Difficulty Adjustment in Games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in*