Characterising Simulation-Based Program Equilibria

Emery Cooper, Caspar Oesterheld, Vincent Conitzer

Carnegie Mellon University
emerycooper@cmu.edu, oesterheld@cmu.edu, conitzer@cs.cmu.edu

Abstract

In Tennenholtz's program equilibrium, players of a game submit programs to play on their behalf. Each program receives the other programs' source code and outputs an action. This can model interactions involving AI agents, mutually transparent institutions, or commitments. Tennenholtz (2004) proves a folk theorem for program games, but the equilibria constructed are very brittle. We therefore consider *simulation-based programs* – i.e., programs that work by running opponents' programs. These are relatively robust (in particular, two programs that act the same are treated the same) and are more practical than proof-based approaches. Oesterheld's (2019) ϵGroundedπBot is such an approach. Unfortunately, it is not generally applicable to games of three or more players, and only allows for a limited range of equilibria in two player games. In this paper, we propose a generalisation to Oesterheld's (2019) ϵGroundedπBot. We prove a folk theorem for our programs in a setting with access to a shared source of randomness. We then characterise their equilibria in a setting without shared randomness. Both with and without shared randomness, we achieve a much wider range of equilibria than Oesterheld's (2019) ϵGroundedπBot. Finally, we explore the limits of simulation-based program equilibrium, showing that the Tennenholtz folk theorem cannot be attained by simulation-based programs without access to shared randomness.

1 Introduction

Consider a game in which, rather than choosing actions, players submit programs (McAfee 1984; Howard 1988; Rubinstein 1998, Sect. 10.4; Tennenholtz 2004). Programs receive as input the source code of the opponents' programs, and output actions. E.g., in the (one-shot) prisoner's dilemma, the programs might be CliqueBots: 'cooperate if the other program is equal to this program, else defect'. CliqueBots form a cooperative equilibrium. This illustrates how such games – *program games* – allow for new equilibria.

Traditional game-theoretic concepts, such as Nash equilibrium, have been developed without mutual transparency in mind, and may therefore be less relevant to interactions involving AI agents whose source codes might be available to other agents (Conitzer and Oesterheld 2023). Program games could model interactions between principals designing AI agents, interactions between AI agents themselves, or smart contracts. This could include interactions between large language models where the programs are prompts provided by humans and interpreted by the models. Program games can also model some interactions between humans, such as those involving mutually conditional commitments, or transparent institutions (Critch, Dennis, and Russell 2022).

Tennenholtz (2004) proves a folk theorem for program equilibria (cf. Rubinstein 1998, Sect. 10.4): any set of feasible and individually rational (i.e., better-than-minimax) payoffs of a game is achievable in a program equilibrium. However, this result is proved using programs based on syntactic comparison, such as the above CliqueBot in the prisoner's dilemma. Such equilibria are extremely fragile: if someone, perhaps from a different community, writes a slightly different version of CliqueBot (CliqueBot'), then CliqueBot and CliqueBot' will defect on each other.

Previous authors have proposed different ways of achieving more robust program equilibria. One proposal is that of Löbian programs (Bárász et al. 2014; Critch 2019; Critch, Dennis, and Russell 2022; Oesterheld 2022). In the prisoner's dilemma, the Löbian FairBot roughly works by cooperating if and only if it can prove the opponent cooperates. The Löbian FairBots cooperate against each other. Unfortunately, there are practical difficulties with this approach. If the opponent program is not very interpretable (e.g., a large neural net), it may be difficult or impossible to prove things about its output.

We therefore turn our attention to simulation-based approaches, such as Oesterheld's (2019) ϵGroundedFairBot and ϵGroundedπBot. In the prisoner's dilemma, the ϵGroundedFairBot cooperates with probability ϵ and otherwise simulates the opponent and cooperates if and only if in the simulation the other player cooperates. If two ϵGroundedFairBots play against each other, the simulated player may in turn simulate the other, etc., but eventually the ϵ-probability case will be triggered, ensuring termination. This program is easy to implement and can be used against programs that are not very interpretable.

We can view Oesterheld's (2019) ϵGroundedπBot as simulating a repeated game. With probability ϵ, the program is on the first 'time step', and does not observe any previous actions. Otherwise, it simulates the other program. If the other program is another ϵGroundedπBot, that program will again be on the first time step with probability ϵ, and will do another simulation with probability $1 - \epsilon$. Overall, the total number of simulations (or 'time steps') is then geometric, with programs at each time step observing only the immediate previous time step. Oesterheld (2019) proves a correspondence between program games of ϵGroundedπBots and repeated games. Since only the previous time step is observed, the policy π must be *myopic* – i.e., only depend on the last action of the opponent.[1]

We focus on the following question: What equilibria can be obtained by simulation-based approaches? Specifically, can they obtain all equilibria in Tennenholtz' folk theorem?

Unfortunately, there is a huge gap between the Oesterheld (2019) results and Tennenholtz's folk theorem. The ϵGroundedπBot cannot perform more than one simulation (when its opponents are also ϵGroundedπBots) without running into halting issues due to infinite recursion, and so does not work well for games of more than two players. (Simulating two other players with probability $1 - \epsilon$ each leads to each simulation in expectation generating $2 - 2\epsilon$ new ones, resulting in a hydra of simulations – cf. Appendix A, in the version of this paper at arXiv:2412.14570.) Even in two-player games, the equilibria it can enforce fall short of the folk theorem.

Example 1. Consider the following three-player game (payoff matrix shown in Table 1):

- The utilities are *additively separable*, i.e., for each player i, that player's utility u_i may be decomposed into a sum of functions u_{ij}, where u_{ij} depends only on the action of player j: $u_i(a_1, a_2, a_3) = u_{i1}(a_1) + u_{i2}(a_2) + u_{i3}(a_3)$.
- Players 2 and 3 may either Cooperate (C), granting all players an additional 3 utility, or Defect (D), granting 8 utility to the defecting player. Formally, for $j = 2, 3$ we have $u_{ij}(C) = 3$ for all i, while $u_{ij}(D) = 8$ if $i = j$ and 0 otherwise.
- Player 1 may either Cooperate (C), Punish player 2 (P_2), or Punish player 3 (P_3). Playing P_2 (resp. P_3) takes 3 utility from player 2 (resp. 3) and gives it to player 3 (resp. 2). Formally, $u_{i1}(C) = 0$ for all i, while $u_{21}(P_2) = u_{31}(P_3) = -3$, $u_{21}(P_3) = u_{31}(P_2) = 3$, and $u_{11}(P_2) = u_{11}(P_3) = 0$.

		Player 2	
Plr 3	Plr 1	C	D
	C	$6, 6, 6$	$3, 11, 3$
C	P_2	$6, 3, 9$	$3, 8, 6$
	P_3	$6, 9, 3$	$3, 14, 0$
	C	$3, 3, 11$	$0, 8, 8$
D	P_2	$3, 0, 14$	$0, 5, 11$
	P_3	$3, 6, 8$	$0, 11, 5$

Table 1: The payoff matrix for Example 1

[1]This is similar to the Markov property, viewing the actions taken on the previous time step as the state – given the last action of the opponent, the ϵGroundedπBot's action does not depend on earlier actions.

Can we obtain a (C, C, C) equilibrium in this game with simulation-based programs? With the Oesterheld (2019) ϵGroundedπBot, the answer is no. As noted above, it cannot be naïvely extended to simulate more than one other ϵGroundedπBot while still halting with probability 1 (see Appendix A, arXiv:2412.14570 for details). In some three-player games, the ϵGroundedπBots can work around this, for example, by choosing one player to simulate at random. But this will not work in Example 1. Intuitively, player 1 needs to be able to punish a defecting player; but if player 1 randomises which player to simulate, and then sees that player (say, 2) defect in simulation, then player 1 will not know which player defected 'first' in the mental repeated game – e.g., it could be that 2 is defecting only because 3 defected in a *subsimulation*. That is, the simulated version of player 2 might in turn simulate player 3, observe player 3 defecting in that simulation, and defect so as to penalise player 3 for this. Thus, player 1 will not know which player to punish. See Appendix C of the version of this paper at arXiv:2412.14570 for details.

Contributions Now, imagine the program worked as follows instead: First it randomly determines its own 'time step' in the mental repeated game, according to a geometric distribution. Then if its time step is 1, it immediately outputs an action. Otherwise, it simulates the other programs, but manipulates those simulations' sources of randomness to ensure that (assuming the other programs operate similarly) they believe themselves to be on earlier time steps. This would then allow it to simulate multiple different opponents (and different time steps) while still halting. Our programs will be based on this idea.

We first consider a setting (Section 3) where programs have access to a shared random sequence. We propose a generalisation of the Oesterheld (2019) ϵGroundedπBot for this setting, which we call a *correlated ϵGroundedπBot*. This correlated ϵGroundedπBot may be viewed as simulating a repeated game between n players, and responding to the observed action history. We relate payoffs in games where all players but (at most) one are correlated ϵGroundedπBots to payoffs in repeated games. We hence obtain a folk theorem: any feasible and strictly individually rational payoffs can be obtained in a program equilibrium of correlated ϵGroundedπBots. From this, we can immediately see that these programs can solve Example 1, since the profile (C, C, C) is better for each player than that player's minimax utility. Our correlated ϵGroundedπBots can be seen as a (decentralised) implementation of Kovarik, Oesterheld, and Conitzer's joint simulation device (cf. Section 6).

We then consider the same approach in a setting without access to shared randomness (Section 4). We call the resulting program an 'uncorrelated ϵGroundedπBot'. We characterise the equilibria that can be attained by our uncorrelated ϵGroundedπBots. We give examples to show that our uncorrelated ϵGroundedπBots attain equilibria that the Oesterheld (2019) ϵGroundedπBots cannot, even with 2 players. Nonetheless, our uncorrelated ϵGroundedπBots do not achieve the full program equilibrium folk theorem, as we show with an example. While there is in general no simple relationship between program games of uncorrelated ϵGroundedπBots and repeated games, we do obtain such a result in the special case of games with additively separable utilities. From this, we derive a folk theorem in the case of games with additively separable utilities. Hence, the uncorrelated ϵGroundedπBots can still achieve a (C, C, C) equilibrium in Example 1.

Finally, we ask whether more general simulation-based programs allow for additional equilibria in the setting without shared randomness (Section 5). We give an example to show that it is possible to attain equilibria that cannot be attained by the uncorrelated ϵGroundedπBot. We then prove a negative result on the equilibria attainable by simulation-based programs. From this, we show that without shared randomness, simulation-based programs cannot give us the full Tennenholtz (2004) folk theorem. It remains an open question exactly which equilibria can be attained.

For full proofs, and appendices, see the full version of the paper at https://arxiv.org/abs/2412.14570.

2 Background and Preliminaries

We will sometimes denote the tuple $(x_1, \dots, x_n)$ by $x_{1:n}$, and the tuple $(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$ by x_{-i}. Similarly, we will use 'players $-i$' to refer to all players of a game but player i.

Normal-form games We assume familiarity with game theory. See, e.g., Osborne (2004) for an introduction. We introduce some definitions for normal-form games.

An n-player *normal-form* game comprises finite *action sets* $\mathcal{A}_i$ and *utility functions* $u_i : \prod_i \mathcal{A}_i \to \mathbb{R}$ for each player $i \in \{1, \dots n\}$. A *strategy* for player i is a probability distribution $s_i \in \Delta(\mathcal{A}_i)$ over actions for player i. A *strategy profile* is a tuple $(s_1, \dots s_n)$ of strategies for all players.

Given a joint distribution over *outcomes* $(a_1, \dots, a_n)$, the *expected utility* for player i is, writing A_i for the ith player's action (a random variable), $\mathbb{E}[u_i] = \mathbb{E}[u_i(A_{1:n})] = \sum_{a_{1:n}} \mathbb{P}(A_{1:n} = a_{1:n})\, u_i(a_1, \dots, a_n)$. When players' actions are independent, with player i following strategy s_i for each i, write $u_i(s_1, \dots, s_n) := \mathbb{E}[u_i(A_{1:n})] = \sum_{a_{1:n}} u_i(a_{1:n}) \prod_j \mathbb{P}(A_j = a_j)$.

We call a set of expected payoffs $v_{1:n}$ *individually rational (without coordination)* if it is at least as good for each player as that player's minimax utility, i.e., $v_i \geq \min_{s_{-i} \in \prod_{j \neq i} \Delta(\mathcal{A}_j)} \max_{a_i \in \mathcal{A}_i} u_i(s_{-i}, a_i)$ for all i. Say $v_{1:n}$ is *strictly* individually rational if this inequality is strict for all i.

Say that a set of payoffs $v_{1:n}$ is *feasible (without correlation)* if it is achieved by some set of (independently sampled) strategies, i.e., if for some $s_{1:n}$, for all i, $v_i = u_i(s_{1:n})$. Meanwhile, $v_{1:n}$ is *feasible with correlation* if it is achieved in expectation by some set of strategies not sampled independently: i.e., if when $A_{1:n}$ follows some joint distribution over $\prod_i \mathcal{A}_i$, we have $v_i = \mathbb{E}[u_i(A_{1:n})]$.

Program games We now define *program equilibrium* (Tennenholtz 2004). We will consider programs that are themselves deterministic, but take as input one or more random sequences. We also assume that programs always have access to their own source code. Write *apply* for the function that takes as input a program and a tuple of input arguments and runs the program on the input arguments, outputting whatever the program outputs (if anything).

Definition 1 (Program games)**.** *Given an n-player normal-form game G, the **correlated program game** of G is a game where player i's action set is the set of programs taking as input a list of $n - 1$ opponent programs, and sequences $(r_m)_{m \geq 0} \in [0, 1]^\infty$ and $(x^i_m)_{m \geq 0} \in [0, 1]^\infty$, and outputting an action in $\mathcal{A}_i$. Given a **program profile** $(p_1, \dots, p_n)$ of programs that almost surely halt against each other (i.e., halt against each other with probability 1), the **outcome distribution** is given by the distribution of output actions, $A_{1:n}$, where $A_i = apply(p_i, (p_{-i}, (r_m), (x^i_m)))$, and (r_m), $(x^1_m), \dots (x^n_m)$ are independent i.i.d. $U[0, 1]$ sequences. The **expected utility** for player i is then $\mathbb{E}[u_i(A_{1:n})]$.*

*An **uncorrelated program game** is the same, except that for each i, program i only receives a single sequence (r^i_m), independent for each i.*

If the programs do not halt against each other, we leave the outcome distribution and expected utility undefined. We will show that the programs we define halt against each other and a range of other programs. When we later consider more general programs, we will place restrictions on them to prevent them from strategically not halting (which our earlier programs also satisfy). While we can consider players randomising over programs (cf. Appendix D, arXiv:2412.14570), it will generally suffice to imagine each player submitting a single program.

Given an n-player (correlated or uncorrelated) program game and programs p_{-i} for players $-i$, a program p_i is a *best response* to p_{-i} if it achieves maximal expected utility against p_{-i} (among programs p'_i such that p_{-i}, p'_i all almost surely halt against one another). Say $(p_1, \dots p_n)$ is a *program equilibrium* if for all i, p_i is a best response to p_{-i}.

Tennenholtz (2004) proves the following folk theorem for program games (where the set of programs for each player is restricted to avoid issues related to halting). We will be interested in whether this result (or something similar) holds for simulation-based programs.

Theorem 1. *Payoffs $v_{1:n}$ are achievable in an equilibrium of the uncorrelated program game of G (given aforementioned restrictions on the programs available to each player) if and only if they are feasible (without correlation) and individually rational.*

3 Correlated Program Games

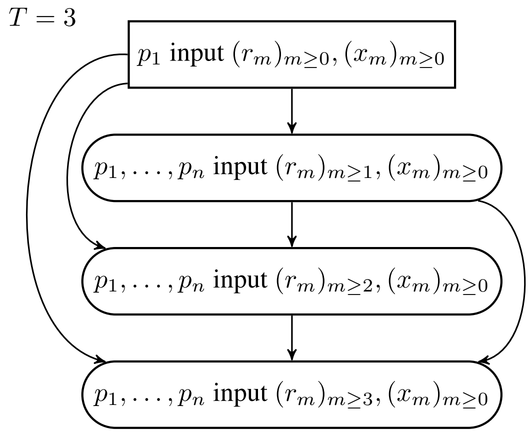
We first consider correlated program games. We use an additional variant on *apply* calls for this setting: Define $apply^*(p_i, (p_{-i}, (r_m), (x_m)))$ to be equal to $apply(p_i, (p_{-i}, (r_m), (x_m)))$ if the variables (x_m) are not *accessed* except inside further $apply^*$ calls, and R_i, a special symbol to indicate that the private random sequence was accessed, otherwise. For example, in the random-access machine model this would trigger whenever the simulated program runs the command for reading (with either direct or indirect addressing) one of memory addresses containing the independent random bits (e.g., loading from, comparing with the contents of, or performing an arithmetic operation using the contents of such a memory location, in the formalism of 'MIX' from Knuth (1997, Sec. 1.3.1, Volume 1)). We do not include simply passing the (x_m) on via *apply* calls as 'accessing' them – we may assume that these pass on only a pointer to the (x_m).

The idea of $apply^*$ calls is to screen off any dependence on private random variables (by replacing the output in cases where it could depend on them). Computing whether a program's output *depends* on the (x_m) is undecidable (by Rice's theorem), so we instead look at whether the program *accesses* the (x_m). We will discuss the purpose of this later.

Our algorithm We present our algorithm, which we call a *correlated ϵGroundedπBot*, in Algorithm 1. Note that the ϵGroundedπBot *policy*, π_i, takes as input action histories in which some actions may be replaced with R_js, and outputs an action for player i (possibly also depending on random variables r_0 and x_0). The program essentially simulates a repeated game between the programs. It works as follows (see also Figure 1 for an illustration):

- First, it determines the number T of previous 'time steps' as the number of elements of the shared random sequence (r_m) until the first element that is less than ϵ. This gives it a geometric distribution over its time step.
- If $T = 0$, it immediately outputs its policy π_i applied to the empty history.
- Otherwise, it simulates all programs (including itself) at earlier time steps. It does this by calling them with (between 1 and T) initial elements of the shared random sequence removed – effectively putting them on earlier 'time steps' (or with smaller values of T). This ensures that ϵGroundedπ_i bots will halt against one another, since the tree of *apply* calls will then have depth at most T.
- The simulated programs are not able to tell how many 'time steps' there are 'after' them, and hence cannot tell that they are simulated. This is analogous to how in a repeated game that ends with probability ϵ each round, players do not know if they are on the last time step.
- It then outputs its policy π_i applied to the action history.

Note that aside from the availability of shared random variables, the main difference between our algorithm and the ϵGroundedπBot from Oesterheld (2019) is that we allow for more than 2 players, and look back at more than one time step. This is possible because having the random variables as an input argument allows for manipulating the random variables that simulated programs receive, thus allowing us to correlate their maximum simulation depth. This in turn allows us to consider non-myopic policies π_i. One final thing to note is the use of $apply^*$ calls rather than *apply* calls. This serves a few purposes. First, programs independently randomising can cause problems, as we will see in later sections, and using $apply^*$ calls allows for punishing programs for doing this. Whilst we could additionally allow our programs to observe the results of the randomisation using the (x_m)s, this would result in the simulations being correlated in a way that the actual programs would not be, which would complicate our results. Finally, by making it so the output of



$T = 3$
p_1 input $(r_m)_{m\geq 0}, (x_m)_{m\geq 0}$
$p_1, \ldots, p_n$ input $(r_m)_{m\geq 1}, (x_m)_{m\geq 0}$
$p_1, \ldots, p_n$ input $(r_m)_{m\geq 2}, (x_m)_{m\geq 0}$
$p_1, \ldots, p_n$ input $(r_m)_{m\geq 3}, (x_m)_{m\geq 0}$

Figure 1: The simulation tree in a game between n correlated ϵGroundedπBots with common ϵ in the case $T = 3$ (i.e., when $r_0, r_1, r_2 \geq \epsilon$ and $r_3 < \epsilon$). An arrow from a group of programs with one input, to a second group of programs with a second input, indicates that the first group with the first input all run all of the second group of programs with the second input (in this case, via an $apply^*$ call). We omit the part of the input that is simply the list of other programs.

Algorithm 1 Correlated ϵGroundedπ_iBot

```
Input: Programs $\boldsymbol{p}_{-i}$, sequences $(r_m)_{m=0}^{\infty}, (x_m)_{m=0}^{\infty} \in [0,1]^{\infty}$
$T \leftarrow \min\{t : r_t < \epsilon\}$
for $t = 1$ to $T$ do
    for $j = 1$ to $n$ do
        $a_j^t \leftarrow apply^*(p_j, (\boldsymbol{p}_{-j}, (r_m)_{m\geq T+1-t}, (x_m)_{m\geq 0}))$
return $\pi_i(\boldsymbol{a}^{1:T}; r_0, x_0)$
```

the ϵGroundedπBots themselves will not depend on the private random bits, it allows the history to still be observed normally after such randomisation occurs, allowing for non-grim-trigger strategies.

Halting results We show that the ϵGroundedπBots halt against each other, and that this halting is relatively robust. Note that checking whether $r_t < \epsilon$ might take arbitrarily long (or be impossible) – we abstract away this problem by simply assuming this computation is instant. Similarly, we assume throughout that the ϵGroundedπBot policies π_i are computable.

Theorem 2. *Suppose that p_i is a correlated ϵ_iGroundedπ_iBot with $\epsilon_i > 0$, for $i = 1, \ldots n$. Then programs $p_1, \ldots, p_n$ almost surely halt against each other.*

The proof for this result and others may be found in Appendix K of the version of the paper at arXiv:2412.14570. The intuition for this result is as follows: We can consider the 'time step' (T_i) from the perspective of the program with minimal ϵ_i. The ϵGroundedπBots' simulations are always on a smaller time step (in the sense of T_i) than they are. On time step 1 (if not earlier), they must halt without running any simulations. Thus, the overall process must halt.

If one of the programs is not an ϵGroundedπBot, the programs still halt under certain conditions. Given program p_i with input programs p_{-i}, say that an *apply* (resp. *apply** call) is *standard* if the program being called is p_j for some j, with input programs p_{-j}.

Theorem 3. *Suppose that program p_i has the following properties: (i) It never makes a standard apply/apply* call to itself. (ii) Its standard apply/apply* calls all have first input sequence of the form $(r_m)_{m \geq k}$ with $0 \leq k \leq c$ for some constant c. (iii) p_i almost surely makes only finitely many standard apply/apply* calls directly (i.e., not counting those made inside other apply calls). (iv) p_i almost surely halts whenever all its standard apply/apply* calls halt. (v) For all events A with $\mathbb{P}\left(((r_m),(x^i_m)) \in A\right) = 0$, p_i almost surely makes no standard apply/apply* calls with input sequences in A. Then if for each $j \neq i$, p_j is a (correlated) ϵ_jGroundedπ_jBot with $\epsilon_j > 0$, programs $p_{1:n}$ almost surely halt against each other.*

Expected runtime By default, ϵGroundedπBots have infinite expected runtime for small ϵ, despite halting with probability 1. This is due to an exponential blow-up of the number of simulations, with the number of simulations increasing by a factor of at least n at each level of simulation. However, these are mostly duplicates of each other – there are only n distinct simulations at each 'time step' (i.e, with each possible pair of input sequences). Thus, we may use *memoization* (Michie 1968; Norvig 1991; Abelson and Sussman 1996, Exercise 3.27) to circumvent this problem – if we store the results of computations of *apply** calls, we can avoid redoing them, and obtain finite expected runtime.

Theorem 4. *Suppose that program p_i is a correlated ϵ_iGroundedπ_iBot for $i = 1, \ldots n$. Then if at least two p_i have $\epsilon_i < 1/2$, every p_i has infinite expected runtime against p_{-i}. However, with memoization, the expected runtime is finite and polynomial in $\frac{1}{\min_i \epsilon_i}$, provided that for all i, $\pi_i(\boldsymbol{a}^{1:t})$ runs in polynomial time in t.*

Repeated games with correlating signals and screening Our results will come from analogy to repeated games. We will first need to define a slight variant on repeated games. These will have two new features. *Screening*: we allow each player at each time step to 'screen' their action, thereby hiding the action from the other players. This is analogous to actions being 'screened' from the ϵGroundedπBots by the *apply** calls when they are based on private randomness. *Correlating signals:* We assume that at each time step, players all receive the same $U[0,1]$ random variable. We now define screened histories and policies.

Given a game G with n players, define a *screened history* of length t to be an element $\boldsymbol{\alpha}^{1:t} \in (\prod_i(\{R_i\} \cup \mathcal{A}_i))^t$ (where α^k_i then denotes the action of player i on time step k). Say that a screened history is *compatible* with an *unscreened history* in $(\prod_i \cup \mathcal{A}_i)^t$ if they agree up to screening (i.e., they agree on all elements that are not R_i (for some i) in the screened history). That is, screened history $\boldsymbol{\alpha}^{1:t} \in (\prod_i(\{R_i\} \cup \mathcal{A}_i))^t$ is compatible with unscreened history $\boldsymbol{\beta}^{1:t} \in (\prod_i(\mathcal{A}_i))^t$ iff for all j, t', whenever $\alpha^{t'}_j \neq R_j$ we have $\alpha^{t'}_j = \beta^{t'}_j$.

Similarly, a *screened policy* for player j is a map from, for $t = 0, 1, \ldots$, screened histories of length t and correlating signals $q \in [0,1]$ to $\Delta(\mathcal{A}_j \cup \{R_j\})$. An *unscreened policy* in a repeated game with correlation and screening is the same, but instead mapping to $\Delta(\mathcal{A}_j)$. Say that a screened policy π^*_i is *compatible* with an unscreened policy π_i if they agree up to screening – i.e., for each action $a_j \in \mathcal{A}_j$, each history $\boldsymbol{\alpha}^{1:t}$, and each q, π^*_i assigns no more probability than does π_i to action a_j. That is, $\pi^*_i(a_j \mid \boldsymbol{\alpha}^{1:t}; q) \leq \pi_i(a_j \mid \boldsymbol{\alpha}^{1:t}; q)$, writing $\pi_i(a_j \mid \boldsymbol{\alpha}^{1:t}; q)$ for the probability mass given by $\pi_i(\boldsymbol{\alpha}^{1:t}; q)$ to a_j.

Then, given screened and unscreened policies π^*_i and π_i for player i, define their joint distribution by $(\tau_i, \tau^*_i)((a_j, a'_j) \mid \boldsymbol{\alpha}^{1:t}; q) := \tau^*_i(a_j \mid \boldsymbol{\alpha}^{1:t}; q)$ if $a'_j = a_j$, and $\tau_i(a_j \mid \boldsymbol{\alpha}^{1:t}; q) - \tau^*_i(a_j \mid \boldsymbol{\alpha}^{1:t}; q)$ if $a'_j = R_j$, and 0 otherwise. Given a repeated game with correlation and screening, and screened and unscreened policy profiles $\boldsymbol{\tau}^*$ and $\boldsymbol{\tau}$ respectively, the probability of the screened and unscreened histories at time t (given that the game does not end before then) being equal to (compatible) histories $\boldsymbol{\alpha}^{1:t}$ and $\boldsymbol{\beta}^{1:t}$ is, where $q_1, q_2, \ldots \overset{\text{iid}}{\sim} U[0,1]$, defined as $\prod_{k=1}^t \mathbb{P}\left(\boldsymbol{\tau}^*(\boldsymbol{\alpha}^{1:k-1}; q_k) = \boldsymbol{\alpha}^k \cap \boldsymbol{\tau}(\boldsymbol{\alpha}^{1:k-1}; q_k) = \boldsymbol{\beta}^k\right) := \prod_{k=1}^t \mathbb{E}\left[\prod_{i=1}^n (\tau_i, \tau^*_i)((\beta^k_i, \alpha^k_i) \mid \boldsymbol{\alpha}^{1:k-1}; q_k)\right]$. (Thus, we assume that given correlating signals, samples from policies are independent.)

The history length is independent of the actions taken, with, for *discount factor* $1-\epsilon$, probability $(1-\epsilon)^{t-1}\epsilon$ of the final history having length t. This therefore defines a distribution over the final histories, and hence over the final unscreened history.

Given that the game ends after action history $\boldsymbol{a}^{1:t}$, player i receives utility $\sum_{t'=1}^t u_i(a^{t'}_1, \ldots a^{t'}_n)$. The *expected utility* for player i is then the expectation of the utility for player i, taken with respect to the distribution over unscreened histories.

Screened and unscreened policy profiles $\boldsymbol{\tau}^*, \boldsymbol{\tau}$ are an *equilibrium* if for all i, (τ^*_i, τ_i) maximizes expected utility against (τ^*_{-i}, τ_{-i}) among pairs of compatible screened and unscreened policies for player i.

Given a game G, we write G_ϵ for the corresponding correlated repeated game with screening with discount factor $1-\epsilon$. We obtain the following folk theorem, as for usual repeated games. The idea is to consider policies that punish screening, such that it won't happen on the path of play.

Theorem 5. *Suppose that payoffs $v_{1:n}$ for game G are feasible with correlation and strictly individually rational. Then for ϵ sufficiently small, there exists an equilibrium of G_ϵ in which player i receives expected utility $\frac{v_i}{\epsilon}$, for all i.*

Relating the program game to the repeated game We now translate from programs to policies in correlated repeated games with screening. Essentially, the policy of a program at time t will be determined by its action distribution conditional on $T+1 = t$, given the simulated actions of opponents at earlier time steps. The screening will be determined by whether the program accesses its private random sequence.

Given programs $p_1, \dots p_n$ for game G, we define the *associated screened and unscreened policies* for the repeated game G_ϵ as follows. The screened policy is, where we use ω to denote the empty history: $\tau_i^*(\omega; q) := \mathbb{E}[apply^*(p_i, (p_{-i}, (r_m), (x_m))) \mid r_0 = \epsilon q]$ and $\tau_i^*(\boldsymbol{\alpha}^{1:t}; q) := \mathbb{E}[apply^*(p_i, (p_{-i}, (r_m), (x_m))) \mid r_0 = \epsilon + (1-\epsilon)q,\ \boldsymbol{a}^{1:t} = \boldsymbol{\alpha}^{1:t}, T = t]$, where T and $\boldsymbol{a}^{1:t}$ are as defined in Algorithm 1, i.e., T is minimal such that $r_T < \epsilon$ and $a_j^k := apply^*(p_j, (p_{-j}, (r_m)_{m \geq T+1-k}, (x_m)_{m \geq 0}))$. The unscreened policy is the same but with *apply* instead of *apply** calls: $\tau_i(\omega; q) := \mathbb{E}\left[apply(p_i, (p_{-i}, (r_m), (x_m))) \mid r_0 = \epsilon q\right]$ and $\tau_i(\boldsymbol{\alpha}^{1:t}; q) := \mathbb{E}[apply(p_i, (p_{-i}, (r_m), (x_m))) \mid r_0 = \epsilon + (1-\epsilon)q,\ \boldsymbol{a}^{1:t} = \boldsymbol{\alpha}^{1:t}, T = t]$. If for some $\boldsymbol{\alpha}^{1:t}$ the event $\boldsymbol{a}^{1:t} = \boldsymbol{\alpha}^{1:t}$ occurs with probability 0, we simply define the screened and unscreened policies for G_ϵ arbitrarily at $\boldsymbol{\alpha}^{1:t}$. (For our results, it does not matter how the screened and unscreened policies are defined at histories with probability 0, since these histories will also have probability 0 in the associated repeated game with the associated policies.) Note that these form compatible screened and unscreened policies, where they are defined.

For ϵGroundedπ_iBots, the associated repeated game policy then corresponds to the policy π_i:

Lemma 1. *If p_j is a correlated ϵGroundedπ_jBot, then regardless of programs p_{-j}, its associated repeated game policies are, where defined, given by screened policy: $\tau_j^*(\omega; q) = \pi_j(\omega; \epsilon q, x_0)$ unless $\pi_j(\omega; \epsilon q, x_0)$ accesses x_0, in which case $\tau_j^*(\omega; q) = R_j$, and $\tau_j^*(\boldsymbol{\alpha}^{1:t}; q) = \pi_j(\boldsymbol{\alpha}^{1:t}; \epsilon + (1-\epsilon)q, x_0)$ unless this accesses x_0, in which case $\tau_j^*(\boldsymbol{\alpha}^{1:t}; q) = R_j$. The unscreened policy is: $\tau_j(\omega; q) = \mathbb{E}\left[\pi_j(\omega; \epsilon q, x_0)\right]$ and $\tau_j(\boldsymbol{\alpha}^{1:t}; q) = \mathbb{E}\left[\pi_j(\boldsymbol{\alpha}^{1:t}, \epsilon + (1-\epsilon)q, x_0\right]$.*

If all but one program is an ϵGroundedπBot, all players obtain the same utility as their repeated game policies do in G_ϵ, up to a factor of ϵ:

Theorem 6. *Let G be an n-player correlated program game. Let p_{-i} be correlated ϵGroundedπBots with policies π_{-i}, and p_i be any program for player i such that programs $p_1, \dots, p_n$ almost surely halt against each other. For each j, let τ_j and τ_j^* be the associated G_ϵ policies for player j. Then $\mathbb{E}\left[\boldsymbol{u} \mid \boldsymbol{p}\right] = \epsilon\, \mathbb{E}_{G_\epsilon}[\boldsymbol{u} \mid \boldsymbol{\tau}, \boldsymbol{\tau}^*]$.*

This gives us the following folk theorem:

Corollary 1. *Suppose that G is an n-player program game. Then for any feasible (with correlation), strictly individually rational $v_{1:n}$ there exists a program equilibrium of correlated ϵGroundedπBots (with a common choice of ϵ).*

Example 2. Pirates 1, 2, 3 have plundered 30 gold pieces. Each pirate can Cooperate (C) with the other pirates, Defect (D) and try to run off with more than their fair share of the gold pieces, or hire a Lawyer (L) to enforce a fair split. If all three pirates Cooperate, they each receive 10 gold pieces. If one or more pirates defect, and the rest cooperate, then the first up-to-two pirates in numeric order to defect receive 14 gold pieces, and all other pirates receive 0. (The defecting pirates clumsily drop the rest of the gold pieces in the sea.) If one pirate hires a lawyer, and the other pirates cooperate, the pirate who hired the lawyer receives 9 gold pieces, and other pirates receive 10 gold pieces as usual. If at least one pirate defects, and exactly one pirate hires a lawyer, the outcome is the same as if the latter pirate cooperated (they have no money to pay the lawyer). If two pirates hire a lawyer, the lawyer enforces a fair split, and all pirates receive 9 gold pieces, regardless of what the third pirate plays. We show the payoffs in Table 2.

		Player 2		
Plr 3	Plr 1	C	D	L
	C	$10, 10, 10$	$0, 14, 0$	$10, 9, 10$
C	D	$14, 0, 0$	$14, 14, 0$	$14, 0, 0$
	L	$9, 10, 10$	$0, 14, 0$	$9, 9, 9$
	C	$0, 0, 14$	$0, 14, 14$	$0, 0, 14$
D	D	$14, 0, 14$	$14, 14, 0$	$14, 0, 14$
	L	$0, 0, 14$	$0, 14, 14$	$9, 9, 9$
	C	$10, 10, 9$	$0, 14, 0$	$9, 9, 9$
L	D	$14, 0, 0$	$14, 14, 0$	$9, 9, 9$
	L	$9, 9, 9$	$9, 9, 9$	$9, 9, 9$

Table 2: The game between the three pirates.

Note that (C, C, C) is the unique symmetric (strictly) *Pareto optimal* strategy profile (strictly Pareto optimal meaning that for any other strategy profile $(s_1, s_2, s_3) \neq (C, C, C)$, we have $u_i(s_1, s_2, s_3) < u_i(C, C, C)$ for some i). Can correlated ϵGroundedπBots enforce a (C, C, C) equilibrium, for a utility of 10 for each player? Corollary 1 gives that the answer is yes: The minimax utility for each player is at most 9. Hence, this payoff profile is feasible and strictly individually rational. We explicitly describe a correlated ϵGroundedπBot equilibrium for this game in Appendix L.1 of the version of this paper at arXiv:2412.14570.

4 Epsilon-Grounded Simulation without Shared Randomness

We now consider uncorrelated program games, where no correlated random sequence is available to the programs, so that programs can only randomise independently. We present our algorithm, the *uncorrelated ϵGroundedπBot*, in Algorithm 2. It works almost identically to correlated ϵGroundedπBots: it determines its time step based on the first element of the random sequence that is less than ϵ, and then simulates the other programs at earlier time steps.

Halting Behaviour We obtain essentially identical halting, halting robustness, and expected runtime results as for the correlated πBots – the uncorrelated ϵGroundedπBots

Algorithm 2 uncorrelated ϵGroundedπBot

```
Input: Programs $\boldsymbol{p}_{-i}$, sequence $(r_m)_{m=0}^{\infty} \in [0,1]^\infty$
$T \leftarrow \min\{t : r_t < \epsilon\}$
for $t = 1$ to $T$ do
    for $j = 1$ to $n$ do
        $a_j^t \leftarrow apply(p_j, (\boldsymbol{p}_{-j}, (r_m)_{m \geq T+1-t}))$
return $\pi_i(\boldsymbol{a}^{1:T}; r_{0:T})$
```

halt against each other, have finite expected runtime against each other if memoization is used, and still halt if one program is replaced with a program satisfying conditions similar to those in Theorem 3. We therefore state these results formally only in Appendix E, arXiv:2412.14570.

No full folk theorem We cannot obtain the full folk theorem with the uncorrelated ϵGroundedπBots. We show this as follows:

Example 3. Recall the game between the pirates from Example 2. Can we still obtain a (C, C, C) equilibrium with uncorrelated ϵGroundedπBots? Suppose that programs p_1, p_2, p_3 are ϵGroundedπ_iBots which play C against one another with probability 1. We claim that this cannot be an equilibrium. To see this, consider a program p_1' which looks at r_{T+1} and plays D if $r_{T+1} < q$, for some q, and otherwise plays C.

Now, note that in computing the $\boldsymbol{a}^{1:t}$, the *apply* call tree of the ϵGroundedπBots only has calls with the sequence (r_m) truncated by at most T. Thus, r_{T+1} is the same for each call in the call tree. Then, if $r_{T+1} \geq q$, which occurs with probability $1-q$, p_i $(i \neq 1)$ must output C. Moreover, the outputs of the three programs are independent. Hence, player 1 receives utility at least $\min_{s_2,s_3} u_1((1-q)C+qD, (1-q)C+qs_2, (1-q)C+qs_3)$. Now, when minimising the above quantity with respect to s_2 and s_3, we may assume WLOG that $s_2(C) = s_3(C) = 0$, since players 2 and 3 playing C is always at least as good as other actions for player 1. Moreover, a mixture of D and L is always no better than both players using the better one of the two (see Appendix L.2, arXiv:2412.14570, for a derivation of this). We then need only consider $s_2 = s_3 = L$ and $s_2 = s_3 = D$. Taking $q = \frac{3}{4}$ and considering both possibilities, we obtain that player 1 receives utility equal to the minimum of $10\frac{3}{4} > 10$ and $10\frac{21}{32} > 10$. Thus, (p_1, p_2, p_3) cannot be an equilibrium.

The above example illustrates a general problem: if an opponent randomises in such a way that their actions at different time steps in the ϵGroundedπBots' simulations are perfectly correlated, deviation with probability q can be detected with probability at most q. Since actions are then uncorrelated between programs, a player may be able to benefit from deviating.

Characterising the equilibria We now characterise the equilibria that may be obtained by uncorrelated ϵGroundedπBots.

Proposition 1. *Suppose that payoffs $v_{1:n}$ are feasible (without correlation), with $v_{1:n} = u_{1:n}(s_{1:n})$. Suppose further that $v_i \geq \min_{(S_j':\mathcal{A}_i\to\Delta(\mathcal{A}_j))_{j\neq i}} \max_{q\in[0,1],s_i'\in\Delta(A_i)}$ $u_i\left((1-q)s_i + qs_i', \left((1-q)s_j + qS_j' \bullet s_i'\right)_{j\neq i}\right) - \lambda q$, for some $\lambda > 0$, writing $S_j' \bullet s_i'$ for $\sum_a s_i'(a)S_j'(a)$. Then there exist uncorrelated ϵGroundedπBot equilibria with payoffs arbitrarily close to $v_{1:n}$.*

The main ideas for the above are as follows: We can approximate any strategy s_i arbitrarily closely via a repeating sequence of pure actions over the different time steps. As ϵ goes to zero, the chance of detecting a deviation from this sequence is almost as large as the probability of said deviation. Program j then plays a punishment strategy (S_j') according to the observed deviation action.

Using similar ideas to Example 3, we obtain a converse:

Proposition 2. *Suppose that there exists an uncorrelated ϵGroundedπBot equilibrium with payoffs $v_{1:n}$. Then $v_{1:n}$ must be feasible, and for each i, we must have $v_i \geq \min_{(S_j':\mathcal{A}_i\to\Delta(\mathcal{A}_j))_{j\neq i}} \max_{q\in[0,1],s_i'\in\Delta(\mathcal{A}_i)}$ $u_i\left((1-q)s_i + qs_i', \left((1-q)s_{-i} + qS_j' \bullet s_i'\right)_{j\neq i}\right)$.*

We also obtain the following corollary (by considering small probability deviations in the above result):

Corollary 2. *Suppose that $v_{1:n} = u_{1:n}(s_{1:n})$ in some game G. Then if for some i, there exists $s_i' \in \Delta(\mathcal{A}_i)$ such that $\min_{(s_j'\in\Delta(\mathcal{A}_i))_{j\neq i}} \sum_j u_i(s_j', s_{-j}) - v_i > 0$, then $v_{1:n}$ is not attainable in any uncorrelated ϵGroundedπBot equilibrium.*

Since the behaviour of the uncorrelated ϵGroundedπBots is unaffected by changes to a program that do not affect the program's output against the other programs on a given input, the equilibria are also robust to such changes. As a special case, if an uncorrelated ϵGroundedπBot equilibrium entails one program playing a pure strategy at equilibrium, then replacing that program with any other program that almost surely plays that same pure strategy against the ϵGroundedπBots will leave the expected utilities of all players unchanged.

Even in two-player games, our ϵGroundedπBots retain an advantage over the Oesterheld (2019) ϵGroundedπBots. In particular, they allow for better detection and punishment of deviation from mixed strategies, which allows for more equilibria. This is illustrated in the following example:

Example 4. Consider the following trust game (cf. Berg, Dickhaut, and McCabe (1995); with payoffs shown in Table 3): Player 1 receives \$3, which he may Keep (K) or Send to the other player (S). Money sent to the other player is doubled. Player 2 may then either be Greedy (G) and take \$4 and send only \$2 back, or be Charitable (C) and take only \$2, giving \$4 to player 1. We show in Appendix L.3, arXiv:2412.14570, that there exists an uncorrelated ϵGroundedπBot equilibrium in which player 1 plays S and player 2 randomises at an approximately $1:1$ ratio between T and R, whereas the Oesterheld (2019) ϵGroundedπBot cannot attain any equilibria in which player 1 plays S and player 2 mixes.

In general, there is no simple relationship between utilities in games of ϵGroundedπBots and repeated games. We discuss this further in Appendix F of the version of this paper at arXiv:2412.14570. In the special case of additively separable utilities, we do obtain a folk theorem:

		Player 2	
		G	C
Player 1	K	$3, 0$	$3, 0$
	S	$2, 4$	$4, 2$

Table 3: The trust game from Example 4

Theorem 7. *Suppose G is an n-player program game with additively separable utilities. Then for any set of payoffs $v_{1:n}$, there exists a program equilibrium of uncorrelated ϵGroundedπBots (with a common ϵ) whenever $v_{1:n}$ is feasible (without correlation) and strictly individually rational.*

Example 5. Consider Example 1 from the introduction. Can the uncorrelated ϵGroundedπBots attain a (C, C, C) equilibrium? Since the game has additively separable utilities, Theorem 7 gives that they can. In particular, we may use the following policies: $\pi_1(\boldsymbol{a}^{1:t}) = C$ if no player has defected, and P_i if player i is the first to defect. For $i > 1$, use $\pi_i(\boldsymbol{a}^{1:t}) = C$ if no player has defected and D otherwise. Player 1 cannot benefit by deviating. Meanwhile, if player 2, say, defects with probability q, the defection will be observed by each other player with probability at least $q(1 - \epsilon)$ (independently between players). Thus, the gain from defecting will be $5q - (3 + 3)q(1 - \epsilon)$. Thus, this is an equilibrium provided $\epsilon < 1/6$.

5 General Simulation-based Programs

We now turn to more general simulation-based programs, in the uncorrelated setting from the last section. Can we do better than the uncorrelated ϵGroundedπBots? We start by defining the class of programs we are interested in.

Say a program p_i is *simulationist* if it only references the input programs, p_{-i}, via calls of the form $apply(p_j, (p'_{-j}, (r'_m)))$ where each program in p'_{-j} is either one of the input programs, or is itself simulationist, (r'_m) is any sequence, and we allow $i = j$.

This definition is inductive – e.g., uncorrelated ϵGroundedπBots are simulationist, as are, say, DefectBots. Meanwhile, a program p that ran the input programs on a mix of such programs and input programs would be simulationist, as would a program q that ran the input programs on p. Note that we exclude programs that run the input programs on programs that are *not* simulationist. We explain this further in Appendix G, arXiv:2412.14570.

We give here a simple example of a game in which it is possible to sustain a profile of equilibrium payoffs with *some* simulationist programs, but not with uncorrelated ϵGroundedπBots.

Example 6. Consider the following trust game: Player 1 receives \$3, which she may Keep (K) or Send to the other player (S). Money sent to Player 2 is doubled. Player 2 may then be Greedy (G) and keep all \$6, or Fair (F) and return \$4 to player 1. The payoff matrix is shown in Table 4. Can an (S, F) equilibrium be sustained?

For the uncorrelated ϵGroundedπBots, Corollary 2 shows that the answer is no: Note that $u_2(S, F) = 2$. Let $s'_2 = G$.

		Player 2	
		G	F
Player 1	K	$3, 0$	$3, 0$
	S	$0, 6$	$4, 2$

Table 4: The trust game from Example 6.

Then, we have $\min_{s'_1 \in \Delta(\mathcal{A}_1)}(u_2(s'_1, F) - v_2) + (u_2(S, G) - v_2) = u_2(K, F) - 2 + u_2(S, G) - 2 = 2 > 0$.

However, an (S, F) equilibrium can be sustained as follows: Player 2 submits program p_2, which just outputs F. Player 1 submits program p_1, which simulates the other program twice on independent input sequences (e.g., by splitting its input sequence into odd and even entries), and then outputs S only if both simulations output F. Since p_2 never calls any other program, p_1 and p_2 halt against each other, with outputs (S, F). It is easy to show that this is then an equilibrium.

The above example is relatively extreme, but illustrates two ideas for outperforming uncorrelated ϵGroundedπBots. First, taking multiple samples. Typically, we cannot take multiple independent samples of a program's action without halting issues. In the previous example, we could, because of program 2 always being effectively on time step 1. In general, we can take multiple samples of a program's action, provided that we ensure all samples are on earlier time steps than our own. Second, using different distributions over the time step for different players. In the above example, there is no need for player 2 to simulate player 1, so we may have player 2 be always on time step 1. More generally, we might do better in some games by having players have different (and potentially non-geometric) distributions over their time step. We discuss these ideas further in Appendix I, arXiv:2412.14570.

Negative results We will now explore the limits of equilibria enforceable with simulationist programs. We first need to place a mild additional constraint on the programs we consider:

Say that programs $p_1, \ldots, p_n$ halt against each other *output independently* if they almost surely halt against each other, and this remains true if one program, p_i, is replaced by a program p'_i that almost surely halts on any input that p_i almost surely halts on. This condition rules out programs that, e.g., loop forever once they determine that another program has played the wrong action. In particular, if programs halt against each other output independently, they must still halt if one program is replaced with a program that simply plays according to its minimax strategy, and so individually rational payoffs is a necessary condition for equilibria of such programs. ϵGroundedπBots halt output independently against each other (see Appendix H, arXiv:2412.14570).

We obtain the following negative result. An equivalent result expressed explicitly in terms of payoffs may be found in Appendix H, arXiv:2412.14570.

Theorem 8. *Consider games of the following form, given $\mathbb{N}$-valued random variables $T_1, \ldots T_n$: Each player i observes T_i, sampled independently from other players, and for each $t < T_i$ sees each other player j's policy conditional on that player observing $T_j = t$, and then chooses a strategy.*

If an equilibrium with payoffs $v_{1:n}$ is attained by simulationist programs that halt output independently, there must exist a policy profile $\tau_{1:n}$ in a game of the above form attaining payoffs $v_{1:n}$, with the following property: for all i, all possible deviations of the form 'play according to τ_i if

$T_i < t_0$, else play according to d_i' for some fixed strategy d_i attain at most as much utility against τ_{-i} as does τ_i.

The idea behind this result is as follows: for any profile of simulationist programs $p_{1:n}$, we may effectively define a 'time step' T_i for each program p_i, depending on the random sequence, as the maximum depth of the simulation tree with input programs p_{-i}. We then consider programs p_i' that look at the 'time step' T_i, and deviate if it is sufficiently high (at least t_0, say). We can show that the other programs only observe this deviation when their own time step is in turn high enough (at least $t_0 + 1$). This result then also provides additional justification for the idea of a simulated repeated game, since it shows that all simulationist programs may be in some sense viewed as being on a particular (random) time step and only being able to observe deviations occurring on earlier time steps.

From this result, we obtain that in the pirate example (Example 2) the action profile (C, C, C), cannot be attained in a simulationist uncorrelated program equilibrium (see Example 7, below). Thus, the folk theorem does not hold for simulationist programs in the uncorrelated setting. We further discuss which equilibria may be attained in Appendix H, in the version of this paper at arXiv:2412.14570.

Example 7. Recall the game between the pirates from Example 2. Can the pirates obtain pure cooperation in a simulationist equilibrium? We will apply Theorem 8 to show that the answer is no. Consider any choice of random variables T_1, T_2, T_3 for the time steps of the three players. Let t_0 be maximal such that $\mathbb{P}(T_1 \geq t_0) \geq \frac{3}{4}$, and suppose pirate 1 defects whenever she observes $T_1 \geq t_0$. There are two potentially optimal punishment strategies: punishing with D, and punishing with L (cf. Appendix L.2, arXiv:2412.14570). A D punishment strategy cannot work here, since even if both other players were to always play D, player 1 would still receive a utility of $\frac{3}{4} \times 14 > 10$. So we may assume that the other players play L when they observe player 1 defect. Now, player 2 observes player 1 defect with probability $\mathbb{P}(T_2 \geq t_0 + 1)$. Even if player 3 were to always observe the defection, for the punishment to be effective, we then need $4\mathbb{P}(T_1 \geq t_0)(1 - \mathbb{P}(T_2 \geq t_0 + 1)) - \mathbb{P}(T_2 \geq t_0 + 1) \leq 0$, i.e. $\mathbb{P}(T_2 \geq t_0 + 1) \geq \frac{4\mathbb{P}(T_1 \geq t_0)}{4\mathbb{P}(T_1 \geq t_0)+1} \geq \frac{3}{4}$. Then instead consider the case where pirate 2 defects whenever he observes $T_2 \geq t_0 + 1$. Pirate 1 sees this defection with probability $\mathbb{P}(T_1 \geq t_0 + 2)$, and we may obtain by similar argument that we then need $\mathbb{P}(T_1 \geq t_0 + 2) \geq \frac{4\mathbb{P}(T_2 \geq t_0+1)}{4\mathbb{P}(T_2 \geq t_0+1)+1} \geq \frac{3}{4}$. But this contradicts the definition of t_0.

6 Related Work

Our work is most closely related to papers on program equilibrium, especially Oesterheld (2019), to which we have related our ideas throughout the paper. DiGiovanni and Clifton (2023) use a variant of ϵGroundedFairBot to implement mutual commitments to disclose information. Their extension addresses similar limitations of Oesterheld's program as ours (failures to halt when simulating multiple opponents). Like the present paper, their main idea is based on correlated halting, somewhat similarly to our Section 3, but via access to a shared simulation device. They prove a variant of the full folk theorem for their setting. However, their approach differs from our correlated ϵGroundedπBots in important ways. For example, they assume that programs directly observe each others' randomisation probabilities, and their programs only allow for a restricted range of policies. We give a detailed comparison in Appendix C, arXiv:2412.14570.

A recent line of papers has studied games in which players are specifically given the ability to simulate each other. Most relevantly, Kovarik, Oesterheld, and Conitzer (2024) give players a joint simulation device, which allows simulating an interaction between the players. With probability $1 - \epsilon$, the simulated players in turn have access to a joint simulation device, and so on. Very roughly, our correlated ϵGroundedπBots could be viewed as decentralised implementations of such joint simulation. Like our ϵGroundedπBots, joint simulation devices enable a full folk theorem. Kovařík, Oesterheld, and Conitzer (2023) study games in which only one player is able to simulate the other at some cost, which is technically much less related.

7 Conclusion

The concept of program equilibrium promises to enable one-shot cooperation among agents whose code is legible to each other. Among several approaches to designing bots that play program equilibria, Oesterheld's (2019) ϵGroundedπBot stands out for its simplicity and robustness, requiring only simulation of the other program. Nevertheless, the original design of has several limitations, facing challenges in generalising to settings with more than two players, and falling short of the full folk theorem even with two players.

In this paper, we considered how ϵGroundedπBot might be generalised to address these issues. The key insight was to have a program determine its own 'time step' first and then manipulate simulated programs' sources of randomness so that they must be on earlier time steps. With this idea, we first showed that if the programs have access to shared randomness, we obtain a full folk theorem. We also characterised the equilibria that can be obtained without shared randomness, but these fall short of the full folk theorem. We showed one could do a bit better with more general 'simulationist' programs, but these still fall short of the full folk theorem as well.

Acknowledgments

We are grateful to the anonymous AAAI reviewers for helpful comments that improved the exposition of this paper. We also thank the Cooperative AI Foundation, Polaris Ventures (formerly the Center for Emerging Risk Research) and Jaan Tallinn's donor-advised fund at Founders Pledge for financial support. Caspar Oesterheld was supported by an FLI PhD fellowship. Emery Cooper began work on this project while working for the Center on Long-Term Risk.

References

Abelson, H.; and Sussman, G. J. 1996. *Structure and interpretation of computer programs*. The MIT Press, 2 edition.

Bárász, M.; Christiano, P. F.; Fallenstein, B.; Herreshoff, M.; LaVictoire, P.; and Yudkowsky, E. 2014. Robust Cooperation in the Prisoner's Dilemma: Program Equilibrium via Provability Logic. *CoRR*, abs/1401.5577.

Berg, J.; Dickhaut, J.; and McCabe, K. 1995. Trust, Reciprocity, and Social History. *Games and Economic Behavior*, 10: 122–142.

Conitzer, V.; and Oesterheld, C. 2023. Foundations of Cooperative AI. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(13): 15359–15367.

Critch, A. 2019. A Parametric, Resource-Bounded Generalization of Löb's Theorem, and a Robust Cooperation Criterion for Open-Source Game Theory. *Journal of Symbolic Logic*, 84(4): 1368–1381.

Critch, A.; Dennis, M.; and Russell, S. 2022. Cooperative and uncooperative institution designs: Surprises and problems in open-source game theory. arXiv:2208.07006.

DiGiovanni, A.; and Clifton, J. 2023. Commitment games with conditional information disclosure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 5616–5623.

Howard, J. V. 1988. Cooperation in the Prisoner's Dilemma. *Theory and Decision*, 24: 203–213.

Knuth, D. E. 1997. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Reading, Mass.: Addison-Wesley, third edition. ISBN 0201896834 9780201896831.

Kovařík, V.; Oesterheld, C.; and Conitzer, V. 2023. Game theory with simulation of other players. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, 2800–2807.

Kovarik, V.; Oesterheld, C.; and Conitzer, V. 2024. Recursive Joint Simulation in Games. *arXiv preprint arXiv:2402.08128*.

McAfee, R. P. 1984. Effective Computability in Economic Decisions.

Michie, D. 1968. "Memo" functions and machine learning. *Nature*, 218(5136): 19–22.

Norvig, P. 1991. Techniques for automatic memoization with applications to context-free parsing. *Computational Linguistics*, 17(1): 91–98.

Oesterheld, C. 2019. Robust program equilibrium. *Theory and Decision*, 86(1): 143–159.

Oesterheld, C. 2022. A Note on the Compatibility of Different Robust Program Equilibria of the Prisoner's Dilemma. arXiv:2211.05057.

Osborne, M. J. 2004. *An Introduction to Game Theory*. New York: Oxford University Press.

Rubinstein, A. 1998. *Modeling Bounded Rationality*. Zeuthen Lecture Book Series. The MIT Press.

Tennenholtz, M. 2004. Program equilibrium. *Games and Economic Behavior*, 49(2): 363–373.