Pseudo-visibility: A Game Mechanic Involving Willful Ignorance

Samuel Allen Alexander
The U.S. Securities and Exchange Commission

Arthur Paul Pedersen
The City College of New York

Abstract

We present a game mechanic called *pseudo-visibility* for games inhabited by non-player characters (NPCs) driven by reinforcement learning (RL). NPCs are incentivized to pretend they cannot see pseudo-visible players: the training environment simulates an NPC to determine how the NPC would act if the pseudo-visible player were invisible, and penalizes the NPC for acting differently. NPCs are thereby trained to selectively ignore pseudo-visible players, except when they judge that the reaction penalty is an acceptable tradeoff (e.g., a guard might accept the penalty in order to protect a treasure because losing the treasure would hurt even more). We describe an RL agent transformation which allows RL agents that would not otherwise do so to perform some limited self-reflection to learn the training environments in question.

Introduction

From Monte Carlo Tree Search to AlphaStar, there has been much progress training new types of agents by means of letting the agent simulate the environment. In this paper, we go the other direction. We allow the environment to simulate the agent. This gives rise to new types of environments. It would be hard to apply this strategy directly to human-playable environments, as it is hard to simulate a human. But the strategy can be used indirectly to endow human-playable environments with novel gameplay mechanics. Namely: a human-playable environment can be populated with Non-Player Characters (NPCs) pre-trained in apparently not-human-playable agent-simulating environments.

We will illustrate the above technique with a novel game mechanic we call *pseudo-visibility*. NPCs can see pseudo-visible players, but their pre-training incentivizes them to ignore those players. This is different than true invisibility, because the NPC can weigh said incentives against competing incentives. For example, if a pseudo-visible player gets too close to a guarded treasure, the guard (who was previously ignoring the player) might strategically decide to react to the player, calculating that the incentive to protect the treasure outweighs the incentive to ignore the player. Using reinforcement learning (RL), these incentives can be numerically fine-tuned. All else equal, the bigger the training penalties are for reacting to pseudo-visible players, the more pseudo-visibility resembles invisibility.

The idea of an agent-simulating environment can be traced back to Newcomb's Problem (Nozick 1969). Interest in the idea has picked up recently. Bell et al (2021) investigated so-called *Newcomblike decision processes*, environments which can, at each step in the agent-environment interaction, query the agent's policy function as of that step. Alexander et al (2022) investigated so-called *extended environments*, which can query how the agent would act in response to arbitrary training (not just the training it has actually received). The logic of ignorance in general has also seen recent interest, see (Aldini, Graziani, and Tagliaferri 2020). Our contribution is to point out how these technologies can be used to create exotic game mechanics (describing one particular such mechanic). The high complexity of such mechanics might make them somewhat impractical (Johansson, Eladhari, and Verhagen 2012), but it is interesting that they are theoretically possible.

The paper proceeds as follows. We work in deterministic RL for simplicity. After formally defining pseudo-visibility, we give examples of pseudo-visibility in games and draw attention to its features in game-play. Current state-of-the-art RL agents are not designed to base actions on their own hypothetical behavior. We accordingly offer (in Definition 10) a method for equipping RL agents with some limited self-reflection to help them learn these environments.

Pseudo-visibility

For the sake of simpler definitions, we will assume deterministic RL: policies are deterministic, environments' observation-reward calculations are deterministic, and training is deterministic.

We are interested in games where NPCs are driven by RL policies which are pre-trained before any actual player plays the game. During pre-training, the "player" is another character (whose actions could be, for example, random, trained, or even selected by the game's designer in real-time). Thus, in the following definitions, we will speak of "an NPC X who co-inhabits a larger game together with another character Y," with the unspoken understanding that in the final game (when all the pre-training is done), Y will be played by the game's player. But in the definitions, we avoid referring to Y as *player* because we want to avoid confusing the

player (who drives Y) with the *agent* (who drives the NPC X). So, in the following definitions, the character whom we are intending to potentially become pseudo-visible is Y.

Definition 1. *Suppose μ is an RL environment intended to train an NPC X to co-inhabit a larger game together with another character Y. We say that μ **paints** Y **last** if μ computes its tth observation O_t as $O_t = P(S_t, D(S_t))$, where:*

1. *S_t is an **internal state** (including information about Y) which μ maintains.*
2. *D is a function which paints the world (with state S_t) as it would be seen through X's eyes, except that D does not paint the character Y.*
3. *P is a function which takes the image from D and adds the character Y to it (or leaves it unchanged if, e.g., Y is outside of X's field-of-view).*

In the following definition, we assume the game in question has a notion of whether or not the player is pseudo-visible. For example, the player might become temporarily pseudo-visible after drinking a pseudo-visibility potion or casting a pseudo-visibility spell. The definition does not depend on the specific in-game definition of what it means for the player to be pseudo-visible, only that the game's state S_t includes a Boolean variable pv_t for the player's pseudo-visibility status.

Definition 2. *Let μ be an RL environment intended to train an NPC X to co-inhabit a larger game together with another character Y. Assume μ paints Y last, and let P, D, S_t be as in Definition 1. Assume that the internal state S_t includes a Boolean variable pv_t (intended to represent whether or not Y is pseudo-visible). For each rational number $q \in \mathbb{Q}$, the **result of taking Y's pseudo-visibility into account in μ with penalty** q, denoted $\mu_{Y,q}$, is defined as follows.*

Whenever $\mu_{Y,q}$ needs to compute an observation and reward in response to the agent taking action A_t in response to observation $O_{t-1} = P(S_{t-1}, D(S_{t-1}))$, $\mu_{Y,q}$ computes the observation and reward exactly as μ does, except that:

- *If pv_t = True, then $\mu_{Y,q}$ queries the agent's policy to determine which action A'_t the agent would have taken in response to $O'_{t-1} = D(S_{t-1})$ instead of $O_{t-1} = P(S_{t-1}, D(S_{t-1}))$. If $A'_t \neq A_t$, then $\mu_{Y,q}$ subtracts q from the reward μ would output.*

Informally, in Definition 2, $\mu_{Y,q}$ is the environment which is identical to μ except that the agent is incentivized to ignore Y (whom we should think of as the player in a larger game) any time Y is pseudo-visible. Specifically, if the agent does not ignore Y (i.e., if the agent acts differently than it would act if Y were not painted at all), then penalty q is applied to whatever reward the agent would otherwise receive for the action in question.

Definition 3. *Let G be a single-player game in which a player interacts with various NPCs, and let $q \in \mathbb{Q}$ be a rational-number **penalty**. Define a new game G'_q, the **result of taking pseudo-visibility into account in G with penalty** q, to be the game identical to G except that for every NPC X in G pre-trained in environment μ as in Definition 2 (with the player's character as Y), in G'_q, X is instead pre-trained in $\mu_{Y,q}$.*

The following lemma is trivial, but we state it because it (and its proof) illustrate some subtle nuances.

Lemma 4. *Let G be as in Definition 3. For each NPC x in G, assume that during x's pre-training, the policy function being trained is side-effect-free (i.e., calling said policy function is read-only). Then $G'_0 = G$.*

Proof. Since we are assuming deterministic training, it suffices to show that x's policy is trained the same for G'_0 as for G. The way the policy is trained is based on the reward-observation-action sequence when the agent training the policy interacts with the training environment. Let μ be x's training environment in the creation of G. So $\mu_{Y,0}$ is x's training environment in the creation of G'_0. We argue by induction that the reward-observation-action sequence is the same when the policy is trained in μ as in $\mu_{Y,0}$. For the base case, the initial observation cannot differ in μ and $\mu_{Y,0}$ because no actions have taken place yet and μ and $\mu_{Y,0}$ only differ in how they reward actions. The tth action A_t is the same in both sequences because, by induction, the policy has been trained identically up until A_t is chosen, and we are assuming deterministic training (the policy may have been *called* additional times in $\mu_{Y,0}$, but the hypothesis is that such additional calls have no side effects). And for $t > 0$, the tth reward and observation R_t and O_t are the same in both sequences because they are deterministically computed based on preceding sequences which (by induction) are the same, and, although the action policy might be called an additional time in $\mu_{Y,0}$, that call has no side effects, and the penalty resulting from that extra call (if the agent fails to ignore a pseudo-visible Y) is 0. □

As a particularly treacherous example where Lemma 4 would fail, imagine that the action policy being trained chooses actions by invoking a deterministic pseudo-random number generator (pseudo-RNG) and thus, implicitly, changes the internal state of that pseudo-RNG. Then a subsequent call to the same action policy might generate a different action, even if the policy is not further trained in between calls. This would mean that when $\mu_{Y,q}$ queries the policy in order to compare the most recent action A_t with the hypothetical action A'_t which X would have taken if Y were invisible, the resulting A'_t might not reliably capture which hypothetical action X would have taken if Y were invisible, because by the time $\mu_{Y,q}$ calculates A'_t, X has already used the policy to calculate A_t, which could have polluted the pseudo-RNG. Thus if pseudo-randomness is desired in NPCs in G'_q, care should be taken to avoid such side-effects: for example, give each policy its own isolated pseudo-RNG and only update the pseudo-RNG's internal state when the policy is *trained*, not when the policy is *called*.

Behavior of NPCs in a game with pseudo-visibility

Pseudo-visibility differs from true invisibility. If the player were truly invisible, the NPCs would not see the player, and could not react to seeing the player whether they wanted to or not. If the player is merely pseudo-visible, then NPCs see the player but are incentivized to act as if they don't, however, they may still choose to react to seeing the player be-

cause other incentives outweigh the incentive to ignore the player. By choosing different values for q in Definition 3 we could calibrate these tradeoffs. Large values of q would mean that NPCs would be incentivized to react to the player only in dire circumstances; small positive values of q might only incentivize NPCs to ignore players whom they deem harmless. Here are a few examples.

Example 5 (Guarded treasures)**.** *A guard protects a treasure that the player might steal. Pseudo-visibility would incentivize the guard not to react to the player unless the player seems likely to steal the treasure. Depending how large the penalties are for losing the treasure vs. reacting to the player, the guard might continue ignoring the player until the player gets quite near.*

Example 6 (Amusing erratic behavior)**.** *A street-racer might deliberately swerve all over the road at all times whether the player is nearby or not. By doing so, the racer might be able to cut off a pseudo-visible player without suffering any reaction penalty. The extent to which the racer is incentivized to act so erratically depends on the relative penalties for reacting to the pseudo-visible player, getting passed by the player, or (e.g.) crashing into lamp-posts.*

Example 7 (Emergent behavior)**.** *Multiple NPC soldiers might organically develop a signalling system whereby one soldier can point at the pseudo-visible player so the other soldiers can open fire. The pointing soldier suffers a reaction penalty but the other soldiers avoid it: "I didn't see the player, I only opened fire because I saw my friend pointing at an empty patch of dirt!"*

Example 8 (Disguises)**.** *A clever player (or even an NPC) attempts to cheat by disguising as a pseudo-visible player without actually being one, hoping to fool some NPCs.*

Limitations of current RL agents

If an RL agent were to self-reflect, asking itself questions like "Which action would I have taken if the pseudo-visible player had been invisible?", then the RL agent would presumably eventually be able to figure out patterns like: "If the player is pseudo-visible, then the environment seems to penalize me for acting differently than I would act if the player were invisible." As far as we know, none of the current state-of-the-art RL agents are designed to self-reflect in such a fashion. Thus, we suspect that those agents would have trouble figuring out this aspect of training-environments $\mu_{Y,q}$. In this section, we describe a way to modify a given RL agent to force it to self-reflect in exactly the way necessary to understand the penalties for reacting to the pseudo-visible player. Basically, given an RL agent π, the construction is quite simple: let π' be the agent obtained by intercepting observations sent to π and annotating them with the action π would take if the pseudo-visible player were invisible (or dummy actions if the observation has no pseudo-visible player). Thus in a sense, π' acts as π would act if π were self-reflecting.

For the purpose of formally stating the construction, we will work in a concrete version of RL in which agents are instances of AgentClasses, python classes implementing certain required methods. This will allow us to give the construction using actual python code, which will make it far more readable than if we would attempt to give it using abstract mathematics.

Definition 9. *An **AgentClass** is a python class which implements the following methods:*

- ***act**: takes an observation and returns an action.*
- ***train**: takes an observation* o_prev*, an action* a*, a reward* r*, and an observation* o_next*, and returns nothing. The intuition is that when this method is called, the agent should update its action policy (neural network weights, Q table, etc.) in response to receiving reward r and new observation* o_next *for having taken action a in response to old observation* o_prev.

Definition 10. *Let μ be an environment as in Definition 2. Assume -1 is not a valid action in μ. Suppose:*

1. *There is a function* detect_pseudovis *which takes a μ-given observation* obs *and outputs* True *if* obs *contains a pseudo-visible player,* False *otherwise.*
2. *There is a function* erase_player *which erases the character Y from any given observation containing a pseudo-visible player, by which we mean*

$$\text{erase\_player}(P(S_t, D(S_t))) = D(S_t)$$

for every μ-output observation $P(S_t, D(S_t))$ containing a pseudo-visible player.

For any AgentClass A, we define the AgentClass $\text{SelfRefl}(A)$*, the **version of** A **which self-reflects about pseudo-visible players using*** detect_pseudovis ***and*** erase_player*, to be the following AgentClass:*

```
class SelfRefl_of_A:
  def __init__(self):
    self.A_inst = A() # Store A-instance

  def annotate_obs(self, obs):
    # Annotate obs w/action A would
    # take if player were erased (or -1
    # if no pseudovis player in obs)
    if detect_pseudovis(obs):
      erased = erase_player(obs)
      erased = self.annotate_obs(erased)
      a = self.A_inst.act(erased)
      return (obs, a)
    else:
      return (obs, -1)

  def act(self, obs):
    # Act as A would on annotated obs
    annotated = self.annotate_obs(obs)
    return self.A_inst.act(annotated)

  def train(self,o_prev,a,r,o_next):
    # Train A w/annotated observations
    o_prev = self.annotate_obs(o_prev)
    o_next = self.annotate_obs(o_next)
    self.A_inst.train(o_prev,a,r,o_next)
```

Lemma 11. $\text{SelfRefl}(A)$ *really is an agent.*

Proof. The methods of $\mathrm{SelfRefl}(A)$ clearly have the correct signatures, so the only way $\mathrm{SelfRefl}(A)$ would fail to be an agent would be if it would get stuck in an infinite loop when it's not supposed to. The only place an infinite loop could be caused would be when $\mathrm{annotate\_obs}$ recursively calls itself. But $\mathrm{annotate\_obs}$ only calls itself if the observation in question satisfies $\mathrm{detect\_pseudovis}(\mathrm{obs})$, and when it calls itself, it calls itself on $\mathrm{erased} = \mathrm{erase\_player}(\mathrm{obs})$. Since $\mathrm{erase\_player}$ erases the pseudo-visible character Y from obs, the inner call to $\mathrm{annotate\_obs}$ does not recursively self-call again, so infinite looping is avoided. □

Equivalently, the annotations in Definition 10 could be built into the environment rather than into the agent:

Proposition 12. *Suppose μ, A, $\mathrm{detect\_pseudovis}$ and $\mathrm{SelfRefl}(A)$ are as in Definition 10, $q \in \mathbb{Q}$. Let μ' be the environment which is just like $\mu_{Y,q}$ (Definition 2) except that whenever μ would output observation $O_t = P(S_t, D(S_t))$, μ' instead outputs observation:*

- *(O_t, a), where a is the action the agent would take on observation $(D(s_t), -1)$, if $\mathrm{detect\_pseudovis}(O_t) = \mathrm{True}$, or*
- *$(O_t, -1)$ otherwise.*

Then: the actions and rewards which arise when an A-instance interacts with μ', are the same as the actions and rewards which arise when a $\mathrm{SelfRefl}(A)$-instance interacts with $\mu_{Y,q}$.

Proof. Let $O_1, R_1, A_1, \ldots$ be the observations, rewards, and actions that arise when an A-instance interacts with μ', and let $O^1, R^1, A^1, \ldots$ be the observations, rewards, and actions that arise when a $\mathrm{SelfRefl}(A)$-instance interacts with $\mu_{Y,k}$. A simple inductive argument shows that each $O_n = (O^t, a)$ or $(O^t, -1)$ (as in the statement of the proposition), each $R_n = R^n$, and each $A_n = A^n$. □

Proposition 12 suggests that $\mathrm{SelfRefl}(A)$ acts as a version of A that hallucinates that it is interacting with μ' instead of μ. With current RL agent technology, μ' might actually be easier to implement than $\mathrm{SelfRefl}(A)$. But if RL agents in the future are capable of self-reflecting on their own, neither construction might be necessary. Thus, we consider $\mathrm{SelfRefl}(A)$ to be the more natural way to implement pseudo-visibility-based behavior, because the lack of self-reflection in current RL technology is an agent shortcoming, not an environment shortcoming.

Informally, it seems that $\mathrm{SelfRefl}(A)$ should outperform A in $\mu_{Y,k}$ (and thus the resulting trained policy should do a better job at driving the NPC in the game G'_q of Definition 3), because the kind of self-reflection apparently needed ("What would I do if this pseudo-visible character were invisible?") is prosthetically built into $\mathrm{SelfRefl}(A)$. We do not currently know of any way to articulate this better-performance conjecture formally. One difficulty is that, although current state-of-the-art RL agents are not designed to engage in such self-reflection, nevertheless, for stating formal theorems about the relative performance of $\mathrm{SelfRefl}(A)$ in general, we would have to anticipate the possibility that A might already engage in said self-reflection. If so, then the $\mathrm{SelfRefl}$ transformation might not improve A's performance. What we can do, however, is make some informal arguments based on the following observation:

Lemma 13. *Let μ'' be the same as μ' from Definition 12, except for the following differences:*

- *When $\mathrm{detect\_pseudovis}(O_t) = \mathrm{True}$, let μ'' output (O_t, a) where a is a pseudo-randomly generated action (unlike μ' which outputs (O_t, a) where a is the action the agent would take on observation $(D(s_t), -1)$).*
- *Rather than applying penalty q to the agent if the agent reacts to a pseudo-visible player, instead apply penalty q to the agent if the agent fails to take action a in response to observation (O_t, a) when $a \neq -1$.*

Then: If μ is a pseudo-random MDP, then so is μ''.

Proof. Trivial. □

Lemma 13 together with Proposition 12 strongly suggests that if A performs well on pseudo-random MDPs, then $\mathrm{SelfRefl}(A)$ should inherit that good performance on $\mu_{Y,k}$. This is because, by Proposition 12, the performance of $\mathrm{SelfRefl}(A)$ on $\mu_{Y,k}$ is identical to that of A on μ', and μ' closely resembles the pseudo-random MDP μ'' (Lemma 13). In μ'', in certain states (those states with pseudo-visible Y within the agent's field of view), the agent's view is annotated with a pseudo-random action and the agent is penalized for not taking that action. In μ', the annotated action is not pseudo-random, but calculated from the agent's own action policy. If anything, this should make μ' easier to learn, not harder. It seems that in the worst case, if A completely failed to learn any patterns in the μ' annotations, those annotations would seem random: μ' would seem like μ'' to A. Thus, although we do not currently know how to articulate a theorem stating that $\mathrm{SelfRefl}(A)$ should perform better on $\mu_{Y,k}$, at least we have a compelling informal argument that if A performs well at pseudo-random MDPs, then $\mathrm{SelfRefl}(A)$'s performance on $\mu_{Y,k}$ should benefit accordingly.

References

Aldini, A.; Graziani, P.; and Tagliaferri, M. 2020. Reasoning about ignorance and beliefs. In *CIFMA*.

Alexander, S. A.; Castaneda, M.; Compher, K.; and Martinez, O. 2022. Extending environments to measure self-reflection in reinforcement learning. *Preprint*.

Bell, J. H.; Linsefors, L.; Oesterheld, C.; and Skalse, J. 2021. Reinforcement learning in Newcomblike environments. In *NeurIPS*.

Johansson, M.; Eladhari, M. P.; and Verhagen, H. 2012. Complexity at the cost of control in game design. In *CGAT*.

Nozick, R. 1969. Newcomb's problem and two principles of choice. In Rescher, N., ed., *Essays in honor of Carl G. Hempel*. Springer. 114–146.