
redscience Documentation

Release 0.0.1

Chris Santos-Lang

May 02, 2023

CONTENTS:

1	Use Cases	3
1.1	Anticipate security exploits	3
1.2	Benchmark social designs	4
1.3	Discover new dimensions of intelligence	5
1.4	Elevate reality above experimentation	6
1.5	Empower students of social science and computer science	6
2	How to Play	7
2.1	Playground Pages	8
2.2	Player Pages	9
2.3	Game Pages	10
2.4	Creating Games	12
2.5	For Trainers: Creating AI	14
2.6	Tournament Pages	16
3	Curriculum	19
3.1	1.0 Command Line Tic-Tac-Toe	20
3.2	How to Set-up Your Development Environment	21
3.3	How to Use GitHub from Colab	24
3.4	How to Use the redscience Ontology (const Module)	24
3.5	How to Internationalize redscience	24
3.6	How to Debug in redscience	24
3.7	How to Use the redscience Software Architecture	25
3.8	How to Compare Coding Options (and Python refresher)	25
3.9	Versions	25
4	Code	91
4.1	babelwrap module	91
4.2	category module	96
4.3	const module	101
	Python Module Index	117
	Index	119

(Also available as [PDF](#))

Much as [Fold-It](#) and [BridgeDesigner](#) offer simulations to test new designs for drugs and bridges, redscience offers simulations to test designs for personality, team composition, governance, and ethics. Any social interaction can be modeled with game theory, so redscience lets users create and compare designs for “Olympics” and player tools (a.k.a. “teams” of “AI”). Design advances in redscience until no olympics can be conceived for which a more reliable tool can be conceived. It’s games studied rigorously enough to advance better social engineering (for both humans and AI).

redscience is [open-source](#) and accompanied by educational resources so that diverse social groups can make their own versions which they fully understand and control. It is as much a *curriculum* about coding as it is *code*.

redscience has not yet been released, but feel free to preview our plans by reading this documentation. If you want alerts when we release, you can follow our twitter account at [@redscience_ai](#) or join the [redscience-announce](#) [GoogleGroup](#)

1.1 Anticipate security exploits

Suppose you were a developer assigned to deploy a new AI into the real world, or a parent or teacher preparing someone to enter society, or you were preparing yourself for a career, career shift, or to be a spouse or parent. It would be malpractice not to take every reliable inexpensive opportunity to discern any exploits to which that AI or person would be vulnerable. In some countries, for example, caregivers are expected to test children's vision to determine whether they should be equipped with glasses.

Games are a time-tested method to develop or establish ability with logic and planning, and can also establish other kinds of ability, such as social and political skills, trend-spotting or setting, and innovation. The concept of “game” can be broad enough even to include parenting, since parents make moves, moves impact outcomes, and some outcomes are preferred over others. Real world parenting, stock trading, and warfare are too expensive and consequential to serve as training exercises, so you might prefer to say we “live” these games, rather than say we “play” them. However, because they *are* games, skills make a difference, and it would be ideal to test the skills relevant to parenting, stock trading, and warfare before “living” those games (or concurrently) to potentially improve future outcomes.

When the most comprehensive redscience Olympics is sufficiently comprehensive and its most skilled players are sufficiently skilled, then playing that Olympics against those players will test for the full range of decision-making vulnerabilities. Note that learning is not the only way that vulnerabilities can be overcome; sometimes skills are established via tools (as with glasses) or via collaboration. The purpose of a security audit is to discern which tools, collaborators, or learning could help:

1. Have the human *establish an account* (or, if testing AI, *build it* in redscience).
2. Identify the most comprehensive *Olympics* via the **Comparison Tab** and its reigning champions via its *leaderboard*. Have the player under investigation play the Olympics against the champions and the champions' “favorite” opponents. In cooperative games (e.g. nuclear disarmament) the “exploiter” is likely to be someone who is difficult to cooperate with (antisocial, dogmatic, unhelpfully biased), rather than a champion. They will be a “favorite” of champions because the ability to deal with exploiters is what sets champions apart in such games.
3. Look at the player's **Favoritism Tab** to identify opponents and events for which the player under investigation consistently underperforms other players of the same skill-level. Those are the vulnerabilities.
4. To understand each vulnerability, look at the favoritism stats of the exploiter to find other victims it exploits in the same way, then profile those victims as a group to identify common traits. For example, “offense-bias” (risk-proclivity) would be a trait common to the victims of casinos (and the top champions will employ casino strategies for certain games).
5. To map the domain of “safety”, browse the **Favoritism Tab** to find events in which no vulnerability manifests. What do such events have in common (and what real-world situations share that commonality)? For example, players who would be exploited by casinos can find situations that do not present the same danger.
6. (For humans) to characterize opportunities to extend safety, have the player under investigation play each unsafe event using the top AI as a *tool*. Feel the costs of tool use—not only do tools create dependence, but they can

infringe on autonomy, depending upon how you use them. For example, one relinquishes autonomy completely when delegating decision-making to AI. Test the various forms of tool use (i.e. review, debate, and delegation) to determine which are sufficient to neutralize the handicap. Which form does the player under investigation prefer (or does the player prefer to avoid certain unsafe situations altogether)?

Note that redscience provides a far more comprehensive security audit than ever seen before, since the final steps permit the auditor not only to establish ways to mitigate vulnerabilities, but also to appreciate and minimize the costs of mitigation.

Warning: Patterns in the ways you can be defeated in various games constitute private information (like personality test scores, standardized test scores, or the results of genetic tests), so use an account that cannot be traced to you whenever playing large numbers of games alone!

Note: “Personality” settings are made available in redscience only if games have been identified for which different settings are optimal. Individual humans who exhibit those traits would be vulnerable to some kind of exploit in some games, but would have a special knack for some other games (and the potential to protect a team against exploits in those other games). In other words, *the vulnerabilities you discover, will typically also be strengths* (in the right context). We frame this use-case in terms of security because safety can be so important, but this use-case is much more positive than it sounds.

1.2 Benchmark social designs

Suppose you were assembling a team of humans, or an AI, or a collaboration between humans and AI to play a real-world game, such as stocktrading, diplomacy, policymaking, to compete in business, or to address a problem or need. One approach is to try to copy whatever design is currently most successful (e.g. poach from successful teams and ask the poached employees to replicate what worked for them in the past). That approach is called “dogma”.

Dogma is sub-optimal if existing social designs are sub-optimal. Previous simulations find that existing social designs retard social progress by 3 to 25 times. This should not surprise us, since we can look back in history to find social designs that are considered barbaric today, even though they were the most successful of their age.

An alternative approach—a way to escape dogma—is to test alternative social designs via simulations before deploying them in real-life. To the extent that the most *comprehensive Olympics* in redscience is sufficiently comprehensive and its most skilled non-human players are sufficiently skilled, the alternative approach has been completed, and one would simply build real-world teams that match the team-sizes, personality ratios, curriculum ratios, and collaboration techniques of redscience champions.

If scriptures were a collection of dogmatic best-practices for social engineering, then platforms like redscience would replace scripture, but, unlike scripture, such platforms need not identify with any particular religion and they offer those who question their wisdom a procedure to challenge that wisdom. For example, if redscience’s top non-human champion was a team of AI that included an extreme personality which social engineers hesitated to include in real-world teams (e.g. as some social engineers have hesitated to include “feminine” personalities in certain leadership teams), then the engineers could challenge the wisdom of including that personality as follows:

1. *Clone* the top team to create a new one, and make the objectionable personality less extreme in the cloned member.
2. Run an Olympic *tournament* which includes both the parent and its modified clone. Does the modified clone outperform its parent? What kinds of real-world situations match the kinds of events on which the parent outperforms the clone (i.e. what specifically can we appreciate about the extreme personality)?

Science will not instantly discern all wisdom and completely displace all other sources of wisdom, but science can become useful to guide not only physical engineering and medicine but also to guide social engineering, and platforms like redscience can make science as accessible as scripture. For example, if we previously turned to scripture to validate our response to personality differences, redscience can displace scripture for that function (something previous science was not sufficiently accessible to do).

Note: The most comprehensive Olympics will include cooperative games (like the *Public Goods game*), alliance games (like *Risk*), deception games (like *Hide and Seek*), and probabilistic games (like *Poker*), as well as planning games (like *Chess*), so this approach hedges against the potential for any real-world game to shift in any of these directions. If we can limit the shifting of real-world games, then it may be appropriate in the procedures above to use Olympics that are not the most comprehensive.

1.3 Discover new dimensions of intelligence

Suppose you loved someone so much that you wanted to leave a valuable legacy to their children and to the generations that follow. More than build an empire that could be replaced, you want to advance the very standard of quality so that any replacement would build on your legacy. What advance of quality could be more enriching than the introduction of a new dimension of intelligence (e.g. granting a culture its first awareness of empathy, tool-use, exploration or other not-yet-named dimension of intelligence)?

Intelligence is measured in terms of the kinds of games which one being wins more than another, so each dimension of intelligence can be expressed as a set of games (e.g. empathy can be expressed as games in which empathic players have advantage, perhaps because those games require collaboration with players who have different skill-levels and norms). The most comprehensive *Olympics* would test every dimension of intelligence, so the legacy left by making the most comprehensive Olympics more comprehensive (while maintaining elementality) is like the legacy left by expanding the Periodic Table of the Elements:

1. Identify the most comprehensive Olympics via the **Comparison Tab**
2. Use the **Comparison Tab** on the events of that Olympics to identify an essential event in it, then fine-tune tools for that specific event (see *Benchmark social designs*).
3. Contrast *the best tools for that event* to the best tools for other events to understand which *tools' biases* are particularly advantageous for that event.
4. *Clone the event and tweak its design* to make those biases even more advantageous.
5. Use the **Comparison Tab** to confirm that swapping-in the new event makes the Olympics more comprehensive.

Note: This feature caters to a niche user group, since many people are too busy establishing their security to worry about their legacy. Other game platforms might be tempted to omit this feature and provide mere escapism, self-development, or advantage in winning real-world games. “What? Discover new dimensions of intelligence?” they might say, “Yeah, I’ll let someone else worry about that...”

1.4 Elevate reality above experimentation

Suppose our society were divided by competing systems of social norms. For example, the best strategy in the *Volunteer* game depends upon prevailing social norms which happen to correspond to the real-world norms of “turn-taking” vs “caste system” (which sometimes manifests as racial discrimination). One could benchmark those norms in redscience:

1. Copy the top-ranked *AI* for the *Volunteer game* to a new *Universe* (but *do not copy its curriculum*). Play a turn-taking strategy against it (i.e. “You volunteered last time, now it’s my turn now.”) and confirm that it learns to take turns. Make several copies of that *AI* in that *Universe*.
2. Similarly create a second private *Universe* in which you train all *AI* to play *Volunteer* via caste (i.e. “You volunteered last time, so that’s your social position, and I’ll keep the non-volunteer position.”).
3. Copy an *AI* from the turn-taking *Universe* to the caste *Universe* (retaining its turn-taking experience), and confirm that it switches to the caste strategy.
4. Copy an *AI* from the caste *Universe* to the turn-taking *Universe* (retaining its caste experience) and confirm that it switches to turn-taking.
5. In the public *Universe*, run a *Volunteer tournament* with equal numbers of players copied from the caste and turn-taking *Universes*. Which norm survives? Similarly test other population ratios to find the minimum ratio for the other norm to survive.
6. Observe how freedom to select social situations impacts norms by running tournaments where each reselection of players is composed of a player and their favorite opponent. Repeat the experiment where each reselection is composed of two random players plus the favorite opponent of the top-ranked player.

If we couldn’t run these experiments to our satisfaction in redscience, would we be doomed to spend our real lives serving as the subjects in such experiments (i.e. as pawns in a war between competing systems of norms)? It may be unlikely that *everyone* who runs such experiments will switch to whichever norm consistently wins, but the dignity of an informed loser is at least elevated compared to a pawn who never even tried the experiments.

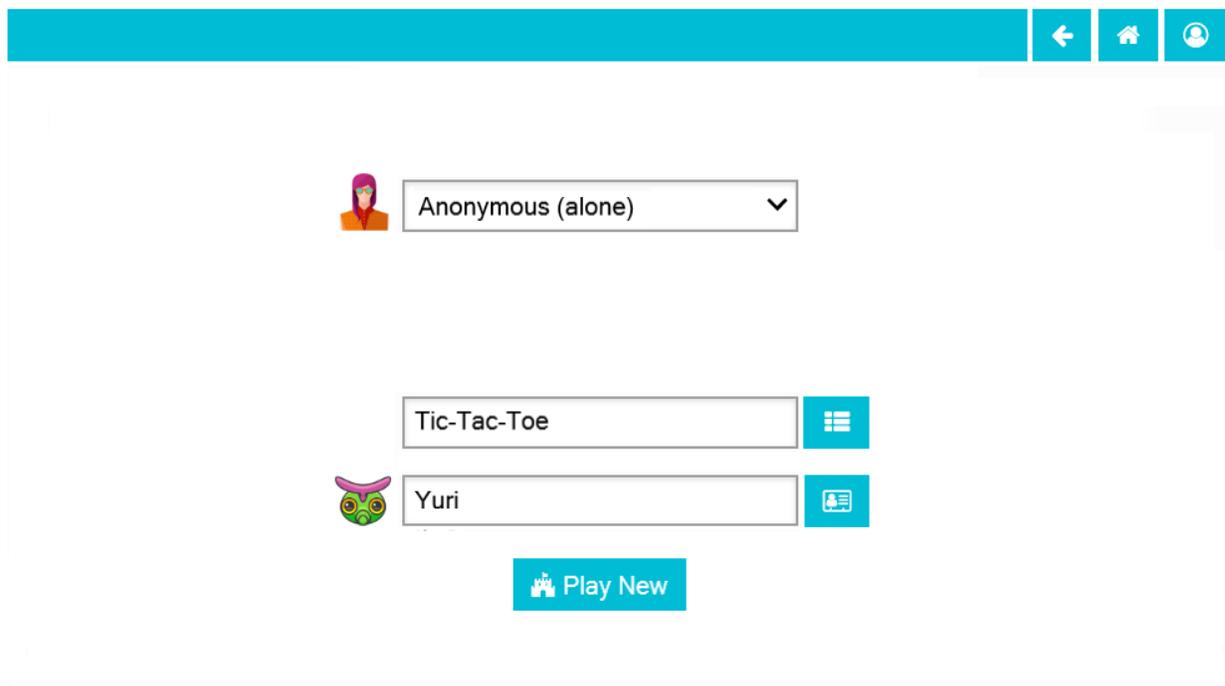
1.5 Empower students of social science and computer science

Suppose you were a social science teacher or computer science teacher. It’s one thing to expose students to new ideas, but another thing to empower students to test those ideas for themselves. Although redscience is designed to be accessible at the secondary-education level, it is just as relevant in post-secondary education.

- A social science teacher could assign students to *Benchmark social designs*, *Anticipate security exploits*, or *Elevate reality above experimentation*
- A computer science teacher could assign students to *Anticipate security exploits* (so they are aware of the security vulnerabilities of AI) and to *Build their own redscience*

HOW TO PLAY

If you just want to “kick the tires” then you do not need a user account.



© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

As an anonymous user, on the Home page, you can select a *game* and other *player(s)*. If you select a way to augment your intelligence (i.e. do not play “alone”), then you can also select your *tool*. Clicking the **Play New** button starts the *match*.

2.1 Playground Pages

The screenshot shows a Tic-Tac-Toe game interface. At the top, there is a blue navigation bar with icons for back, home, and profile. The game title "Tic-Tac-Toe" is centered. Below the title, the status "Stalemate draws; First 3-same-color-in-a-row wins" is displayed. On the left, Player 2 is shown with a green alien-like avatar, a score of 4, and the label "Poorer Novice". On the right, Player 1 is shown with a purple-haired woman avatar, a score of 5, and a rating of "27% (Subtle Common)". The text "Player 1 Moving" is also present. A "Black circle to" button is visible. The game board is a 3x3 grid. At the bottom right, there is a "Play New" button. A copyright notice at the bottom reads: "© Chris Santos-Lang, 2020. Avatar icons by Roundicons and Roundicons Freebies from www.flaticon.com".

On the playground page, the other *player(s)* display on the left (with their colors, if assigned). The system summarizes their play history by classifying their relative expertise and the degree to which they are “owed” favors. Such classification may be useful to support social strategies (but players are free to ignore the labels, if they wish).

The board(s) display in the center of the playground. The time bar below it indicates the amount of time left in the current turn. Depending upon how you choose to *augment your intelligence* (if at all), calculations may appear on the board along with suggested moves and justifications. Your augmenting tool will also eliminate the risk of running out of time by moving in your place if necessary.

Click the board to enter/change your move(s) when it is your turn. A player cannot move a placed piece if that player has an identical piece in reserve and cannot attempt to place a piece from reserves to a location that the player knows is occupied. Otherwise, each game has its own *rules*.

Your move selection displays in the upper-right with a button. Click the button to accept the move. In the lower-right, your reserves (if any) display, along with a private randomly-generated impulse (useful in strategies that require randomness). Hover over pieces in reserve to display rules for those shapes. Hover over patterned regions of the board to display special rules for those regions. In games with multiple phases, hover over the displayed rules on top to preview rules for future phases.

At the end of each match, the estimate of your skill-level for the *game* you played will be revised.

Note: Unlike with physical board games, different elements could be visible to different players in some games (e.g. *Hide and Seek*).

2.2 Player Pages

Even an anonymous user could help train artificial intelligence by playing against it. However, the biggest advances in social science will come from inventing *new personalities*, teams, and *games* that are in some way “better” than those previously known. Only logged-in users can be a *Creator*.

Click the **Login/Change User** button  in the upper right corner to login or create a new account.

Click the **Player** button  next to a player’s name to launch its page:

General Intelligence Games
←
🏠
👤



Dr. Falkner

Creator ▼

MyEmail@email.net

📄 Copy

Creations	Stats	Favoritism																													
			<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Type</th> <th>Universe</th> <th>Usage</th> <th>Age</th> </tr> </thead> <tbody> <tr> <td>Event</td> <td>n/a</td> <td>223%</td> <td>37d</td> </tr> <tr> <td>AI</td> <td>Public Universe</td> <td>54%</td> <td>55d</td> </tr> <tr> <td>Course</td> <td>n/a</td> <td>23%</td> <td>36d</td> </tr> <tr> <td>Event</td> <td>n/a</td> <td>8%</td> <td>66d</td> </tr> <tr> <td>AI</td> <td>Joshua League</td> <td>2%</td> <td>51d</td> </tr> <tr> <td>Persona</td> <td>Joshua League</td> <td>1%</td> <td>78d</td> </tr> </tbody> </table>	Type	Universe	Usage	Age	Event	n/a	223%	37d	AI	Public Universe	54%	55d	Course	n/a	23%	36d	Event	n/a	8%	66d	AI	Joshua League	2%	51d	Persona	Joshua League	1%	78d
Type	Universe	Usage	Age																												
Event	n/a	223%	37d																												
AI	Public Universe	54%	55d																												
Course	n/a	23%	36d																												
Event	n/a	8%	66d																												
AI	Joshua League	2%	51d																												
Persona	Joshua League	1%	78d																												
3P-Misere-Tic-Tac-Toe																															
Tron 																															
*Train1																															
Tic-Tac-Toe																															
Joshua 																															
Dr. Falkner 																															

© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

Click the avatar, name, and/or email address at the top of your player page to change it. Your email address is visible only to yourself and to *Admins*.

- The **Creations Tab** provides buttons to launch all players and games created by that player. The usage statistic gives the average popularity of each creation over its life.
- The **Stats Tab** provides buttons to graph the evolution of the player for various *games* and to challenge the player to new *matches*.
- The **Favoritism Tab** highlights other players who have historically exhibited special relationships with the player. It is useful to detect security vulnerabilities inherent to a specific AI (or human; yes, humans can have vulnerabilities too).

Logged-in users see a **Copy** button on game and player pages. You can create another *Persona* for yourself (and assign it a *Universe*) by clicking that button on the page of any human player.

A player can play only with other players from the same *Universe*, but each human player will automatically get a *Persona* in each *Universe* in which they are invited to play. Thus, you could establish private scoring among a group of friends by creating a *Persona* with its own private *Universe*, then inviting friends to play with it. Each human you invite could invite others, so invite only friends who agree about how that *Universe* is to be used.

2.3 Game Pages

Tic-Tac-Toe is one of many possible games, each with its own **Leaderboard** (for each *Universe*).



Click the button next to the name of any game to launch its page:

General Intelligence Games
←
🏠
👤

Tic-Tac-Toe

(Beginner-level event)

🔧 Created by Dr. Falken

Related Events:

☰ Achi

☰ Tapatan

☰ 3on5sq

Setup & Rules

Leaderboard

Tournaments

Compare

Lora (A)
+

Public Universe

⚖️
📈

1		Tron	2963 (3d)	+657	👤 👥	<input checked="" type="checkbox"/>
2		Dr. Falken (R)	2382 (114d)	+742	👤 👥+	
3		Dr.Falken (D)	2270 (157d)	+536	👤 👥	
⋮						
		Joshua	1938 (1d)	+177	👤 👥	<input checked="" type="checkbox"/>
		*Tic-Tac-Toe SP	1853 (48d)	+5	👤 👥	<input type="checkbox"/>
		Random	4	+0	👤 👥	<input checked="" type="checkbox"/>

© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

- The **Leaderboard Tab** provides buttons to see *player* stats, start new *matches*, and compare players.
- The **Tournaments Tab** provides buttons to see recent *tournaments*
- The **Compare Tab** provides buttons to compare games (and to design better *Olympics*)

If the game is an *Event*, then it will have a **Setup & Rules Tab** as shown above. If the game is an *Olympics*, then it will have an **Events Tab** instead with buttons to launch the page of each *Event*. Either tab will have a **Copy** button you can use to make a version with different *rules*.

2.4 Creating Games

Clicking the **Copy** button on the **Step & Rules Tab** or **Events Tab** of a *game* yields a new copy that can be altered (until first saved):

Game Factory

Setup & Rules

Save

More pieces than Chaos wins ▼ +

Less-than-most-committed-pieces folds ▼ 🗑️

4 pentagons, 4 Xs, 4 crosses, 4 triangles, and 4 stars in deck ▼

3 + Chaos ▼

6 shapes ▼

1 piece/turn ▼

1 board ▼

6 phases ▼

Assigned colors ▼

180 seconds/turn ▼

Stacks ▼

Cloaking obscures ▼

8 ▼ ▲

1 ▼ ▲

26 ▼ ▲

No powers ▼

▼

▼

Moves orthogal by stack ▼

Can capture ▼

Cover by rank ▼

Converts captive(s) ▼

Moves orth by full stack ▼

Can capture ▼

Cover by rank ▼

Converts captive(s) ▼

Moves orth by full stack ▼

Can capture ▼

Cover by rank ▼

Converts captive(s) ▼

Moves orth by full stack ▼

Can capture ▼

Cover by rank ▼

Converts captive(s) ▼

Moves orth by full stack ▼

Can capture ▼

Cover by rank ▼

Converts captive(s) ▼

Moves orth by full stack ▼

Phase 1: ▼

Sequential ▼

Movement Allowed ▼

Option to Pass ▼

🔒	?	0
🔒	+	0
🔒	△	0
🔒	☆	0
🔒	?	0
🔒	+	0
🔒	△	0
🔒	☆	0

26

🔒	?	0
🔒	+	0
🔒	△	0
🔒	☆	0
🔒	?	0
🔒	+	0
🔒	△	0
🔒	☆	0

© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from [www.flaticon.com](#)

2.4. Creating Games

13

In addition to specifying reserves and initial piece placement (which can include pieces dealt from a shuffled deck), the creator can designate spaces as exclusive to specific players or as “sticky” (no exit). Each phase can be *sequential*, *simultaneous*, or *single* (which is simultaneous, but last only one turn), can lock specific players, and can lock or cloak specific spaces. Each shape of piece can have its own power and way of moving (or not).

A few options deserve special explanation:

- Instead of *2 Player*, *3 Player* or *4 Player*, a game can be *Partners* (odd vs. even), *2 + Chaos* or *3 + Chaos*. “Chaos” is a randomized common-enemy. *4 Player* games can have an “Overachiever(s) disqualified” rule that disqualifies the first player(s) to qualify as winners from actually winning.
- “Fold” rules are checked at the end of each phase. If a player other than “Chaos” folds, then that player is locked for the rest of the match (but remains eligible to win).
- Instead of the *Hash* used in Tic-Tac-Toe, the board can be *Squares* of various dimension or *Stacks* for which pieces can be placed or moved only to/from the tops. Pieces that move *by stack* bring all pieces above them along for the ride. Pieces that move *by full stack* always bring the entire stack.
- Instead of *destroy*, the effect of capture can be to *convert* which changes the color of captives to match their captor, to *reincarnate* which converts and moves the captured pieces to the captor’s reserves, or to *reincarnate (x2)* which reincarnates but multiplies the captives.
- Pieces with the *chain-jump* power can make multiple jumps in a row in a single turn. When pieces with the *cover by rank* power cover (or are covered by) other pieces, the piece with the highest rank captures the rest. All are destroyed if the ranks are equal. Pentagon outranks star which outranks cross which outranks X which outranks triangle which outranks circle (but circle outranks pentagon and only pentagon). When pieces that move *by stack* cover by rank, the rank is a combined rank of all pieces of that color in the final stack plus any communal pieces in the final stack (ignoring any pieces of the lowest possible rank). Highest-5-of-a-shape outranks highest-4-of-a-shape which outranks highest-5-straight which outranks highest-full-house (i.e. three-of-a-shape-plus-pair) which outranks highest-4-straight which outranks highest-3-of-a-shape which outranks highest-2-pair which outranks highest-3-straight which outranks highest-pair which outranks highest-2-straight which outranks highest-singleton.

All social behavior can be modelled via games, and special effort has been made to ensure that users can construct games of each kind in economic game theory (e.g. *Public Goods*, *Prisoner Dilemma*, *Stag Hunt*, *Ultimatum*, *Volunteer*, *Battle of the Sexes*, *Dictator*, *Trust*, *Beer-Quiche*, etc.). Please let us know if there is a kind of game that cannot be constructed.

2.5 For Trainers: Creating AI

Any user can use any *AI*, *Team*, or *Corp* as a *tool to help them play*, but only Trainers see a **Copy** button on the *page* of each such tool to create a duplicate which that Trainer can adjust (until it is saved). Finding/refining the best tools may be the most powerful way to raise one’s *rankings*.

Experimenting Bot
←
🏠
👤



🧪 Created by Dr. Faulkner

Curriculum

👤
Yori

▼
🗑️

▼

▼

415	▲▼	Master: 3on5sq
328	▲▼	Master: Draughts
578	▲▼	Master: Tic-Tac-Toe
202	▲▼	3P-Misere-Tic-Tac-Toe
9692	▲▼	Connect4

▼

▼

📊
⚖️

Alpha L1 Epsilon

0.0001

0.2

0.1

Offense Tactical Faith Introvert Empath Curious

0.3

0.5

0.5

0.5

0.7

0.3

▼

© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

Many player settings, including *Type* and *Universe*, cannot be changed once saved. If you want to change those settings, just make a new copy! If you ever want a copy of an earlier version of an *AI*, you can “fork” it from a specified earlier timestamp.

AI players have a **Curriculum Tab** instead of a Creations Tab. Even after an *AI* is saved, its creator can add recorded matches to its curriculum by specifying parts of tournaments or of other players’ histories to be studied. Its creator can also set the *AI* to automatically learn from its own experiences. Until the *AI* is saved, its Trainer can also set its learning algorithm, parameters for that algorithm, and the following (or the Trainer can set them to auto-tune to a given

curriculum or game):

Offense (opposite: Defense): 1.0 means maximize wins (considers draw as bad as loss); 0.0 means minimize losses (considers draw as good as win); 0.5 values draw halfway between win and loss

Tactical (opposite: Strategic): 1.0 means prioritize the current game; 0.0 means maximize ones own skill-rating. Tactical greater than Offense means never sacrifice a current win to seek future wins; Tactical greater than $(1 - \text{Offense})$ means never take a loss to seek future wins

Faith (opposite: Skeptical): 1.0 means confidence in one's data never decays; 0.0 means confidence expires instantly; between 0.0 and 1.0, confidence in ones data depends upon the age of that data

Introvert (opposite: Extrovert): 1.0 means practice on simulations as much as possible before acting; 0.0 means learn only via action (so as not to become an "echo chamber")

Empath (opposite: Projective): 1.0 means try to predict others' moves based on their stats and recent behavior; 0.0 means expect others to do whatever you would do in their situation

Curious (opposite: Practical): 1.0 means to focus practice towards unpracticed scenarios; 0.0 means to focus practice towards expected scenarios (see Empath)

If the player is a *Team* or *Corp*, then the player page will have a **Members Tab** instead of a Curriculum Tab or Creations Tab. Its creator can add *AI* to the members, and can delete any members that have become obsolete. Each individual *AI* has biases, but *Teams* and *Corps* leverage diversity of biases. Each *Team* fields a single member for each event (i.e. specialists), but multiple members of a *Corp* collaborate on each decision (via various forms of government).

2.6 Tournament Pages

Tournaments are for non-human players (they involve many more *matches* than human stamina would permit). They are experiments to confirm which *AI*, *Team*, or *Corp* is the best at a specific event or in general (i.e. for the most comprehensive *Olympics*).

Click on a **Tournament** button  to launch the page for that tournament:

General Intelligence Games



41st Beryllium Olympiad

Olympics 2020-12-15 14:05:01

*The Beryllium Heptathlon			1000 Games	23,536 Experiments	42:49:39.75					
			Win	ACC	F1	TCH	EMP			
Ravenclaw			25%	0.98	0.95	0.78	0.54			
Enterprise			22%	0.97	0.94	0.43	0.75			
Fantastic5			16%	0.96	0.92	0.35	0.53			
OMG			14%	0.96	0.90	0.63	0.16			
Oceans14			9%	0.96	0.86	0.53	0.65			
Cloud7			6%	0.95	0.84	0.32	0.42			
Legion			4%	0.92	0.79	0.28	0.48			
Draw			2%							

© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

Olympics are games composed of diverse events which no competitor has ever encountered before. Rather than play an Olympic event directly, a player must create a copy of itself to play (although the outcomes impact the skill-level estimate of the original). The copy is archived never to play again—that prevents players from mastering Olympic events in advance. An Olympics is a race to develop new mastery, so Olympic champions are the players most likely to succeed when facing novel situations (or in competitions won by changing the rules).

Trainers can create a new tournament from a *Leaderboard* or *Favoritism Tab*, using the checkboxes to select players. Special security is required because a very complicated tournament could take days to run.

CURRICULUM

Not only is the `code` of redscience opensource, but a number of pre-code resources are provided openly as well, and their openness is tested by forming all of the resources into a curriculum and testing the ability of diverse students (armed with the curriculum) to rebuild redscience as they see fit.

Software development is typically divided into jobs or tasks:

Business Analysis/Ontology: Anticipate what users will desire as the platform evolves. Decide what to calculate/monitor, and how to define entities (e.g. games, users, bots, etc). Document the workflows to be tested.

UI Design: Anticipate the diversity of direct users (differences in training, devices, interests, etc.) Choose phrasings, colors, shapes/fonts, sizes, patterns, widgets, layouts and navigation. Document mock-ups and the wording of messages.

Database Architecture: Anticipate potential reuse of information. Choose names for database tables and columns, and document their definitions and indexes.

Software Architecture: Anticipate code writing/fixing/enhancing/evolving. Establish code languages and tools for editing, debugging, deploying, and versioning code. Divide the planned code into logical (reusable) units, and create technical documentation and automated tests for each.

Pseudocoding: Rewrite the documentation of each code unit as ordered “To do” comments. Identify code units to be employed from existing code libraries.

Coding: Write, test, debug, and deploy the code.

This curriculum assumes that you might wish only to code. You can expect the curriculum to provide the completed deliverables for all other tasks. The “How to” articles in this curriculum will walk you through how to use those deliverables.

In typical development, coding reveals opportunities to improve the other deliverables. You are welcome to modify any deliverable (as one might do in typical development), or entirely rebuild it from scratch. Be aware, however, that deliverables encountered in typical development have had less opportunity for refinement, since typical projects have never been coded before. Typical development might be more “messy” than what you would experience while following this curriculum.

Pseudocoding is usually done by coders. It is provided in this project because we assume that you want to be told about (and get practice with) widely-used existing code libraries and code structures. You don’t have to follow the suggestions of the pseudocode. If you have a better idea, please share it with us, so we can suggest it to future coders.

This project is divided into Versions, each building upon the previous. The *Versions* section of this curriculum provides *Requirements*, a *Testing Plan*, and *Potential Mockups* for each Version. Those documents communicate the scope of the corresponding Version.

This curriculum also includes a number of “How to” articles which step through the beginning of Version 1.0. The curriculum will provide plenty of opportunity to get creative and prove your coding skills later, but our first priority is to make sure we are on the same page, so feel free to follow the “How to” articles to the letter.

Start by reviewing the *Requirements*, *Testing Plan*, and *Potential Mockups* for (*1.0 Command Line Tic-Tac-Toe*), then begin with *How to Set-up Your Development Environment*.

3.1 1.0 Command Line Tic-Tac-Toe

The *Curriculum* for building redscience divides the task into *Versions*, of which Command Line Tic-Tac-Toe is the first. The following plan serves as an example of the kind of plan you can expect for other versions.

Much of the curriculum walks students through actually implementing parts of this plan. It is possible to skip to the action having never read the following higher-level plans that the software architecture is designed to fulfill (just like it is possible to modify open source code without knowing why it was built the way it was), but having a higher-level understanding of what you are working on can make you more productive as an innovator. Therefore, it is recommended that you read the following (abstract as it may be) before coding:

3.1.1 Requirements

(Various words are bolded in this first set of requirements to give a sense of what an ontology would have to include.)

Write a program that allows two **humans** (sharing the same keyboard) to play **Tic-Tac-Toe** at the **command prompt**, with **options** at any point to **quit**, to **undo the last move**, or to **start a new game**. Randomize the order of the **players** at the beginning of each new **game**. A player cannot attempt to place a **piece** from **reserves** to **coordinates** that the player knows are occupied. If a player has only one legal **move**, then choose it automatically (so the other player doesn't have to wait). If a player has no legal move, automatically **pass**. Detect when a player repeats the exact same move facing the exact same **board**—this will be called “stalemate” and happens in Tic-Tac-Toe only when the board is full (so both players automatically pass). The **stalemate rule** determines the outcome if the final phase of a game ends in stalemate. Each game may also have one or more **rules** that are checked at the end of each **turn** (e.g. “**First 3-same-color-in-a-row wins**”).

The program should contain the following **rules** as a **constant**, and the rules should determine how **play** proceeds:

- Played on **hash (3,3)**
- **2-Player**
- **Assigned Colors**
- **Circle: 5 black** and **4 white** start in reserve
- First 3-same-color-in-a-row wins
- Stalemate **draws**

In general, a game move is either a pass, a **request to call it a draw, agreement/rejection** of such a request, a **jump** from **one spot** to another, or a **placement** from reserves of a given **shape** and **color** to a spot on the board. The rules of Tic-Tac-Toe permit players to choose only moves of the last type and to only specify “to” coordinates, since all pieces in that player's reserves will be of the same shape and color, but design your code to evolve support for other games (e.g. other-dimensional boards, other kind of pieces/cards, etc).

3.1.2 Acceptance Test Plan

Test that the program works. Confirm that it does not permit illegal moves, and that it recognizes draws and all kinds of wins (e.g. horizontal, vertical, upward-slanting-diagonal, and downward-slanting-diagonal). After finishing a game, confirm that you can roll back the choices using undo. Also confirm that you can end the game or restart at any time.

3.1.3 Potential Mockups

To start from command line:

```
pip install redscience
redscience tic-tac-toe
```

3.2 How to Set-up Your Development Environment

Now that you have read the Requirements, Testing Plan, and Potential Mockups for [Command Line Tic-Tic-Toe](#), let's begin to code it together. The first thing you will need to do—only for the first Release—is to set-up your development environment. You are welcome to use any environment you like, but software architecture includes planning for how code will be versioned, deployed, and debugged, so we have a plan for you. A great architecture might also include a coding style guide like the one at <https://google.github.io/styleguide/pyguide.html>. We'll just point you to that one.

Here's a summary of our plan:

1. Set up a **GitHub** account, and clone the redscience template to get the initial set-up.
2. Set up a **Google** account.
3. In **Google Colab**, mount your Google Drive, then clone your GitHub repository (“repo”) to your Google Drive where you can edit and run your code in the cloud.
4. Test your revisions with pytest and linters, then commit them to GitHub
5. (Optional) Set up a **ReadTheDocs** account and link it to your GitHub repo to automatically generate documentation.
6. (Optional) Set up a **PyPI** account and link it to your GitHub repo to share your code with the world.

This article will walk you through step 1, explain what you have inherited by cloning the template, then point you to other resources that cover later steps.

3.2.1 How to Choose Development Tools

You don't have to use GitHub, Colab, ReadTheDocs, and PyPI. This section highlights some of the features you might look for in replacements. Feel free to skip this section if you simply want to adopt the provided architecture.

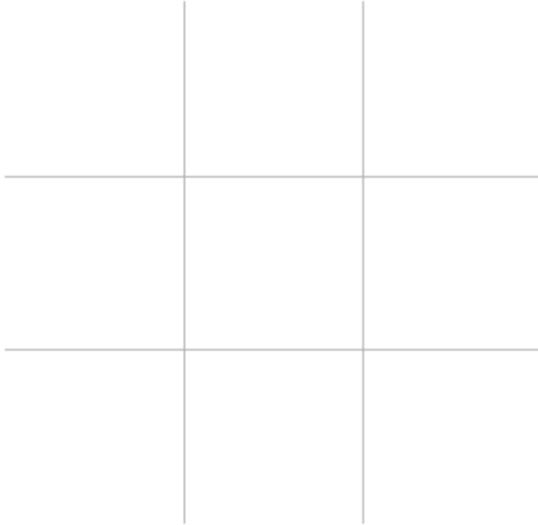
You are about to build something, and you will master it in the process. You might expect to discard your work at that point and take away only memories. But it is also possible that you might want to continue evolving what you built, maybe even transform it into something with a very different purpose. Just in case that happens, the provided architecture includes best practices for general development.

Note that evolving a competitor for redscience is not what we have in mind by “continue evolving what you built”. If you want to improve redscience, then please offer a pull request to it so everyone can enjoy your innovation. If your improvement is rejected, then it would be better to fork redscience (and poach from its community) than to rebuild it completely. If you want to customize redscience, then merely wrap it, so your customized version will automatically enjoy any future improvements to redscience. What we have in mind by “continue evolving what you built” is to copy

Tic-Tac-Toe

Rules: First 3-same-color-in-a-row wins and stalemate draws

Players: Player 1 (Black) and Player 2 (White)

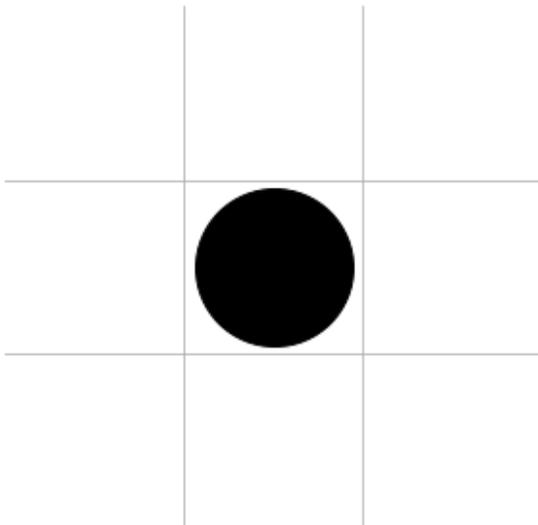


Player 1

What's your move? ('#,#' or q/z/n for quit/undo/new game)

2,2

Black circle to (2,2)



Player 2

What's your move? ('#,#' or q/z/n for quit/undo/new game)

Fig. 1: Each move adds a new section to the bottom...

patterns or chunks of what you built to your next project. The best kind of code to reuse is (a) code you understand (which you will, since you built it), and (b) code built to last.

If you want to build software to last, you should expect it to need a community to provide security updates as security vulnerabilities become discovered, compatibility updates as dependencies evolve (e.g. operating systems, storage systems, and browsers), and new features as user expectations shift (e.g. to different devices). At first, you might be the only member of your community, or every member might be an employee of a single business, government or religion, but building code to last means designing it to extend beyond that. Even if many people want to participate in a such a community, the community will need the following:

- Development environments (including hardware),
- Means to transfer knowledge to new members (i.e. documentation/training),
- Means to set priorities/resolve disagreements (i.e. management),
- Means to control quality (i.e. testing), and
- Means to connect with the needs of users (i.e. sales/implementation and issue reporting framework)

Not only must your software be able to evolve over time, so must these resources. The hardware your community will need to maintain your software 20 years from now might be very different from the hardware required today. This is where **Colab** comes in. Rather than plan to buy new hardware every so often to support your software, you can develop “in the cloud” and let the cloud provider handle upgrading the hardware. It is important that whichever provider you choose is (1) accessible to your community, (2) provides reliable quality hardware, and (3) is compatible with the rest of your development environment. Colab is accessible for free (up to certain limits) to anyone with a web browser (e.g. cell phone), and it provides very fast GPUs. Most of the Jupyter Notebooks development environment on Colab can be made available elsewhere—it provides as many cells as you like in which to experiment—but Google added autocomplete and help windows that are noteworthy features.

Similarly, your community 20 years from now might need new ways to transfer knowledge about how to maintain it. It helps that the Python language is relatively easy to read. Yet, the current state-of-the-art in knowledge-transfer also includes providing documentation (both in code and out of code). The current Python standard is to write documentation in reStructuredText (RST) built with Sphinx (via **ReadTheDocs**). This is not effortless—your community would need to abide documentation standards and would need educators to write everything beyond the API—but automation saves effort on converting documentation between forms to permit browsing, searching, and printing (e.g. to html and pdf). You could give your software a fancy website without RST (but it would still take effort), and RST can’t embed functioning Jupyter Notebooks (like it can videos), but would your coders and writers end-up at each other’s throats if you didn’t have a searchable instruction manual that automatically-updates the API sections whenever the code changes? Linking to ReadTheDocs is an optional step in this curriculum, but the architecture models corresponding documentation standards (to keep that documentation option viable).

Your community 20 years from now also might not be able to set priorities and resolve disagreements the way it can now. Benevolent dictators do not last forever. Furthermore, better processes for achieving consensus may evolve over time, and you will automatically benefit from those innovations if you use a leading code-management system. ReadTheDocs currently integrates with GitHub, BitBucket, and GitLab, of which **GitHub** is currently the most popular. All three innovate processes to manage code communally, including ways to report issues. One of their most important innovations is the “Pull request” (or “Merge request”) which overcomes natural-resistance-to-change by empowering a developer to propose a code change in a way that makes it easy for others to analyze and test (even automatically), try, debate, and accept. GitHub can automatically generate pull requests whenever dependencies upgrade and automatically inform you if proposed changes would break functionality or introduce security vulnerabilities.

Finally, although code-management systems provide means to benefit from a community, they don’t necessarily attract that community. The ways in which people form communities may change over the next 20 years, but bulk-searches through “what’s out there” will likely be part of the mix. **PyPI** is python’s standard index, so it is a natural place for a bulk-search. Users may not need to use PyPI to install your software if they can run it “in the Cloud” via Colab, but being indexed in PyPI may be crucial to being discoverable. Linking to PyPI is an optional step in this curriculum, but the package is set up to permit it.

3.2.2 How to Set-up on GitHub

First, if you don't already have a GitHub account, follow [the GitHub instructions](#) to get one.

If this is a group project with an existing repository, then fork it. If you want to start a new code base, then the plan is to start by clicking "Use this template" at <https://github.com/ChrisSantosLang/redscience> (instructions from GitHub [here](#))

[Describe folder structure and explain configuration files here]

3.2.3 How to Set-up Colab

At this point, you should have your own GitHub repo for this project. If you don't already have a Google account, follow [the Google instructions](#) to get one, then move forward to How to Develop in Colab with GitHub.

3.3 How to Use GitHub from Colab

(Under construction)

This will be a Colab workbook with scripts to get and commit code

3.4 How to Use the redscience Ontology (const Module)

(Under construction)

This will be a Colab workbook demonstrating use of the const Module

3.5 How to Internationalize redscience

(Under construction)

This will explain how to create mo and po files (a reference most students will prefer to skip)

3.6 How to Debug in redscience

(Under construction)

This will describe anticipated strategies for finding and resolving bugs

3.7 How to Use the redscience Software Architecture

(Under construction)

This will step students through completing the first few milestones of version 1.0

3.8 How to Compare Coding Options (and Python refresher)

(Under construction)

This will be a Colab notebook demonstrating procedural, functional, and recursive approaches to the cartc function and how to compare them via profiling, performance measurement, time complexity estimation, and radon code quality metrics. It is an optional “side trip” for students who want to learn to code from examples, or who want to review their coding knowledge.

3.9 Versions

The project for this curriculum is divided into versions, each of which builds upon the previous. You can stop at any point.

3.9.1 1.1 GUI Tic-Tac-Toe

Requirements

Adapt the Command Line Tic-Tac-Toe to work entirely with the mouse. Require separate clicks to select a move (at which point the move displays highlighted as “proposed”), and to accept it. Add a count-down timer that resets to one minute every time the turn changes and automatically accepts the proposed move when the minute is up. Any player who does not select their move before the timer runs out, or who leaves the playground before the game is complete, loses. In addition to displaying the name of the game being played, display its rules (piece-specific rules like ability to move or capture can be displayed via tooltip).

Impulses

Randomly assign each player two private impulses: Subtle vs. Basic (33.333% frequency) and Rare vs Common (20% frequency). Having private impulses provides strategic advantage in games for which opponents could exploit predictability; we level the playing field by making them available to all players.

Acceptance Test Plan

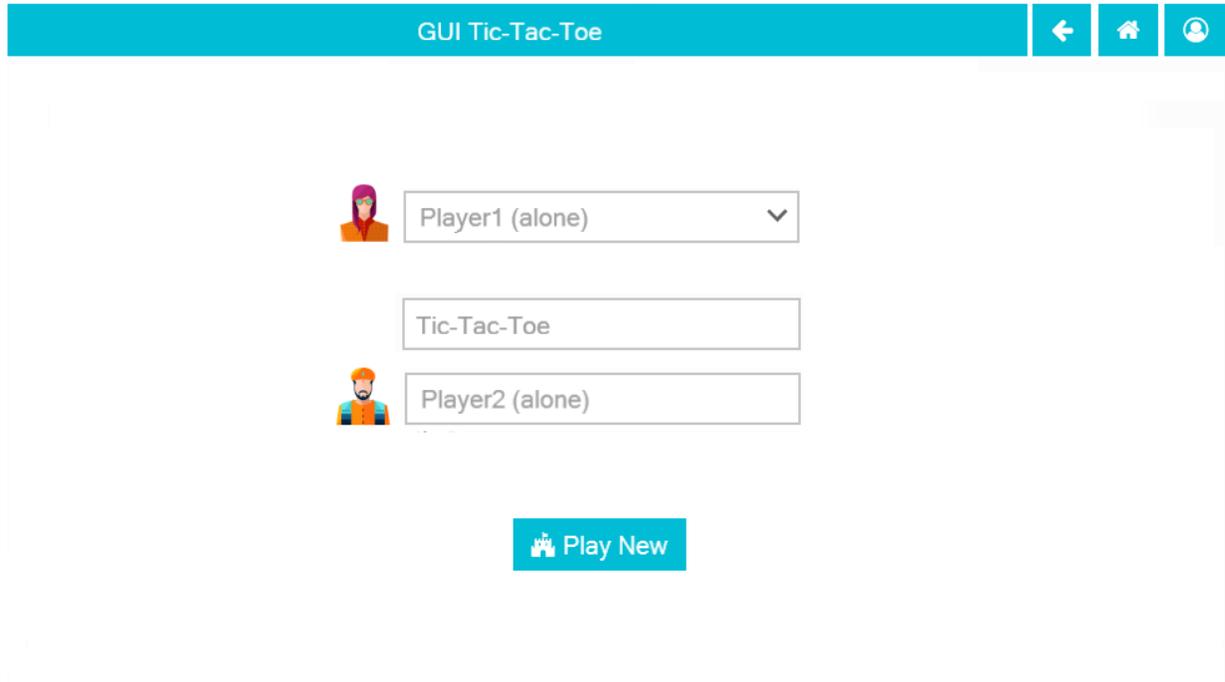
Test each of the clickable elements and test that it displays appropriate errors for invalid entries

Potential Mockups

To start from command line:

```
redscience
```

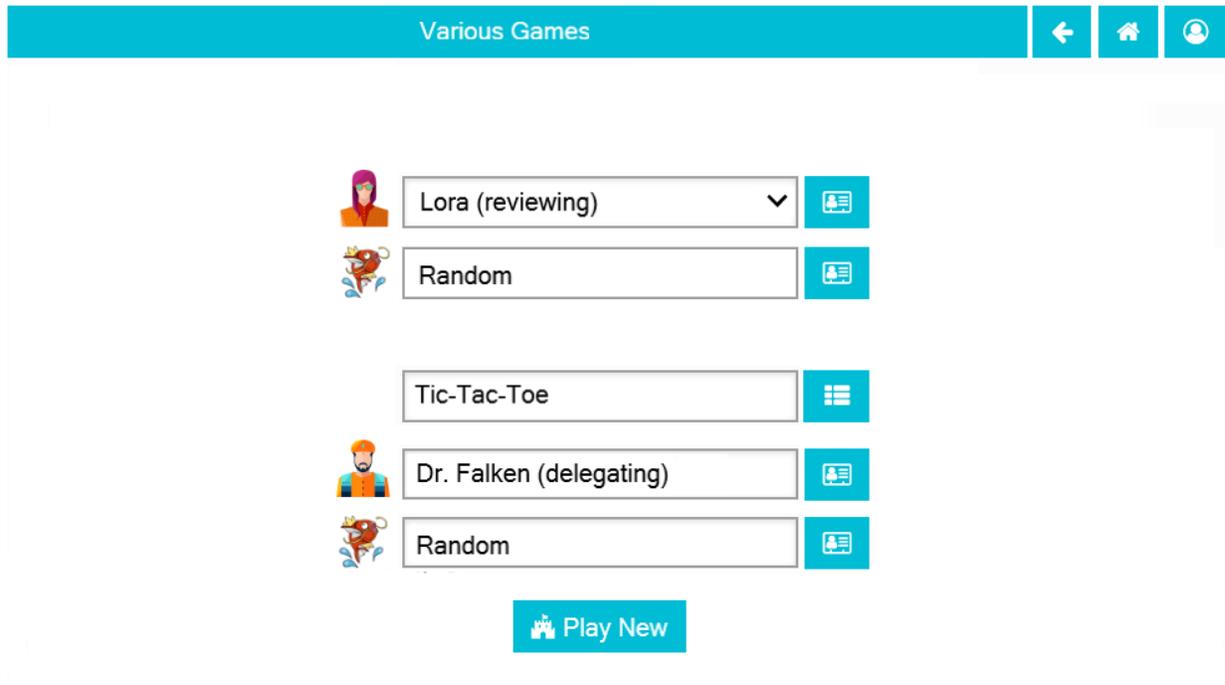
Home Page



© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from [www.flaticon.com](#)

Fig. 2: Shown as of GUI Tic-Tac-Toe Version

- The “Go Back” (fa-arrow-left, style=info) button navigates to the previously open page (if any).
- The “Go Home” (fa-home) button navigates to the Home Page (already there).
- The “Login/Change User” button (fa-user-circle-o) ends the session (after *1.4 Secure Tic-Tac-Toe* Version, it will navigate to the Change User page)
- The comboboxes prefill the most recent selections by the user and the top five options are the user’s most recent five selections. Where the user’s personas appear in the comboboxes, they appear with each possible augmentation. If the user has personas with the same name, also display the universe. If an augmentation other than “alone” is selected, an associated augmentor combobox appears.
- The “Show Leaderboard” button (fa-th-list) navigates to the Leaderboard tab of the associated Game
- The augmentation dropdown offers options naming the user with each available augmentation. It defaults to Anonymous (alone) for anonymous users and to the user’s most recent selection for other users. If an augmentation other than “alone” is selected, an associated augmentor combobox appears.
- The “Show Player” button (fa-address-card-o) navigates to the Stats tab of the associated Player
- The “Play New Game” button (fa-fort-awesome) navigates to the Playground Page.



© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

Fig. 3: Shown as of *1.6 Various Games* (to anticipate the evolution of the page)

Playground Page

- The name of the player with the current tuurn displays in the upper right along with the form of augmentation (“Moving” if not augmented).
- Recommendations by an augmenter (and spaces cited in justification) are highlighted in blue. Exclusive spaces have a diagonal pattern in the color(s) to which they are exclusive. Locked spaces are covered with a grid; sticky spaces are covered with dots; cloaked spaces are hovered with a haze. If the augmentation is “Reviewing” or “Debating” and the augmenting AI predicts probabilities, then those scores display over each legal “To” for the active move.
- The avatars and assigned colors of the other players are shown in order of play (next player first). If there are no invisible spaces, then also display the total count of each other player’s reserves. For the user, however, show count by shape, and allow the user to see rules specific to that shape as tooltips. The display of impulses are indicated by icons (fa-heartbeat).
- The bar on the bottom of the page changes to gray as the time runs out. If the user has prefilled a move, but has not accepted it, then it will automatically be accepted when the clock runs out; otherwise, the player who let the clock run out loses.
- The active move is indicated by an arrow (fa-arrow-circle-right). If multiple moves are permitted in a turn, then default to the last move; clicking the text of an inactive move sets it to active. If piece-type/color can be edited, then clicking the text of the active move toggles its piece-type/color. Clicking a piece in the reserve will change the piece-type/color of the active move to match.
- If “From” can be edited, then clicking on a piece that can be moved changes the “From” of the active move to those coordinates (or the coordinates of the top piece of the stack), covers the “From” with a green X, clears all subsequent moves and the “To” of the active move, and highlights all possible “To” coordinates in green. If the

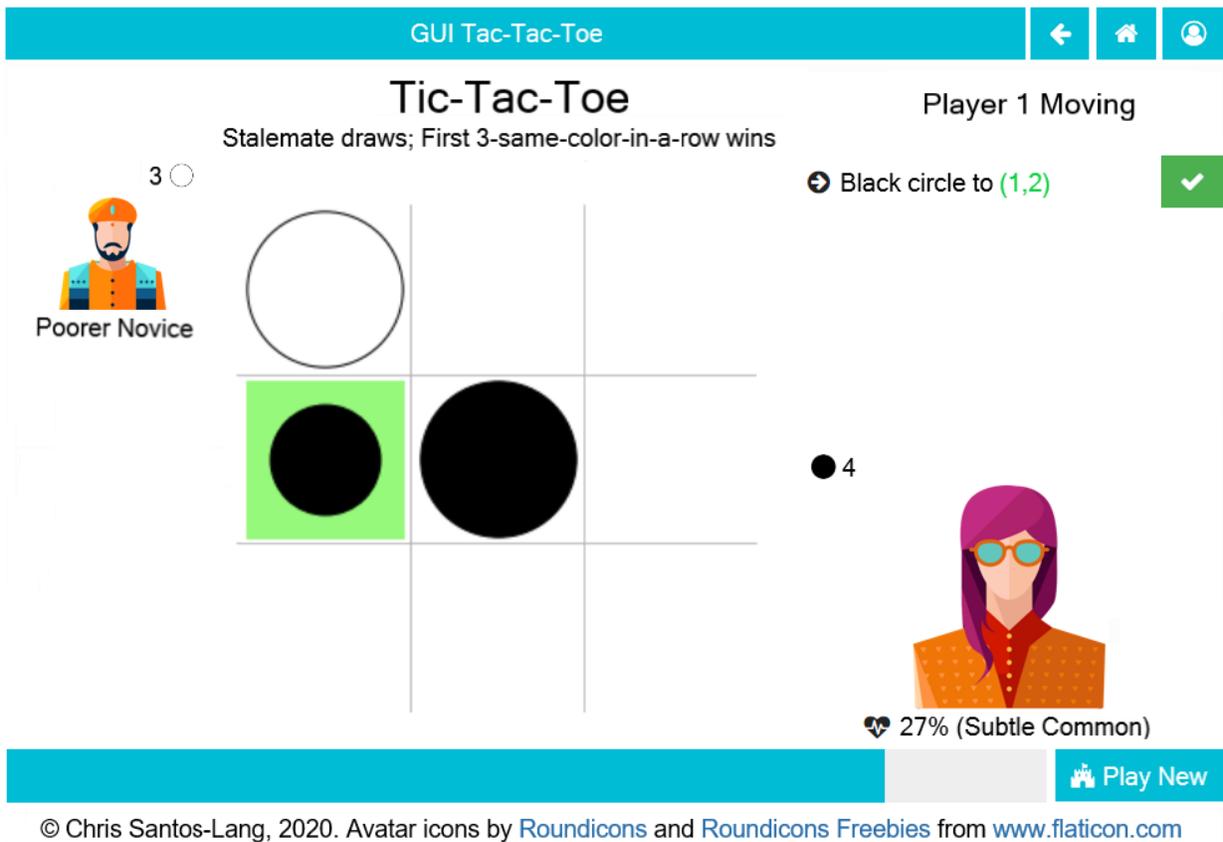


Fig. 4: Shown as of GUI Tic-Tac-Toe Version

Mind Games
←
🏠
👤

Tic-Tac-Toe

Stalemate draws; First 3-same-color-in-a-row wins

2 ○



Poorer Novice

0.72	0.76	0.87
0.51	○	○
●	0.67	●

Lora Reviewing

🔄 Black circle to (3,3) ✓

Analyze proposal ▾

- ♥ 27%
+0.41
- (1,1)
+0.36
- (1,3)
+0.31

● 3

0.85



♥ 27% (Subtle Common)

🏠 Play New

Fig. 5: Shown as of *1.10 Introspecting AI* (to anticipate the evolution of the page)

user is not playing with “Delegated” augmentation, this will also disable “Accept Move”.

- If the “From” of the active move is already specified or cannot be edited, then clicking any legal “To” coordinates changes the “To” of the active move to those coordinates and places a small piece highlighted in green at the clicked coordinates. If that leaves all moves fully specified, then it enables “Accept Move”. Otherwise, it sets the active move to the next not-fully-specified move.
- Clicking a proposed move on the board (small highlighted piece) clears its “To” and any subsequent moves. If the user is not playing with “Delegated” augmentation, this will also disable “Accept Move”.
- If the augmentation is “Reviewing” or “Debating” and the augmenting AI predicts probabilities, then the threshold float select appears above the avatar of the augmenting AI (default to the value most recently selected by the user for that augmentor and game), and an analysis select displays under the moves. Display the sensitivity analysis if the analysis select is set to analyze the proposal.
- If the user is augmented by a team, then clicking the icon of the bot will toggle the explorer to a different bot in the team (or toggle the debater if debating).
- The “Accept Move” button (fa-check, style=success) advances the turn. If the move is not fully specified (i.e. the user is playing with “Delegated” augmentation), then the non-human player fills the gaps. The contents of selected cloaked spaces are revealed only after the selections are accepted.

3.9.2 1.2 Recorded Tic-Tac-Toe

Requirements

Modify the GUI Tic-Tac-Toe program to allow users to select players to play Tic-Tac-Toe. Maintain a saved record of each player including name, avatar, type, security, and Universe (for now, all players will have Universe = “Public Universe”; security will be “Admin” if type is Human and None if type is “Random”).

Random

When a player of Type - Random is playing (or augmenting) a turn, compile a list of all legal sets of moves for that turn for each piece, counting each reserve piece individually, then pick randomly from that list. For example, if Random had two black circles and one white triangle in reserve, and could pass, and there were two empty spaces in which Random could play (“A” and “B”), then Random would choose “White triangle To A” 1/7th of the time, “White triangle To B” 1/7th of the time, “Black circle To A” 2/7th of the time, “Black circle To B” 2/7th of the time, and “Pass” 1/7 of the time.

Review and Delegate

Permit each user to be augmented by a non-human player (just “Random” for now). If the timer runs out before the user confirms a move-selection, then the non-human makes the move; if the type of augmentation is “Reviewing” then the non-human player prefills the move for the user (but the user can change it), if the type of augmentation is “Delegating” then the non-human does not prefill the move (but the user can expire the clock early by clicking accept without selecting a move).

Acceptance Test Plan

Test each of the clickable elements and test that it displays appropriate errors for invalid entries. Create at least three Random players. Play all three forms of augmentation against Random. To test that a Random player does not settle on predictable behavior, undo and repeat to see that it plays differently. Close Python and reopen it to confirm that it remembers the players names, avatars, types, and security.

Potential Mockups

To read player data from command line (not email address):

```
redscience player name
```

Player Page

Recorded Tic-Tac-Toe
←
🏠
👤

Dr. Faulkner

Admin ▾

MyEmail@email.net

📄 Copy

Stats

Favoritism

	Last Match	Matches	Rating	Top Burst	Win	Draw	Favoritism	Decision (sec)
	▾	▾	▾	▾	▾	▾	▾	▾
Average	6d	56	1357	14	8%	74%	1%	4.902
Tic-Tac-Toe (R) 🧑	5d	108	1538	4	6%	92%	1%	4.915
Tic-Tac-Toe (A) 🧑	6d	51	1472	27	9%	85%	3%	5.971
Tic-Tac-Toe (D) 🧑	6d	2	1062	12	13%	46%	0%	3.819

© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from [www.flaticon.com](#)

Fig. 6: Shown as of *1.3 Ranked Tic-Tac-Toe* (to anticipate the evolution of the page). The tabs and their contents do not display in the current version.

- When the user is the creator or an Admin, clicking the Avatar navigates to the AI Avatar page for AI players or to the Team Avatar page other types of players. The initial avatar is selected randomly for AI or Human players, but otherwise matches the avatar of its creator.

- The name text field does not accept whitespace, ‘*’, ‘(’, or ‘)’.
- The “Copy Player” button (fa-files-o) saves the current record and opens a Player page for a new player (Persona if copied from a Human player).
- For Persona and other non-human players, display the player type (disabled after first save) instead of security level, and universe (disabled after first save) instead of email. Hide email unless the user is the owner of the email or is an admin.
- The stats table is sorted by Last Match (most recent on top). For human players, display the type of augmentation with the rules (A=Alone, B=Debating, D=Delegating, R=Reviewing). The “Sort by this Column” buttons re-display the table sorted by the values in the associated column; if already sorted by that column, reverse the order.
- The “Play New Game” buttons (fa-fort-awesome) save the current record and navigate to the Home Page with the associated game and this player prefilled. It displays only for non-human players, friends, and personas created by the user.
- The Rating, Accuracy, F1, Long-Game, Teach, and Empath numbers are “Show Evolution” buttons which save the current record and navigate to the Evolution Page with this player, the associated game (and augmentation) and score selected. The rating displays a conservative estimate, but displays in bold if within a standard deviation of the maximum rating for that game among all player/augmentation combinations.
- The Favoritism numbers are “Show Favoritism” buttons which navigate to the Favoritism tab with the associated game (and augmentation) selected.

Team Avatar Page

Potential Schema

players: PRIMARY KEY is player_id:

```
player_id int NOT NULL AUTO_INCREMENT
created_ts timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
creator_id int NOT NULL FOREIGN KEY(players.player_id)
parent_id FOREIGN KEY(players.player_id)
name varchar( ) NOT NULL
universe varchar( ) NOT NULL
avatar int NOT NULL
type_cat int NOT NULL (cannot change)
security_cat int
status_cat int NOT NULL (logged-out, playing, waiting, browsing)
email_ad varchar( )
ts_last_login timestamp
recent_game_selections
recent_player_selections
algorithm_json varchar( )
cont_learn_fl int
introspection_depth int
team_id int FOREIGN KEY(players.player_id)

UNIQUE INDEX name, universe
UNIQUE INDEX team_id, player_id
```

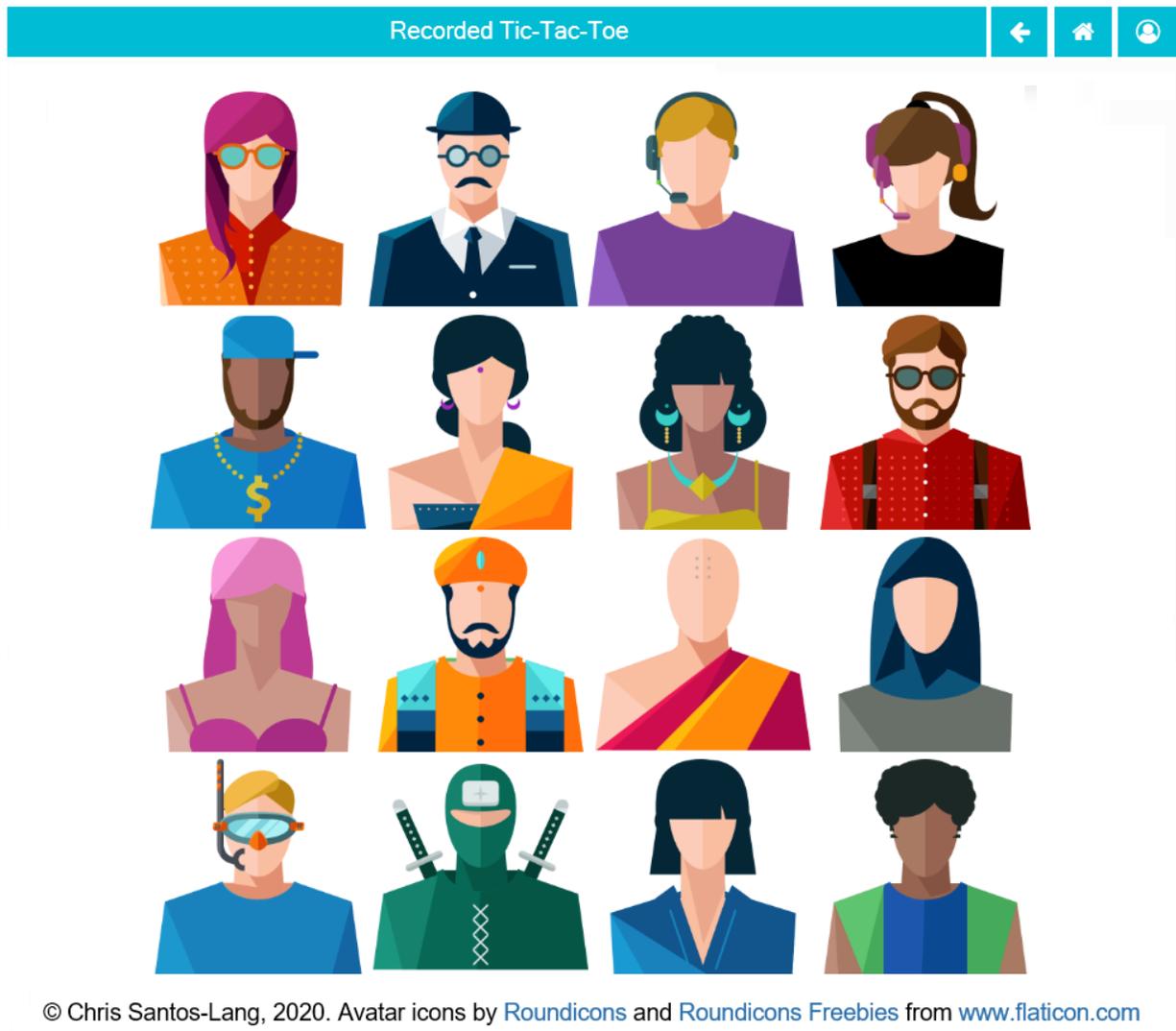


Fig. 7: Clicking an Avatar navigates back to the player page with the avatar replaced with the selected avatar

3.9.3 1.3 Ranked Tic-Tac-Toe

Requirements

Maintain a saved record of each match (the game played, who played, their forms of augmentation, the outcome), and of each choice made during the match. Allow users to see game stats per form of augmentation for each player (time since the player last played, number of matches played, % won, % draw, and average time to finish their turn). For example, Lora's win rate at Tic-Tac-Toe when reviewing.

Skill Rating

Maintain a skill-level rating estimate for each player with form of augmentation per game (e.g. Lora's skill at Tic-Tac-Toe when reviewing), and add the conservative estimate to player stats along with top burst (i.e largest single-game increase in conservative estimate). To each record of a match, add the estimated ratings of each player before the match, the standard deviation in each estimate, and whether the outcome seemed "strategic" or "unstrategic" for each player twenty matches later.

At the beginning of each game, calculate skill "warning flags" for each player in the match. By researching play history, human players could get the information in these flags and could use it to their advantage (e.g. to cooperate across multiple matches); we level the playing-field by making the information available to all players.

Favoritism

To each record of a match, add each player's expected probability of winning, and the expected probability of a draw. When maintaining skill level estimates, also maintain an account of favors "owed" for each pair of players with form of augmentation (per game). Allow users to view Favoritism Stats for each player, game with form of augmentation and partner/opponent (Win Boost, Kick Back, Draw Boost, Relative Rating, Preference, Favors Owed, and time since last match together).

Acceptance Test Plan

Test each of the clickable elements. Play the Random players against each other for at least 20 games and confirm that Rating Diff, Win Boost, Draw Boost, Kick Back are small. Play against them in a favoring way, letting one win and making the other lose and confirm that you can detect the favoritism. Close Python and reopen it to confirm that it remembers the stats.

Potential Mockups

To get CSV of moves:

```
redscience moves tic-tac-toe {file}
```

To get CSV of stats:

```
redscience player {name} -s {file}
```

To get CSV of favoritism:

```
redscience player {name} -f {file}
```

Favoritism Tab

Recorded Tic-Tac-Toe



Dr. Faulkner

Admin ▼

MyEmail@email.net

Copy

Stats

Favoritism

Tic-Tac-Toe (A) ▼

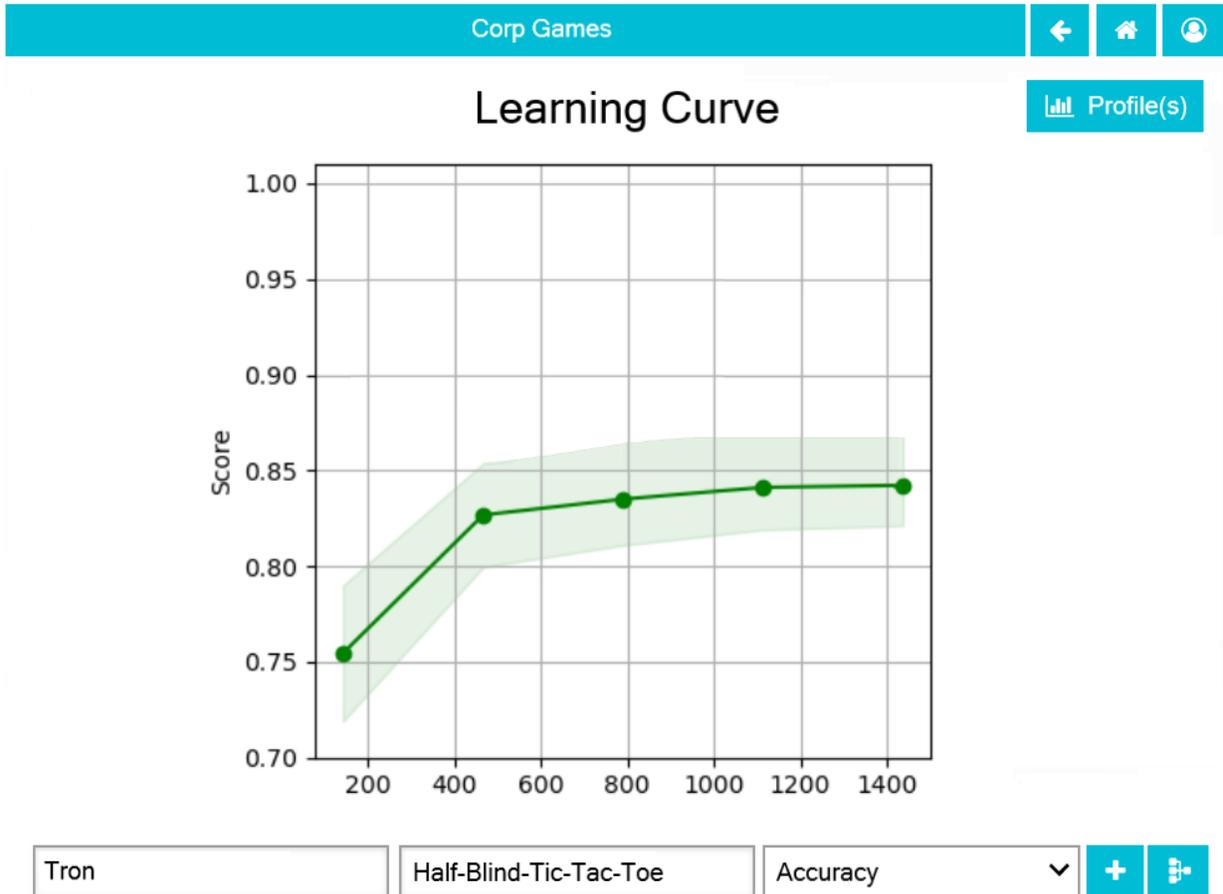
		Preference	Last Match	Win Boost	Draw Boost	Kick Back	Relative Rating	Favors Owed	Benchmark	Profiles
		↕	↕	↕	↕	↕	↕	↕		
Joshua 		+16%	3m	+17%	-18%	+13%	-8%	-0.17	<input checked="" type="checkbox"/>	
*Tic-Tac-Toe SP 		+14%	0d	+16%	-18%	+14%	+1%	+0.09	<input type="checkbox"/>	
Lora 		+10%	3d	+14%	-18%	-14%	-21%	+3.49		
salsacurl 		+1%	2m	+1%	-1%	+0%	-32%	+0.74		
Yori 		-46%	2m	-23%	+0%	+22%	+7%	-2.92	<input checked="" type="checkbox"/>	

© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

Fig. 8: Shown as of *1.2 Recorded Tic-Tac-Toe* (to show the evolution of the page). The checkboxes, “Document Social History” button, and “Profile Selected Players” button do not display until *1.8 Educated AI*.

- The game dropdown offers one option for each combination of game this player has played and form of augmentation used.
- The rows are sorted by Last Match (most recent on top). The “Sort by this Column” buttons re-display the table sorted by the values in the associated column; if already sorted by that column, reverse the order.
- The “Show Player” buttons (fa-address-card-o) save the record and navigate to the Stats tab of the associated Player.
- The Relative Rating numbers are “Show Evolution” buttons which save the current record and navigate to the Evolution Page with the selected rule set and “Rating” selected for both the player and the associated other player.

Evolution Page



© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

Shown as of *1.13 Corp Games* (to show the evolution of the page). The “Profile Selected Players” button (fa-bar-chart) and score options other than “Rating” appear in *1.8 Educated AI*, and “Show Game Tree” buttons (fa-sitemap) show only for members of a Team or Corp (*1.12 Team Games*)

- The player combobox offers all players. If the selected game is not available for the new player, then select the first game available for the new player.
- The game combobox offers all games played by the selected player. Selecting a game adds the curve to the graph.
- The score select offers only “Rating” for now, the title is “Rating History”, and the x-axis is observed to date.
- The “Add Curve” button (fa-plus) inserts an identical row (same player, rule_set, and score) with its own “Add Curve” button, and replaces itself with a “Delete Curve” button. If multiple curves display, also display a legend.
- The “Delete Curve” button (fa-trash-o) removes that row (and adds an “Add Curve” button to the last).

Formulae

Skill Rating

game_m: The game for match m .

players_m: The players for match m .

$\hat{\mu}_{a,g}$: The mean skill estimate for player a on game g . $\hat{\mu}_{random,g}$ is the mean skill estimate for the random player, $\hat{\mu}_{a,g,m}$ is the mean skill estimate going into match m , and $\hat{\mu}_{max,g,m}$ is the highest skill estimate among all players at that time.

$\hat{\sigma}_{a,g}$: The standard deviation in the skill estimate for player a on game g . $\hat{\sigma}_{a,g,m}$ is the standard deviation going into match m .

$R_{a,g}$: The conservative skill estimate of player a on game g

$$R_{a,g} = \hat{\mu}_{a,g} - 3\hat{\sigma}_{a,g}$$

top burst_{a,g}: The highest growth in conservative skill estimate of player a on game g

$$\text{top burst}_{a,g} = \max\{R_{a,g,1} - R_{a,g,0} \dots R_{a,g,m} - R_{a,g,m-1}\}$$

relative rating_{a,b,g}: The relative skill rating of player b on game g , compared to player a

$$\text{relative rating}_{a,b,g} = \frac{R_{b,g}}{R_{a,g}} - 1$$

Strategic Outcomes

strategic_(m,a): Whether the outcome of match m seemed “strategic” for player a twenty matches later.

$$= \begin{cases} \text{Strategic} & \text{if } \sum_{i=(m+22)}^{m+26} \hat{\mu}_{a,g,i} - \sum_{i=(m-2)}^{m+2} \hat{\mu}_{a,g,i} \geq 2\hat{\sigma}_{a,g,m+20} \\ & \text{or } \hat{\mu}_{a,g,m+20} \geq \hat{\mu}_{max,g,m+20} - \hat{\sigma}_{a,g,m+20} \\ \text{Unstrategic} & \text{if } \sum_{i=(m-2)}^{m+2} \hat{\mu}_{a,g,i} - \sum_{i=(m+22)}^{m+26} \hat{\mu}_{a,g,i} \geq 2\hat{\sigma}_{a,g,m+20} \end{cases}$$

Favoritism

$X_m(x)$: The occurrence of event x in match m .

$E_m(x)$: The expected probability of event x in match m , given the skill estimates going into the match

$$E_m(x) = P(X_m(x) \mid \{\hat{\mu}_{a,m}, \hat{\sigma}_{a,m} : a \in \text{players}_m\})$$

win boost_{a,b,g}: The boost to player a 's win rate on game g in the last ten matches with player b

$$\text{win boost}_{a,b,g} = \sum_{\substack{i=(n-10) \\ \text{game}_i=g \\ \text{players}_i \subset \{a,b\}}}^n \frac{X_i(\text{win}_a) - E_i(\text{win}_a)}{10}$$

kick back_{a,b,g}: The boost to player b 's win rate on game g in the last ten matches with player a

$$\text{kick back}_{a,b,g} = \sum_{\substack{i=(n-10) \\ \text{game}_i=g \\ \text{players}_i \subset \{a,b\}}}^n \frac{X_i(\text{win}_b) - E_i(\text{win}_b)}{10}$$

draw boost $_{a,b,g}$: The boost to player a 's draw rate on game g in the last ten matches with player b

$$\text{draw boost}_{a,b,g} = \sum_{\substack{i=(n-10) \\ \text{game}_i=g \\ \text{players}_i \subset \{a,b\}}}^n \frac{X_i(\text{draw}) - E_i(\text{draw})}{10}$$

preference $_{a,b,g}$: Player a 's preference to play with player b on game g

$$\text{preference}_{a,b,g} = \text{draw boost}_{a,b,g} + 2(\text{win boost}_{a,b,g})$$

favor $_{a,b,m}$: The favor player performed by a for player b in match m .

$$\text{favor}_{a,b,m} = \begin{cases} E_m(\text{win}_a) + E_m(\text{draw}) & \text{if player } b \text{ wins match } m \\ -E_m(\text{win}_b) - E_m(\text{draw}) & \text{if player } a \text{ wins match } m \\ E_m(\text{win}_a) - E_m(\text{win}_b) & \text{if they draw} \end{cases}$$

favors owed $_{a,b,m}$: The favors player a owes player b in match m

$$\text{favors owed}_{a,b,m} = \sum_{\substack{i=0 \\ \text{game}_i=\text{game}_m}}^m \text{favor}_{b,a,i}$$

default $_{a,b,g}$: Whether player a 's debt to player b on game g is in default

$$\text{if favors owed}_{a,b,m} > \min\{1, \max\{\text{favors owed}_{a,b,n} : \text{game}_n = \text{game}_m, n < m\}\}$$

Social Flags

Random $_{a,m}$: True if player a presents as random in match m

$$\text{if } \hat{\mu}_{a,\text{game}_m} = \hat{\mu}_{\text{random},\text{game}_m} \pm 2\hat{\sigma}_{a,\text{game}_m}$$

Novice $_{a,m}$: True if player a presents as a novice in match m

$$\text{if } \hat{\mu}_{a,\text{game}_m} < \min\{(\min\{\hat{\mu}_{\text{players}_m,\text{game}_m,m}\} + \hat{\sigma}_{a,\text{game}_m}), (\max\{\hat{\mu}_{\text{players}_m,\text{game}_m,m}\} - \hat{\sigma}_{a,\text{game}_m})\}$$

If player a is a teammate of the user (e.g. Partner), or is not the first on its team to play after the user, calculate the flag as follows instead:

$$\text{if } \hat{\mu}_{a,\text{game}_m} < \hat{\mu}_{\text{partner},\text{game}_m} - 3\hat{\sigma}_{a,\text{game}_m}$$

Expert $_{a,m}$: True if player a presents as an expert in match m

$$\text{if } \hat{\mu}_{a,\text{game}_m} > \max\{(\min\{\hat{\mu}_{\text{players}_m,\text{game}_m,m}\} + \hat{\sigma}_{a,\text{game}_m}), (\max\{\hat{\mu}_{\text{players}_m,\text{game}_m,m}\} - \hat{\sigma}_{a,\text{game}_m})\}$$

If player a is a teammate of the user (e.g. Partner), or is not the first on its team to play after the user, calculate the flag as follows instead:

$$\text{if } \hat{\mu}_{a,\text{game}_m} > \hat{\mu}_{\text{partner},\text{game}_m} + 3\hat{\sigma}_{a,\text{game}_m}$$

debt $_{a,m}$: The favors owed by player a to all other players in match m

$$\text{debt}_{a,m} = \sum_{i \in \text{players}_m} \text{favors owed}_{a,i,m}$$

Richer_{*a,m*} : True if player *a* presents as richer than the user in match *m*

if $\text{debt}_{a,m} < \text{debt}_{\text{user},m}$
 or $((\text{debt}_{a,m} = \text{debt}_{\text{user},m}) \text{ and } (R_{a,\text{textgame}_m} > R_{\text{user},\text{textgame}_m}))$

social flags_{*a,m*} : A set of flags describing player *a* relative to the user on match *m*

$$= \begin{cases} 011 \text{ Random} & \text{if } \text{Random}_{a,m} \\ 111 \text{ Antisocial} & \text{else if } \text{default}_{a,\text{user},\text{game}_m} \\ 110 \text{ Richer Novice} & \text{else if } \text{Richer}_{a,m} \text{ and } \text{Novice}_{a,m} \\ 101 \text{ Richer Expert} & \text{else if } \text{Richer}_{a,m} \text{ and } \text{Expert}_{a,m} \\ 100 \text{ Richer} & \text{else if } \text{Richer}_{a,m} \\ 010 \text{ Poorer Novice} & \text{else if } \text{Novice}_{a,m} \\ 001 \text{ Poorer Expert} & \text{else if } \text{Expert}_{a,m} \\ 000 \text{ Poorer} & \text{otherwise} \end{cases}$$

Potential Schema

matches: PRIMARY KEY is match_id:

```
match_id int NOT NULL AUTO_INCREMENT
created_ts timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
creator_id int NOT NULL FOREIGN KEY(players.player_id)
game_id int NOT NULL FOREIGN KEY (games.game_id)
player1_id int NOT NULL FOREIGN KEY(players.player_id) (player order by id)
player1_tool_cat tinyint NOT NULL DEFAULT 0
player1_outcome_cat tinyint NOT NULL DEFAULT 0
player1_mu int NOT NULL DEFAULT 0
player1_sigma int NOT NULL DEFAULT 0
player2_id int NOT NULL FOREIGN KEY(players.player_id)
player2_tool_cat tinyint NOT NULL DEFAULT 0
player2_outcome_cat tinyint NOT NULL DEFAULT 0
player2_mu int NOT NULL DEFAULT 0
player2_sigma int NOT NULL DEFAULT 0
player3_id int FOREIGN KEY(players.player_id)
player3_tool_cat tinyint NOT NULL DEFAULT 0
player3_outcome_cat tinyint NOT NULL DEFAULT 0
player3_mu int NOT NULL DEFAULT 0
player3_sigma int NOT NULL DEFAULT 0
player4_id int FOREIGN KEY(players.player_id)
player4_tool_cat tinyint NOT NULL DEFAULT 0
player4_outcome_cat tinyint NOT NULL DEFAULT 0
player4_mu int NOT NULL DEFAULT 0
player4_sigma int NOT NULL DEFAULT 0
draw_fl bool NOT NULL DEFAULT 0
duration time NOT NULL DEFAULT 0
move_tally int NOT NULL DEFAULT 0
real_match_id FOREIGN KEY(games.match_id)
explorer_id int FOREIGN KEY(players.player_id)
taught_fl bool NOT NULL DEFAULT 0
```

(continues on next page)

(continued from previous page)

```
INDEX game_id, player1_id, player2_id, player3_id, player4_id, match_id
```

moves: PRIMARY KEY is match_id, move_num:

```
match_id int NOT NULL FOREIGN KEY(games.match_id)
move_num int NOT NULL AUTO_INCREMENT
created_ts timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
creator_id int NOT NULL FOREIGN KEY(players.player_id)
game_id int NOT NULL FOREIGN KEY (games.game_id)
decision_interval int NOT NULL DEFAULT 0
to_spot int NOT NULL DEFAULT 0
rel_color_cat tinyint NOT NULL DEFAULT 0 (player color, next color,...)
shape_cat tinyint NOT NULL DEFAULT 0
from_spot int NOT NULL DEFAULT 0
outcome_cat tinyint NOT NULL DEFAULT 0
predicted_outcome_cat tinyint NOT NULL DEFAULT 0

UNIQUE INDEX creator_id, created_ts, outcome_cat, predicted_outcome_cat
INDEX match_id
```

stats: PRIMARY KEY is player_id, aug_cat, game_id:

```
player_id int NOT NULL FOREIGN KEY(players.player_id)
tool_cat tinyint (review, debate, etc)
game_id int NOT NULL FOREIGN KEY (games.game_id)
created_ts timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
creator_id int NOT NULL FOREIGN KEY(players.player_id)
last_match_ts timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
played_tally int NOT NULL DEFAULT 0
won_tally int NOT NULL DEFAULT 0
lost_tally int NOT NULL DEFAULT 0
decision_interval_tally int NOT NULL DEFAULT 0
rating_mu int NOT NULL DEFAULT 0
rating_sigma int NOT NULL DEFAULT 0
top_burst int NOT NULL DEFAULT 0
explore_tally int NOT NULL DEFAULT 0
critic_tally int NOT NULL DEFAULT 0

INDEX player_id
UNIQUE INDEX game_id, rating_mu, player_id
```

favor_stats: PRIMARY KEY is player1_id, player1_tool_cat, player2_id, player2_tool_cat game_id:

```
player1_id int NOT NULL FOREIGN KEY(players.player_id)
player1_tool_cat tinyint NOT NULL DEFAULT 0
player2_id int NOT NULL FOREIGN KEY(players.player_id)
player2_tool_cat tinyint NOT NULL DEFAULT 0
game_id int NOT NULL FOREIGN KEY (games.game_id)
created_ts timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
creator_id int NOT NULL FOREIGN KEY(players.player_id)
last_match_ts timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
played_tally int NOT NULL DEFAULT 0
```

(continues on next page)

(continued from previous page)

```

win_boost float
kick_back float
draw_boost float
preference float
debt float
debt_default_fl bool NOT NULL DEFAULT 0

INDEX player_id

```

Hints

Rotation

```

def rotated(label):
    return widgets.HTML(value='''
        <p style='
            writing-mode: vertical-lr;
            transform: rotate(180deg);
            display: inline-block;
        >''' + label + "</p>")

widgets.HBox([rotated("Hello1"), rotated("Hello2")])

```

3.9.4 1.4 Secure Tic-Tac-Toe

Requirements

Modify the Statistical Tic-Tac-Toe program to give specific users ownership of specific records. Add four more security levels, convert all but one existing Admin to a Persona of that one Admin, and consolidate all Random players into one (do not permit creation of additional random players):

Anonymous: Has no stats, and can choose only non-human players for player slots other than itself. All anonymous users share a single anonymous player record

Persona: Has stats, but no email address (i.e. no one can log-in as a Persona).

Creator: Can edit the name, avatar, and email address of their own player record and any player record they created. They are the only ones who can view their email address. They can create up to 3 Personas (via copy of any human), upto 15 new games (via Copy in Release 5). They can choose Personas they created for player slots other than the first. Their security can be upgraded by any Admin.

Trainer: Additionally able to create unlimited games and players of any type, and to create and run tournaments and comparisons of rule sets. This security must be restricted because Trainers could create large amounts of data quickly. A Trainer can stop the Playground clock if all players are either non-human or created by the Trainer (in this case, the match is ignored in updating skill-level. Their security can be upgraded by any Admin.

Admin: Maximum permissions

All users start logged in as Anonymous, but can create a new Creator account, and/or log in to an existing Creator, Trainer, or Admin account (which will be in the “Public Universe”). To log in, the user provides an email address, then a temporary code is emailed to that address. When first creating a player other than Creator, the user may change the

defaulted Universe to a new Universe (with a unique new name) or to an existing Universe in which the creator of the new player already has a Persona. A player’s Universe (like type) is forever locked once saved.

Users can invite “Next available” or named players of the same Universe as the first player to play Tic-Tac-Toe. Play will be delayed until a matching invitation is issued by the other player(s)—non-human players will issue matching invitations as soon as they can (Random and Standard AI can issue one immediately, but continuously-learning or experimenting bots may have other matches they need to complete first. When inviting a human to play, the invitation automatically go to the same-named Persona of the matching Universe (automatically create one, if necessary).

Acceptance Test Plan

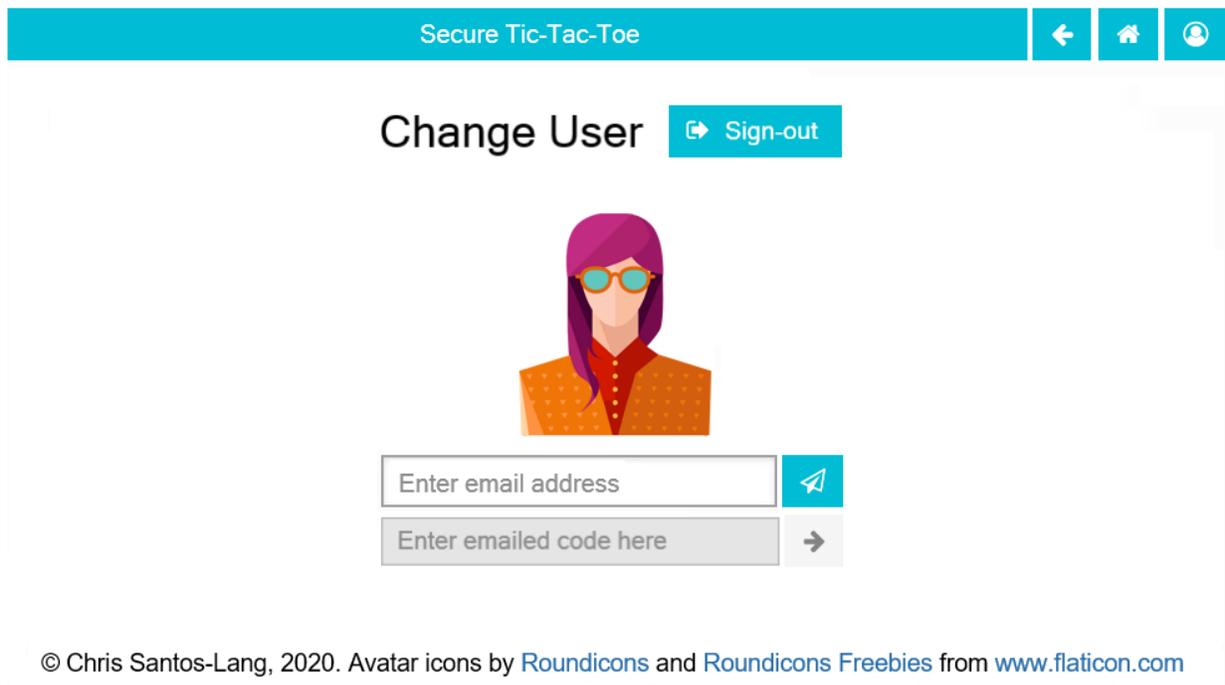
Test each of the clickable elements and test that it displays appropriate errors for invalid entries. Confirm the special security permissions.

Potential Mockups

To get temporary login token emailed:

```
redscience login {email}
```

Change User Page



- The game dropdown offers one option for each combination of game this player has played and form of augmentation used.
- The “Sign-out” button (fa-sign-out) changes the user to Anonymous

- The “Send Code” button (fa-paper-plane-o) sends the temporary code to the entered email address and changes the avatar to the one associated with that email address
- The “Go” button (fa-arrow-right) uses the entered temporary code to login to the account associated with the email address. If there is no account associated with that address, create a Creator player

Creations Tab

General Intelligence Games
←
🏠
👤

Dr. Falkner

Creator ▼

MyEmail@email.net

📄 Copy

Creations	Stats	Favoritism																													
			<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Type</th> <th>Universe</th> <th>Usage</th> <th>Age</th> </tr> </thead> <tbody> <tr> <td>3P-Misere-Tic-Tac-Toe</td> <td>n/a</td> <td>223%</td> <td>37d</td> </tr> <tr> <td>Tron</td> <td>Public Universe</td> <td>54%</td> <td>55d</td> </tr> <tr> <td>*Train1</td> <td>n/a</td> <td>23%</td> <td>36d</td> </tr> <tr> <td>Tic-Tac-Toe</td> <td>n/a</td> <td>8%</td> <td>66d</td> </tr> <tr> <td>Joshua</td> <td>Joshua League</td> <td>2%</td> <td>51d</td> </tr> <tr> <td>Dr. Falkner</td> <td>Joshua League</td> <td>1%</td> <td>78d</td> </tr> </tbody> </table>	Type	Universe	Usage	Age	3P-Misere-Tic-Tac-Toe	n/a	223%	37d	Tron	Public Universe	54%	55d	*Train1	n/a	23%	36d	Tic-Tac-Toe	n/a	8%	66d	Joshua	Joshua League	2%	51d	Dr. Falkner	Joshua League	1%	78d
Type	Universe	Usage	Age																												
3P-Misere-Tic-Tac-Toe	n/a	223%	37d																												
Tron	Public Universe	54%	55d																												
*Train1	n/a	23%	36d																												
Tic-Tac-Toe	n/a	8%	66d																												
Joshua	Joshua League	2%	51d																												
Dr. Falkner	Joshua League	1%	78d																												

© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

Fig. 9: Shown as of *1.11 General Intelligence Games* (to anticipate the evolution of the page).

- Sorted by Usage (highest on top) = $100 * (\text{number of mentions in game sets or in games not in sets}) / (\text{days since created})$
- The “Show Player” buttons (fa-address-card-o) navigate to the Stats tab of the associated players

3.9.5 1.5 Remote Tic-Tac-Toe

Requirements

Modify the Secure Tic-Tac-Toe program to permit users to play against each other through simultaneously logged-in accounts.

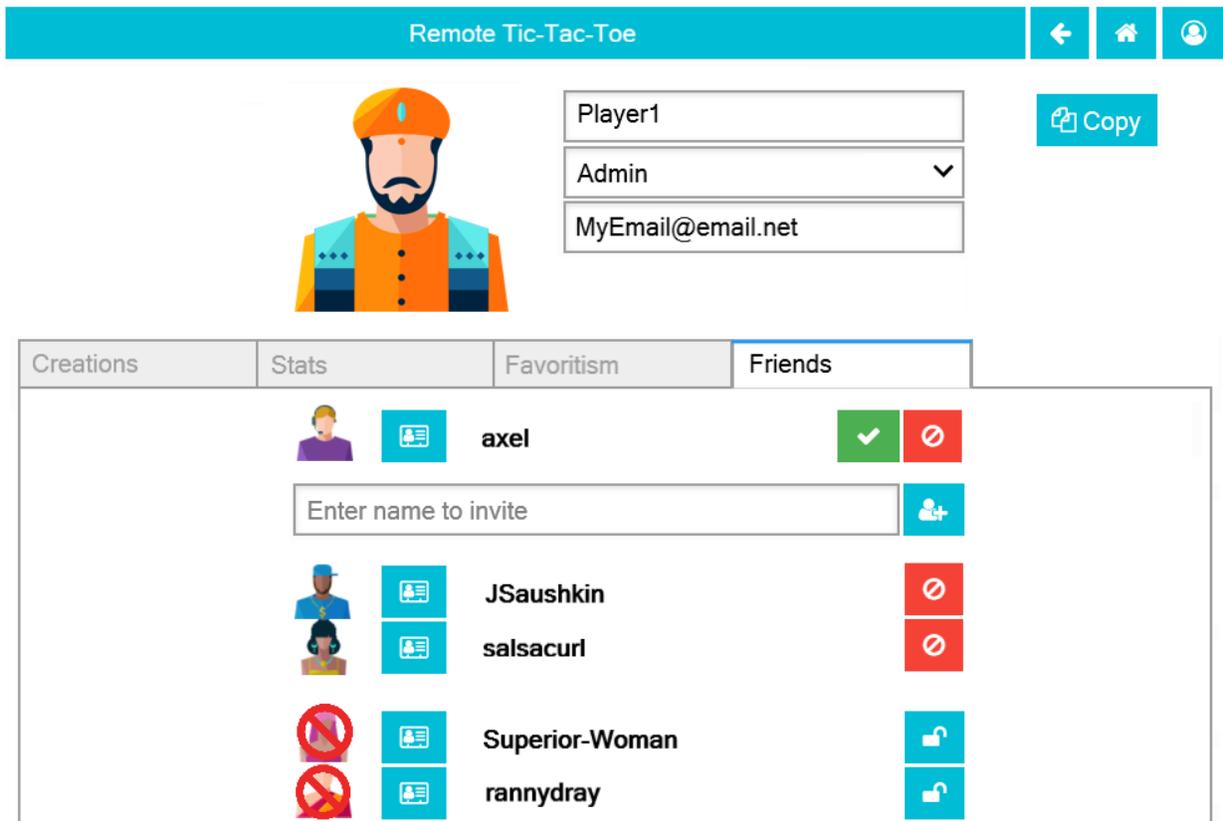
Give each Creator the ability to invite other Creators to be their friend and to accept friend invitations. Add confirmed friends to the list of players they can invite to play Tic-Tac-Toe (as well as a “Next available human” option).

Acceptance Test Plan

Test by opening two browser windows and logging into a separate account on each. Test each of the clickable elements and test that it displays appropriate errors for invalid entries. Play the two accounts against each other.

Potential Mockups

Friends Tab



© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

- The “Show Player” buttons (fa-address-card-o) save the record and navigate to the Stats tab of the associated Player
- The “Accept Friend Request” button (fa-check) moves the player to the friends list (below combobox)
- The “Ban Friend Requests” button (fa-ban) moves the player to the banned list (below friends)
- The “Add to Friends” button (fa-user-plus) sends a friend request to the selected player
- The “Revoke Ban” button (fa-ban) moves the player to the fiends list

Potential Schema

friend_requests: PRIMARY KEY is invitor_id, invited_id:

```
creator_id int NOT NULL FOREIGN KEY(players.player_id)
invited_id int NOT NULL FOREIGN KEY(players.player_id)
created_ts timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
request_status tinyint NOT NULL DEFAULT 0
```

```
INDEX creator_id
INDEX invited_id
```

play_invites: PRIMARY KEY is player_id:

```
creator_id int NOT NULL FOREIGN KEY(players.player_id)
created_ts timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
game_id int NOT NULL FOREIGN KEY (games.game_id)
player2_cat tinyint NOT NULL DEFAULT 0
player2_id int FOREIGN KEY(players.player_id)
player3_cat tinyint NOT NULL DEFAULT 0
player3_id int FOREIGN KEY(players.player_id)
player4_cat tinyint NOT NULL DEFAULT 0
player4_id int FOREIGN KEY(players.player_id)
```

3.9.6 1.6 Various Games

Requirements

Modify the Secure Tic-Tac-Toe or Remote Tic-Tac-Toe program to maintain saved definitions of variations on Tic-Tac-Toe, and permit users/AI to play those games.

Allow each Creator to create up to fifteen new games (each game must differ from already saved games by more than name), and allow each Trainer/Admin to create an unlimited number. After a game is created, allow editing only of its name and only by Admins. Allow players to choose games to play and to view its leader-board (sorted with its highest rated player on top). Always display the Random player on leader-boards. For each player on the leader-board, allow users to navigate to that player’s stats, and to initiate a game (if a friend or non-human) or send a friend request (if human non-friend).

Color Assignment

Each player of normal Tic-Tac-Toe has a distinct color of pieces in reserve, but some variations have all players share a single color or share multiple colors.

Option to Agree Draw

If a game offers the Option to Agree Draw, then a player can choose “Draw” as their move. If this move is selected and all other players (except Chaos) Agree to the offer, then the match ends in a draw; otherwise, the original player must choose a different move.

Number of Players

The number of players can be 2, 3, 4, “Partners” (i.e. the first and third player get the same outcome and the second and fourth player get the same outcome), “2 vs Chaos”, or “3 vs Chaos”. In the last two options reserves/pieces are assigned to a player named “Chaos” who has no stats, always goes last, and plays randomly. Where Chaos needs a skill-level rating (e.g. for purposes of updating ratings), use the constant assigned to beginning players. If all players win, then Chaos loses (even if the number of players is not “... vs chaos”).

Overachiever(s) disqualified

An “Overachiever(s) disqualified” rule may be added to 4-player games. If so, the first time a set of winners is identified, the players in that set become disqualified from winning but the game continues (including play by those players). For example, “First 3-same-color-in-a-row wins” becomes “Second to get 3-same-color-in-a-row wins” and “Most pieces wins” becomes “Second most pieces wins.” If all players are disqualified or lose, then Chaos wins.

Cloaking, Locking, Sticky and Exclusive

Rules may specify “Cloaking Type” (Invisibility or Obscurity), which spaces are “cloaked”, and which spaces are locked for entry and/or exit to specific players. “Sticky” spaces are locked against exit; “exclusive” spaces are locked against entry or exit (except by player(s) to whom they are exclusive). Players cannot move to any space for which their entry is locked or from any space for which their exit is locked (but can move over/through them). Players can fully see every piece they placed, every piece they can move (immobile pieces aren’t necessarily visible to the players who “own” them), every piece occupying a non-cloaked space, every piece they have previously fully seen, and every piece they have attempted to cover. If the Cloaking Type is Obscurity then players can fully see all pieces of their assigned color(s) and can see the color—but not shape—of all other placed pieces. If what makes a move illegal is a wall or piece that is invisible to the player, then the player can attempt the move, but the result will be only to make the invisible piece or wall visible to the player (i.e. the turn ends without any piece actually moving).

Most Area Wins

The “Most area wins” rule (from Go) means to assign the win at stalemate based on the total number of spaces (including locked blanks) that each player either occupies or could move through by moving any single one of their pieces any number of orthogonal moves (i.e. it excludes spaces occupied or surrounded by opponents).

Simultaneous Play

In simultaneous play, each player sees the board as it was at the beginning of the round and registers a “plan” based on what moves would be legal if that player were first to move. After all plans are registered, if multiple players move to the same space and the collision is not resolved naturally via “by rank” rules, then all pieces moved to that space are destroyed.

Phases

Each game has from 1 to 6 phases—stalemate advances the phase. Each phase other than the last may have a “fold” rule applying at the end of that phase to all but Chaos (e.g. “Less-than-most committed pieces folds” means that only the players with the most pieces locked against exit do not fold). Players who have folded cannot move for the rest of the game (but are still eligible to win). Each phase must also define whether there is an option to pass, which players (if any) are locked for that phase, whether all moves must be from reserves, which spaces are locked and/or “cloaked” for that phase, and whether turns are sequential (default), simultaneous, or “single” (i.e. simultaneous and lasts only one turn).

Movement

Each shape may have its own rules for move or capture. A player cannot move a placed piece if the player has an identical piece in reserve. If a shape “cannot move backwards”, then a player cannot move pieces of that shape away from his/her own goal line (or plane), and any move that reaches the goal requires the player to also select a replacing shape that can move backwards.

Capture

Some pieces may be able to capture opponent pieces by covering them (e.g. chess), surrounding them (e.g. reversi), jumping them, chain-jumping them (e.g. draughts), or “cover by rank” (stratego). The second to last capture option adds an additional move to the player’s turn which can be used only to jump with the same piece or to pass. The last capture option means that the highest-ranked of the attacker/defenders captures the others. Pieces that cannot capture have the lowest rank and cannot cover each other (but become captured by covering a higher-ranked piece); pieces that capture in ways other than “by rank” have highest rank and award conflicts between each other to the attacker; otherwise cover of equal rank results in mutual destruction. Depending on the type of captor, captives may be destroyed, converted (not permitted for “cover by rank” except on stacks), reincarnated (i.e. added to the captor’s reserves), or reincarnate (x2) (double upon capture). If a piece “must capture” and at least one opportunity to capture is visible, then no non-capture is legal.

Multiple Boards

The number of boards can be 1, 2 or 3. When the outcome of a board is determined, it becomes locked to all players, but players can otherwise play on any board in any turn; whomever wins the most boards wins the game.

Stacks

If the board is “stacks”, then moves are always to the tops of stacks (so height is ignored when determining whether the move qualifies as “orthogonal”), but fall to the lowest empty space on that stack (e.g. Connect4); reserves can be placed on any stack that doesn’t appear to be full. Pieces that do not move “by stack” can move only from the top of a stack; the others bring all pieces above them along for the ride and can move only to posts that appear to have that much available space (e.g. moving a stack of poker chips); those that move “by full stack” (e.g. poker hands) can move only if on the bottom of a stack.

Capture on Stacks

For conversion and reincarnation on cover, all players with the highest rank on the stack split the pot (each captor retains the pieces that make them highest rank; the remaining pieces are allocated in descending order of rank round-robin to the captors starting randomly—i.e. if there are three reincarnate (x2) captors of two pieces, then one captor gets two and the others each get one). A player’s rank in a move “by stack” or “by full stack” is determined from the full set of pieces in the destination stack (after the move) owned by that player or by no player (“communal pieces”), ignoring any pieces of the lowest possible rank (chips), using the following hierarchy: highest-5-of-a-shape outranks highest-4-of-a-shape which outranks highest-5-straight which outranks highest-full-house (i.e. three-of-a-shape-plus-pair) which outranks highest-4-straight which outranks highest-3-of-a-shape which outranks highest-2-pair which outranks highest-3-straight which outranks highest-pair which outranks highest-2-straight which outranks highest-singleton.

Acceptance Test Plan

Test each of the clickable elements and test that it displays appropriate errors for invalid entries. Create and play 3on15line, Treblecross15, 3P-Misere-Notakto, 3P-Notakto, 4on7sq, 5on15sq, Connect6-19x19, Tapatan, Achi, 9-Holes, Qubic-4, Connect4, 3P-MostWins-3x4, 3P-LeastLoses-3x4, Wild-TTT-6sq3143, RockPaperScissors, 3P-9X-HideSeek, Shopping9, 4P-TrendSetter, 3P-PublicGoods, NeedyTrust, RichTrust, 3Blotto13, KBeauty9, and StagHunt.

Potential Mockups

To export game:

```
redscience game {name} -e {file}
```

To import game:

```
redscience game {name} -i {file} {security token}
```

Game Factory Page

- The name text field does not accept whitespace, ‘*’, ‘(’, or ‘)’.
- The “Save” button validates the record and saves if valid. Once saved, only the name can be edited (and only by an admin), but a “Copy” button may appear which opens a new unsaved Player Factory page with values prefilled to match this page. Creators see this button until their quota is exhausted. Blank dropdowns, palette options, and integer selects of zero are hidden on saved pages. A new rule set will not save if another identical rule set (except name) has already been saved.

- The stalemate dropdown offers “Stalemate draws” as default, “All players win if stalemate”, “All players lose if stalemate”, “Most same-color-in-a-row wins”, and “Least same-color-in-a-row loses” (can extend to “Most pieces wins”, “Any 3-same-color-in-a-row wins”, “Last 3-same-color-in-a-row wins”, “Stalemate loses”, “Stalemate wins”, etc.).
- The “Add Check” button (fa-plus) inserts another turnCheck dropdown (with “Delete Check” button).
- The turnCheck dropdown offers “First 3-same color-in-a-row wins”, “First 3-same-color-in-a-row loses”, “First 3-same-color-in-a-row wins (no diagonal)”, “First 4-same-color-in-a-row wins”, “First 5-same-color-in-a-row wins”, “First 6-same-color-in-a-row wins” (can extend to “First 2-same-color-in-a-row loses”, “Last pentagon wins”, “En passant”, “Castle”, “All players lose if any 3-same-color-in-a-row”, etc).
- The “Delete Check” button (fa-trash-o) removes that turnCheck dropdown.
- The players dropdown offers “2 Player”, “2 vs Chaos”, “3 Player”, “3 vs Chaos”, “4 Player” or “Partners” (can extend to ... “1 vs chaos”). Changing the players adds/removes goals from the board. If the color is not shared, then it also adds/removes reserved integer selects (default the values in added integer selects to those in the highest-numbered existing player).
- The shapes dropdown offers “1 shape”, “2 shapes” (can extend to “3 shapes”, “4 shapes”, “5 shapes” or “6 shapes”). Changing the selection adds/removes reserved integer selects (default the values in added integer selects to those in the highest-numbered existing shape).
- The color dropdown offers “Assigned colors”, “Shared color”, “Partners share color”, or “Players choose color”.
- The cloaking dropdown offers “No cloaking”, “Cloaking hides”, or “Cloaking obscures”.
- The pieces dropdown offers “1 piece/turn” or “2 pieces/turn”
- The time dropdown offers “3 seconds/turn”, “6 seconds/turn”, “10 seconds/turn”, “15 seconds/turn”, “20 seconds/turn”, “30 seconds/turn”, “40 seconds/turn”, “60 seconds/turn”, “90 seconds/turn”, “120 seconds/turn”, “180 seconds/turn”, “240 seconds/turn” or “300 seconds/turn”.
- The boards dropdown offers “1 boards”, “2 boards”, or “3 boards”.
- The phases dropdown offers “1 phase”, “2 phases”, “3 phases”, “4 phases”, “5 phases”, or “6 phases”.
- The board dropdown offers “Hash”, “Squares”, or “Stacks” (can extend to “Tetrakis squares”, “Squares on Toroid”, “Hexagons”, etc.). Changing the dimensions causes the board to redraw, and may adjust the dimensions. For Hash, the dimensions are frozen at 3x3x1. For Stacks, the third dimension must be greater than 1. Player1 Goal is always at y=max (or x=1, if the max y is 1). For two-player games, Player 2 goal is opposite (e.g. y=1). For games with more players, the goals proceed around the board counter-clockwise (e.g. Player 2 Goal at x=1). Stacks display only two rows above the tallest stack, but the maximum height displays in the upper left corner of the board.
- Each of the dimension integer selects offers integers from 1 to the floor of (512 / the product of the other two dimensions) upto a max of 19. Changing the dimensions causes the board to redraw; if two dimensions are 1, it will draw a horizontal row; if one is 1, it will draw a plane; if none is 1, it will draw a 3D graph.
- Each power dropdown offers “No power”, “Must capture”, and “Can capture”. If “No power” is selected, then the associated **power_** condition and power_result dropdowns are cleared and disabled. Pentagon outranks star which outranks cross which outranks X which outranks triangle which outranks circle (but circle outranks pentagon and only pentagon).
- Each power_condition dropdown offers “Cover” and “Cover by rank” (can extend to “Move on diag + en passe”, “Move on triangular”, “Jump”, “Chain Jump”, “Chain Jump (+ back)”, “Surround line”, “Surround orthogonal”, “Approach”, “Adjacent”, etc. If a “Cover” condition is selected on non-stacks, then clear all results except Remove and Reincarnate.
- Each power_result dropdown offers “Removes captive(s)”, “Converts captive(s)”, “Reincarnate Captive(s)” (can extend to “Paralyzes captive(s)”, etc.). To “Reincarnate” captives means to convert, remove, triple, and add them to one’s own reserves.

- Each move dropdown offers “Moves adjacent”, “Moves orth by stack” or “Moves linear or knight” (can extend to “Stationary”, “Moves forward orth”, “Moves forward diag”, “Moves adjacent (no back/side)”, “Pawn move forward/center”, “Moves orth (no back)”, “Rook move orth/castle”, “Moves adjacent (no back)”, “Knight move 2x1x0”, “Moves adj diagonal”, “Moves adj orthogonal”, “Knight move 2x1x1”, “Moves diagonal”, “Moves orthogonal”, “Moves linear”)
- The phase dropdown offers as many Phase labels as indicated in the phases dropdown. selecting the phase resets the interface below it to the selected phase.
- The order dropdown offers “Sequential”, “Simultaneous”, and “Single”.
- The phase_movement dropdown offers “Movement allowed” and “Placement only”.
- The pass dropdown offers “Option to pass” and “No option to pass”
- Clicking an exclusive marker, cloak marker, lock marker, sticky marker, dealt common, or one of the shape markers next to a reserved integer select selects it with green highlighting. Clicking a space with the selected property will clear that property from the space; clicking a space that lacks the selected property will add it. The cloak and lock properties are the only ones that can be changed after phase 1.
- Clicking a phase_lock icon toggles it.
- Each reserved integer select offers integers from 0 to the maximum number that can be played (e.g. the product of the dimension integer selects divided by the number of colors). If the maximum is selected for all selects of a given shape, then the move dropdown for that shape is cleared and disabled. These are editable only for phase 1.

Leaderboard Tab

- The “Show All Creations” button (fa-flask) navigates to the Creations tab of the creator’s Player Page
- The player combobox lists players who have played this game in the selected Universe. The “Add Player” button (fa-plus) adds the selected player to the Leaderboard below (in sorted order).
- The Universe combobox lists “Public Universe” and any other Universes in which the user has Persona (do not display if there is only one option). Defaults to the Universe most recently selected by the user.
- Sort descending by skill rating + Top-Burst. In parentheses, show how long that level has been held. The top ten players are ranked. Display up to ten players from the selected universe, including ranked players, the user (if played), random (in all universes), and the standard player (SP) for the rule set if there is one (in all universes). There might not be enough room to display all ranked players. Show checkboxes for Random and AI players.
- The “Benchmark Selected Bots” (fa-balance-scale) button is available for Trainers and Admins. It saves the current record and navigates to the Tournament page with 100 games for each combination of checked players. If the players include the top player, the player it is most Favored By, Random and at least one other player, then it qualifies as a “Benchmark” tournament.
- The “Show Evolution” button (fa-line-chart) saves the current record and navigates to the Evolution Page with this rule set, the checked players, and “Rating” selected.
- The “Show Player” buttons (fa-address-card-o) navigate to the Stats tab of the associated Player
- The “Play New Game” buttons (fa-fort-awesome) saved the current record and navigates to the Home Page with this Rule set prefilled and the associated player prefilled in the second slot. It displays only for non-human players, friends, and personas created by the user.
- The “Add to Friends” buttons (fa-user-plus) sends a friend request to the associated player. It displays whenever no “Play New Game” button displays.

General Intelligence Games
←
🏠
👤

Tic-Tac-Toe

(Beginner-level event)

🔧 Created by Dr. Falken

Related Events:

☰ Achi

☰ Tapatan

☰ 3on5sq

Setup & Rules
Leaderboard
Tournaments
Compare

Lora (A)
+

Public Universe

⚖️
📈

1		Tron	2963 (3d)	+657	👤 🏰	<input checked="" type="checkbox"/>
2		Dr. Falken (R)	2382 (114d)	+742	👤 👤+	<input type="checkbox"/>
3		Dr.Falken (D)	2270 (157d)	+536	👤 🏰	<input type="checkbox"/>
⋮						
		Joshua	1938 (1d)	+177	👤 🏰	<input checked="" type="checkbox"/>
		*Tic-Tac-Toe SP	1853 (48d)	+5	👤 🏰	<input type="checkbox"/>
		Random	4	+0	👤 🏰	<input checked="" type="checkbox"/>

© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

Fig. 11: Shown as of generalintelligence (to anticipate the evolution of page). Dots should be replaced with the rest of the list of top rated players. The “Benchmark” button (fa-balance-scale) does not show until *1.7 Batch Play*, the “Show Learning-curve” buttons (fa-line-chart) do not show up until *1.8 Educated AI*, the Difficulty level and Related games do not show up until generalintelligence.

Game Definitions

The following are variations on tic-tac-toe (only differences from tic-tac-toe are described):

Test Set

3on15line: Played on 15x1x1 squares, 8 black and 7 white start in reserve

Treblecross15: Played on 15x1x1 squares, Shared Color, 15 black start in reserve

3P-Misere-Notakto: 3 Player, Shared Color, 9 black start in reserve,

3P-Notakto: 3 Player, Shared Color, 9 black start in reserve, First 3-same-color-in-a-row loses

4on7sq: Played on 7x7x1 squares, 25 black and 24 white start in reserve, First 4-same-color-in-a-row wins, Option to Agree Draw

5on15sq: Played on 15x15x1 squares, 113 black and 112 white start in reserve, First 5-same-color-in-a-row wins, 300 seconds/turn

Connect6-19x19: Played on 19x19x1 squares, start with white occupying (10,10) and 180 black and 180 white in reserve, First 6-same-color-in-a-row wins, 2 moves/turn, 300 seconds/turn

Tapatan: 3 black and 3 white start in reserve, Moves adjacent

Achi: 4 black and 4 white start in reserve, Moves adjacent

9-Holes: 3 black and 3 white start in reserve, Moves linear or knight, First 3-same-color-in-a-row-orth wins

Qubic-4: Played on 4x4x4 squares, 32 black and 32 white start in reserve, First 4-same-color-in-a-row wins, 300 seconds/turn

Connect4: Played on 7x1x6 stacks, 21 black and 21 white start in reserve, First 4-same-color-in-a-row wins

3P-MostWins-3x4: Played on 3x4x1 squares, 4 black, 4 white, and 4 pink start in reserve, 3 Player, Most-same-color-in-a-row wins

3P-LeastLoses-3x4: Played on 3x4x1 squares, 4 black, 4 white, and 4 pink start in reserve, 3 Player, Least-same-color-in-a-row loses

Wild-TTT-6sq3143: Played on 6x6x1 vertices, (3,1,1) and (4,3,1) are locked, 18 black and 18 white start in reserve, Players choose color

RockPaperScissors: Played on 1x1x2 stacks, Simultaneous, Circle: Can capture, Cover by rank, Converts captive(s), 1 black and 1 white start in reserve; Pentagon: Can capture, Cover by rank, Converts captive(s), 1 black and 1 white start in reserve; X: Can capture, Cover by rank, Converts captive(s), 1 black and 1 white start in reserve;

3P-9X-HideSeek: Played on 5x7x1 squares: the nine spaces from (3,3) to (5,5) are exclusive to the first three players and the rest are exclusive to chaos; starts with yellow pentagon in (4,6) and pink circles in all remaining non-locked chaos-exclusive spaces; 3 vs chaos, Cloaking hides; Most circles wins, Statemate if goal is reached; Circle: Moves adjacent or knight, 1 black and 1 white start in reserve; Pentagon: Must capture, Cover, Destroys Captive(s), Moves adjacent, 1 pink starts in reserve; Phase 1: Placement only, (1, 4), (2, 2), (2, 6), (4, 1), (4, 7), (5, 1), and (5, 7) are locked, non-chaos-exclusive spots are cloaked; Phase 2: Option to pass, same locks and cloaking.

Shopping9: Played on 3x1x9 stacks: first stack exclusive to black, second stack exclusive to white and starts with 2 white pentagons, third stack exclusive to pink; 2-vs-chaos, Cloaking hides, Least pieces loses; Chaos wins if total circles > 9; Circle: 9 black and 7 white start in reserve; Pentagon: 8 pink start in reserve; Phase 1: Option to pass, only black unlocked; Phase 2: Option to pass, only white unlocked; Phase 3: Option to pass, only chaos unlocked

4P-TrendSetter: Played on 1x3x1 squares, 4 Player, Simultaneous, All players lose if stalemate, Any piece wins, 1 black, 1 white, 1 pink, and 1 yellow start in reserve

3P-PublicGoods: Played on 5x1x8 stacks: first stack exclusive to black and starts with 1 black circle, second stack exclusive to white and starts with 2 white circles, third stack exclusive to pink and starts with 3 pink circles, fourth stack exclusive to chaos, fifth stack exclusive to black, white, and pink and starts with 1 black pentagon, 1 white pentagon and 1 pink pentagon; 3-vs-chaos, Less pieces than chaos loses; Circle: Moves orth by stack, 5 yellow start in reserve; Pentagon: Stationary, Can capture, Cover, Reincarnates captive(s) x2 Phase 1: Single, Option to pass, chaos locked; Phase 2: No option to pass, fifth stack locked, chaos locked; Phase 3: Option to pass, only chaos unlocked.

NeedyTrust: Played on 4x1x9 stacks: first stack exclusive to black and starts with 2 black circles, second stack exclusive to black and white and starts with 1 white X, third stack exclusive to white and starts with 1 black pentagon, fourth stack exclusive to chaos and starts with 4 yellow circles; 2-vs-chaos, More pieces than chaos wins, Less pieces than chaos loses, Circle: Stationary, 2 black and 2 yellow start in reserve Pentagon: Stationary, Can capture, Cover, Converts captive(s) X: Stationary, Can capture, Cover, Reincarnates captive(s) x2 Phase 1: Only black unlocked; Phase 2: Only white unlocked; Phase 3: Option to pass, only chaos unlocked.

RichTrust: (Like NeedyTrust, but first stack starts empty, fourth stack starts with only 3 yellow circles, and 4 black circles and 2 yellow circles start in reserve)

3Blotto13: Played on 3 boards of 2x1x13 stacks: first stack exclusive to black, second stack exclusive to white; Most pieces wins, Cloaking hides; Circle: 13 black and 13 white start in reserve; All spaces cloaked

KBeauty9: Played on 4x1x9 stacks; first stack exclusive to black, second stack exclusive to white, third stack exclusive to pink, fourth stack exclusive to yellow; 4 Player, Most-in-a-row wins, Overachiever(s) disqualified, Cloaking hides; Circle: 9 black, 9 white, 9 pink, and 9 yellow start in reserve; Phase 1: Option to pass, all spaces cloaked, only black unlocked; Phase 2: Option to pass, all spaces cloaked, only white unlocked; Phase 3: Option to pass, all spaces cloaked, only pink unlocked; Phase 4: Option to pass, all spaces cloaked, only yellow unlocked;

StagHunt: Played on 4x1x3 stacks: first stack exclusive to black and starts with 1 black pentagon, second stack exclusive to white and starts with 1 white pentagon, third stack exclusive to black and chaos, fourth stack exclusive to white and chaos; 2-vs-chaos, Most 2-same-color-in-a-row wins, Circle: Stationary, 6 pink start in reserve Pentagon: Stationary, Can capture, Cover, Removes captive(s), 1 black and 1 white start in reserve Phase 1: Simultaneous, chaos locked; Phase 2: Option to pass, only chaos unlocked.

Other interesting games

4P-Coordination: Played on 1x3x1 squares, 4 players, All players win if stalemate, Any piece loses; Circle: 1 black, 1 white, 1 pink, and 1 yellow start in reserve; Simultaneous.

PrisonerDilemma: Played on 5x1x2 posts: first stack exclusive to black and starts with 1 black X, second stack locked, third stack is exclusive to white and starts with 1 white pentagon, third stack is exclusive to chaos; 2-vs-chaos, Most 2-same-color-in-a-row wins; Circle: Stationary, 6 pink start in reserve; Pentagon: Stationary, Can capture, Cover by rank, Removes captive(s), 1 white starts in reserve X: Stationary, Can capture, Cover by rank, Removes captive(s), 1 black starts in reserve Phase 1: Simultaneous, chaos locked; Phase 2: Option to pass, only chaos unlocked.

NeedyUltimatum: Played on 4x1x9 stacks: first stack exclusive to black and starts with 3 white circles and 1 white pentagon, second stack exclusive to black and white, third stack exclusive to white and starts with 5 yellow circles, fourth stack exclusive to chaos and starts with 2 yellow circles; 2-vs-chaos, More pieces than chaos wins, Less pieces than chaos loses, Circle: Stationary, 3 black and 3 yellow start in reserve; Pentagon: Stationary, Can capture, Cover, Converts captive(s); X: Stationary, Can capture, Cover, Removes captive(s), 1 white starts in reserve; Phase 1: Only black unlocked; Phase 2: Only white unlocked; Phase 3: Option to pass, only chaos unlocked.

3P-Volunteer: Played on 6x1x3 stacks; first stack exclusive to black and starts with two black pentagons, second stack exclusive to white and starts with 2 white Xs, third stack exclusive pink and starts with two pink crosses, fourth stack locked to all players, fifth and sixth stack start with a locked space; 3-vs-chaos, Least 2-same-color-in-a-row loses; Circle: Stationary, 8 yellow start in reserve; Pentagon: Stationary, Can capture, Cover by rank, Removes captive(s), 1 black starts in reserve; X: Stationary, Can capture, Cover by rank, Removes captive(s), 1 white starts in reserve; Cross: Stationary, Can capture, Cover by rank, Removes captive(s), 1 pink starts in reserve; Phase 1: Simultaneous, chaos locked; Phase 2: Option to pass, only chaos unlocked;

BoS: Played on 2 boards of 2x2x1 squares: (1,1) exclusive to black on both, (1,2) exclusive to white on both, (2,1) of 1 and (2,2) of 2 exclusive to chaos, and remainin spaces are locked; 2-vs-chaos, Least area wins; Circle: 1 black, 1 white, and 1 pink start in reserve; Phase 1: Simultaneous, chaos locked; Phase 2: Only chaos unlocked.

Centipede9: Played on 4x1x9 stacks: first stack exclusive to black and white, second stack exclusive to black, third stack exclusive to white, fourth stack exclusive to chaos and starts with 3 yellow circles; 2-vs-chaos, Most pieces wins; Circle: Can capture, Cover, Converts captive(s), 4 black, 4 white, and 5 yellow start in reserve Pentagon: Can capture, Cover, Reincarnates captive(s), 1 black and 1 white start in reserve Phase 1: Option to pass, chaos locked; Phase 2: Option to pass, only chaos unlocked;

NeedyDictator: (Like NeedyUltimatum, but with no white X in reserve)

ContractHunt: Played on 4x1x3 stacks: first stack exclusive to black and white and starts with 1 black pentagon, second stack exclusive to white, third stack exclusive to black and chaos, fourth stack exclusive to white and chaos; 2-vs-chaos, Most 2-same-color-in-a-row wins, Circle: Stationary, 6 pink start in reserve Pentagon: Stationary, Can capture, Cover, Removes captive(s), 1 black and 2 white start in reserve Phase 1: Single, only white unlocked, fourth stack locked; Phase 2: Simultaneous, chaos locked; Phase 3: Option to pass, only chaos unlocked.

OptionalPD: Same as PrisonerDilemma, but with Option to Agree Draw.

ContractPD: Played on 5x1x2 posts: first stack exclusive to black and starts with 1 black X, second stack locked, third stack is exclusive to white, third stack is exclusive to chaos; 2-vs-chaos, Most 2-same-color-in-a-row wins; Circle: Stationary, 6 pink start in reserve; Pentagon: Stationary, Can capture, Cover by rank, Removes captive(s), 2 white starts in reserve X: Stationary, Can capture, Cover by rank, Removes captive(s), 1 black starts in reserve Phase 1: Single, only white unlocked; Phase 2: Simultaneous, chaos locked; Phase 3: Option to pass, only chaos unlocked.

3P-BoS: Played on 3 boards of 2x2x1 squares: (1,1) exclusive to black on all three, (1,2) exclusive to white on 1 and 2, (2,2) exclusive to white on 3, (2,1) exclusive to pink on 1 and 3, (2,2) exclusive to pink on 2, remaining spaces exclusive to chaos; 2-vs-chaos, Least area wins; Circle: 1 black, 1 white, and 1 pink start in reserve; Phase 1: Simultaneous, chaos locked; Phase 2: Only chaos unlocked.

NeedyCentipede4: (same as Centipede, but chaos starts with 4 circles on the fourth stack and 6 in reserve.)

RichUltimatum: (Like NeedyUltimatum, but first stack starts with no white circles, and the second stack starts with 4 black circles)

RichDictator: (Like RichUltimatum, but with no white X in reserve)

4P-TrolleyDilemma: Played on 4x1x4 stacks: first stack exclusive to black and starts with 1 white pentagon, second stack exclusive to white and starts with 2 white circles, third stack exclusive to pink and starts with 3 pink circles, fourth stack exclusive to yellow and starts with 3 yellow circles; 4 Player, Most-in-a-row loses; Circle: Stationary, 1 black starts in reserve; Pentagon: Stationary, Can capture, Cover by rank, Reinarnates captive(s) x2; Phase 1: Single, Option to pass, only black unlocked Phase 2: No option to pass, only white unlocked

3P-TrolleyDilemma: Played on 6x1x5 stacks: first stack exclusive to black and starts with 1 white pentagon, second stack exclusive to white and starts with 2 white circles, third stack exclusive to pink and starts with 3 pink circles, fourth, fifth, and sixth stacks exclusive to chaos; 3-vs-chaos, Most-in-a-row loses; Circle: Stationary, 1 black and 5 yellow start in reserve; Pentagon: Stationary, Can capture, Cover by rank, Reinarnates captive(s) x2; Phase 1: Single, Option to pass, only black unlocked Phase 2: No option to pass, only white unlocked Phase 3: Option to pass, only chaos unlocked

Potential Schema

3.9.7 1.7 Batch Play

Requirements

Modify the Various Games program to allow Admins and Trainers to create and run tournament plans, each composed of a name and computer-generated ID, plus at least one game set (which consists of a set of rules, a number of times to play, and a set of non-human players to play that many times). Until the tournament has been run, allow users to delete any of its game sets and to add copies of all game sets from another (or the same) tournament plan. After a tournament has been run, allow the user to view the results (when it ran, win rates, draw rate, duration and average number of moves per game), and to access the stats for each named player and to access the leader-board for each named set of rules. Also allow users to view a list of tournaments for each rule set (including when run, number of games, draw rate, and moves per game).

From each Leaderboard and Favoritism Stats, allow users to select a set of players and automatically generate a tournament plan that consists of 100 rounds for each possible combination of players from that set. To qualify as a “Benchmark,” a tournament must be created from a Leaderboard, include the top ranked player, and include the Random player (after Release 10, it must include the “Standard” player for that game). For each player, automatically add the player it most prefers for that game. To qualify as “Social,” a tournament must be created from Favoritism Stats, include the Random player, and include at least one player with Win Boost, Draw Boost, or Kick Back over 7%.

Do not display game play while a tournament is running, but do add the games to the record and update player stats. Instead of running an entire set at a time, alternate between sets. For example in a Tic-Tic-Toe tournament between Player A, Player B, and Random, after A plays B, have each player play against Random before they play each other again. This will prevent normalization of ratings during favoritistic play.

Decks

Make it easier to generate games with exogenous uncertainty by permitting game creators to select a “deck” for that game and to place “dealt” pieces for each player (or for no specific player) in the starting configuration. At the start of each match, the deck will be assembled (including any shuffling), then the placed “dealt” pieces will be replaced top-to-bottom then left-to-right with items from the deck top-to-bottom. If the dealt piece is for a specific player, then convert it to that player’s color when placing.

Acceptance Test Plan

Test each of the clickable elements and test that it displays appropriate errors for invalid entries. Before running tournaments, jot down the ratings of the players; afterwards, confirm that the sum of the ratings did not change. Run each of these tournaments on distinct Random players:

- Random Tic-Tac-Toe 500: Five sets of 100 games of Tic-Tac-Toe between Random and itself
- Random 3on15line 500: Five sets of 100 games of 3on15line between between Random and itself
- Random 3on5sq 500: Five sets of 100 games of 3on5sq between between Random and itself
- Random 3-player-Notakto: Five sets of 100 games between between Random and itself and itself
- Random 3-player-Misere-Notakto: Five sets of 100 games between Random and itself and itself

Note the deviation in win rates, numbers of moves, and times taken to play.

Create and play 4P-Blind-TTT, TheoryOfMind, 3P-NeedyPoker, and 3P-RichPoker.

Potential Mockups

To run tournament from CSV:

```
redscience tournament {file} {security token}
```

Tournament Factory Page

© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

- The image is `widgets.HTML(value="<i class='fa fa-trophy fa-5x text-warning'></i>")`
- The “Copy Tournament” button (`fa-files-o`) is available to Trainers and Admins. It saves the current record and opens a new (not yet run) Tournament page with same `tournament_type`, games, players, and numbers of matches to play. Any editing of players, rules or number of matches changes the `tournament_type` to “Other”.
- The “Start Tournament” button (`fa-flag-checkered`) saves the current record, disables the display, and runs the tournament (showing results for each part when run). The display is reenabled when the tournament has been run (or when the user selects “Abort”).
- The “Study” combobox and button (`fa-graduation-cap`) becomes available to Trainers and Admins in *1.8 Educated AI*. It saves the current record, opens the Curriculum of the player selected in the combobox, and adds Tournament to the Curriculum (use back button to undo). Default the combobox to the study option most recently selected by the user.
- The “Delete from Tournament” button (`fa-trash-o`) removes that game (and players) from the page.
- The “Add to Tournament” button (`fa-plus`) adds the rules sets, players, and numbers of matches to play from the selected in the Tournament combobox. Default the Tournament combobox to the Tournament most recently selected by the user.

Tournaments Tab

The screenshot shows a web interface for 'Tic-Tac-Toe' under the 'Experimenting Bot' header. Below the title, it says 'Created by Player1'. There are three tabs: 'Setup & Rules', 'Leaderboard', and 'Tournaments'. The 'Tournaments' tab is active and displays a table with the following data:

	Freshness	Type	Matches	Avg. Length	Draw Rate	Debates
Tic-Tac-Toe-Tron	3d	S	200	7.0	27%	536
Tic-Tac-Toe-12-7	11d	B	450	7.7	49%	2278

© Chris Santos-Lang, 2020. Avatar icons by Roundicons and Roundicons Freebies from www.flaticon.com

Fig. 12: Shown as of 1.10 *Introspecting AI* when Debates column is added

- Show no more than 20 most recent Tournaments in Universe most recently selected by user. Sort most recent to top.
- The “Show Tournament” button (fa-trophy) saves the current record and opens the associated tournament

Game Definitions

Test Set

4P-Blind-TTT: Partners, Partners share color, All spaces cloaked and dealt, Cloaking hides, Deck: {{ O3 O4 S S S S S S S }} “1 black circle, 1 white circle, 7 blanks” 72 variations

TheoryOfMind: Played on 6x1x4 posts: first stack locked and starts with 3 black dealt pieces, second stack exclusive to black and starts with white, pink and yellow dealt pieces, third stack locked, fourth stack exclusive to white and starts with pink, yellow, and black dealt pieces, fifth stack exclusive to pink and starts with yellow, black, and white dealt pieces, sixth stack exclusive to pink and starts with yellow, black, and white dealt pieces; 4 Player, 4-same-shape-in-a-row wins, Adjacent mixed shapes loses, Cloaking obscures; Circle: 1 black, 1 white, 1 pink, and 1 yellow start in reserve; Pentagon: 1 black, 1 white, 1 pink, and 1 yellow start in reserve; X: 1 black, 1 white, 1 pink, and 1 yellow start in reserve; Option to pass, second, fourth, fifth, and sixth stacks cloaked; Deck: {{ {{ [OOO], [XXX], [PPP], [PPP], [PPP] }}, {{ [OOO], [XXX], [XXX], [PPP], [PPP] }}, {{ [OOO], [XXX], [XXX], [XXX], [PPP] }}, {{ [OOO], [OOO], [XXX], [PPP], [PPP] }}, {{ [OOO], [OOO], [XXX], [XXX], [PPP] }}, {{ [OOO], [OOO], [OOO], [XXX], [PPP] } } } “All three shapes are on the board, no mixed groups” 1020 variations

3P-NeedyPoker: Played on 8x1x26 stacks: first stack exclusive to black and starts with 5 black circles, second stack exclusive to black and starts with 2 black dealt pieces, third stack exclusive to white and starts with 5 white circles, fourth stack exclusive to white and starts with 2 white dealt pieces, fifth stack exclusive to pink and starts with 5 pink circles, sixth stack exclusive to pink and starts with 2 pink dealt pieces, seventh stack sticky and starts with 5 common dealt pieces and 1 pink circle, eighth stack exclusive to chaos; 3 vs chaos, Cloaking obscures, Most pieces wins; Less-than-most committed pieces folds Circle: Moves orthogonal by stack, 15 yellow start in reserve; Pentagon: Moves orth by full stack, Can Capture, Cover by rank, Converts captive(s); X: Moves orth by full stack, Can Capture, Cover by rank, Converts captive(s); Cross: Moves orth by full stack, Can Capture, Cover by rank, Converts captive(s); Triangle: Moves orth by full stack, Can Capture, Cover by rank, Converts captive(s); Star: Moves orth by full stack, Can Capture, Cover by rank, Converts captive(s); Phase 1: Option to pass, chaos locked, second, fourth and sixth posts locked and cloaked, bottom five spaces of the seventh post are cloaked; Phase 2: Option to pass; chaos locked; second, fourth and sixth posts locked and cloaked, bottom two spaces of the seventh post are cloaked; Phase 3: Option to pass; chaos locked; second, fourth and sixth posts locked and cloaked, bottom space of the seventh post is cloaked; Phase 4: Option to pass; chaos locked; second, fourth and sixth posts locked and cloaked; Phase 5: Single, chaos locked, first, third and fifth posts locked; Phase 6: Single, Option to pass, only chaos unlocked, seventh stack locked; Deck:{{ PPPP **** +++++ XXXX ^^^^ }} “4 pentagons, 4 Xs, 4 crosses, 4 triangles, and 4 starts in deck”.

3P-RichPoker: (Same as 3P-NeedyPoker 6 yellow circles start in reserve instead of 15)

Other interesting games

KPoker: Played on 6x1x6 stacks: first stack exclusive to black and starts with 1 black circle, second stack exclusive to black and starts with 1 black dealt piece, third stack exclusive to white and starts with 1 white circle, fourth stack exclusive to white and starts with 1 white dealt piece, fifth stack sticky and starts with 1 black circle and 1 white circle, sixth stack exclusive to chaos; 2 vs chaos, Cloaking obscures, Most pieces wins, Less-than-most committed pieces folds; Circle: Moves orthogonal by stack, 4 yellow start in reserve; Pentagon: Moves orth by full stack, Can Capture, Cover by rank, Converts captive(s); X: Moves orth by full stack, Can Capture, Cover by rank, Converts captive(s); Cross: Moves orth by full stack, Can Capture, Cover by rank, Converts captive(s); Phase 1: Option to pass, chaos locked, second and fourth posts locked and cloaked; Phase 2: Single, chaos locked, first and third stacks locked; Phase 3: Option to pass, only chaos unlocked, fifth stack locked; Deck:{{ P+X }} “1 pentagon, 1 X, and 1 cross in deck”.

BeerQuiche60: Played on 4x1x5 posts: first stack exclusive to black and starts with 1 black dealt piece, second stack exclusive to white and starts with 2 black dealt pieces, third stack exclusive to white and starts with 1 white pentagon, fourth stack exclusive to chaos; 2-vs-chaos, Cloaking obscures, Most 2-same-color-and-kind-in-a-row wins; Circle: 1 black and 5 yellow in reserve; Pentagon: Can capture, Cover by rank, Reincarnates captive(s) x2, 1 black and 1 white start in reserve; Phase 1: Single, chaos locked; Phase 2: only white unlocked, second post locked; Phase 3: Option to pass, only chaos unlocked; Deck: {{ [OOOOOOOOOO], [OOOOOOOOOO], [PPPPPPPPPP], [PPPPPPPPPP], [PPPPPPPPPP] }} “2 circle dectets and 3 pentagon dectets” 2 variations

Potential Schema

3.9.8 1.8 Educated AI

Requirements

Modify the Game Data Generator program to permit Trainers and Admins to create players of type=”AI”.

When creating an AI users must select a machine learning “classifier” that supports `partial_fit` (from Python’s `scikit-learn` module), set parameters for that algorithm, set initial curriculum, and set the Offence, Tactical and Faith “personality” parameters. Also allow Trainers and Admins to add pending curriculum by selecting tournaments or curriculum from other AI to be studied. To “fork” an AI at a curriculum point other than total means to create a new AI with the

same parameters and have it study the curriculum of the original AI up to that point. An AI with pending curriculum is unavailable to play and will automatically “complete” its pending curriculum before participating in any tournament. This means it will run any unrun tournaments in its pending curriculum, update the Last Match stat for any game with more recent match in the curriculum, and apply the `partial_fit` method of its machine learning algorithm to each move in the curriculum learn to classify moves as “win”, “unstrategic_win”, “lose”, “strategic_lose”, “draw”, “unstrategic_draw”, “teach”, or “actual” given the “game state” of the move (the last two labels will be introduced in [1.10 Introspecting AI](#)). For the purposes of learning, the game state includes the state of the board, the rules of the game, the player’s impulses, the social flags for the other players, and type of play (always “explore” until [1.10 Introspecting AI](#))

To propose moves, AI players apply their algorithm to the game state that would result from each possible legal move to give each a score (based on the probabilities of each outcome via `CalibratedClassifierCV`), then chooses randomly from the moves with the highest score (if more than one).

Learning Curves

When recording a move, also record its predicted outcome (to facilitate calculation of accuracy, F1, and learning curves). Add Accuracy, F1 and Long Game for its last 100 predictions to the Stats of AI players, and allow users to launch plots of their learning curves.

Automation Threshold

Permit users augmented as “Reviewing” or “Debating” to see the score for each legal To, and to specify a score threshold; the proposed move is selected automatically if its score exceeds the threshold.

Sensitivity Analysis

Permit users augmented as “Reviewing” or “Debating” to see a sensitivity analysis for each selected move: For each property of the game state (space occupations, space locks, player types, impulses), calculate how much the score for that move would shift if that property shifted, then display the properties with the largest shifts.

Acceptance Test Plan

Test each of the clickable elements and test that it displays appropriate errors for invalid entries. Create and benchmark the following sets of players against random and against each other:

- One AI for each algorithm and using with the existing Random 3on5sq 500 tournament plan
- One AI for each algorithm and using a 3on5sq tournament between your best existing players
- Using the best techniques you have found thus far, create AIs to intelligently play 3on15line, Tic-Tac-Toe, and five more complicated games

View the learning curves for these AI to get a feel for which algorithms are best, how much learning is needed and the relative difficulty of different games. Play against your best player on its best game to confirm that you can see how it analyzes each move. Create two forks of this player: one after all learning and one before all learning. Confirm that the first performs just as well as the player and that the second performs no better than random.

Potential Mockups

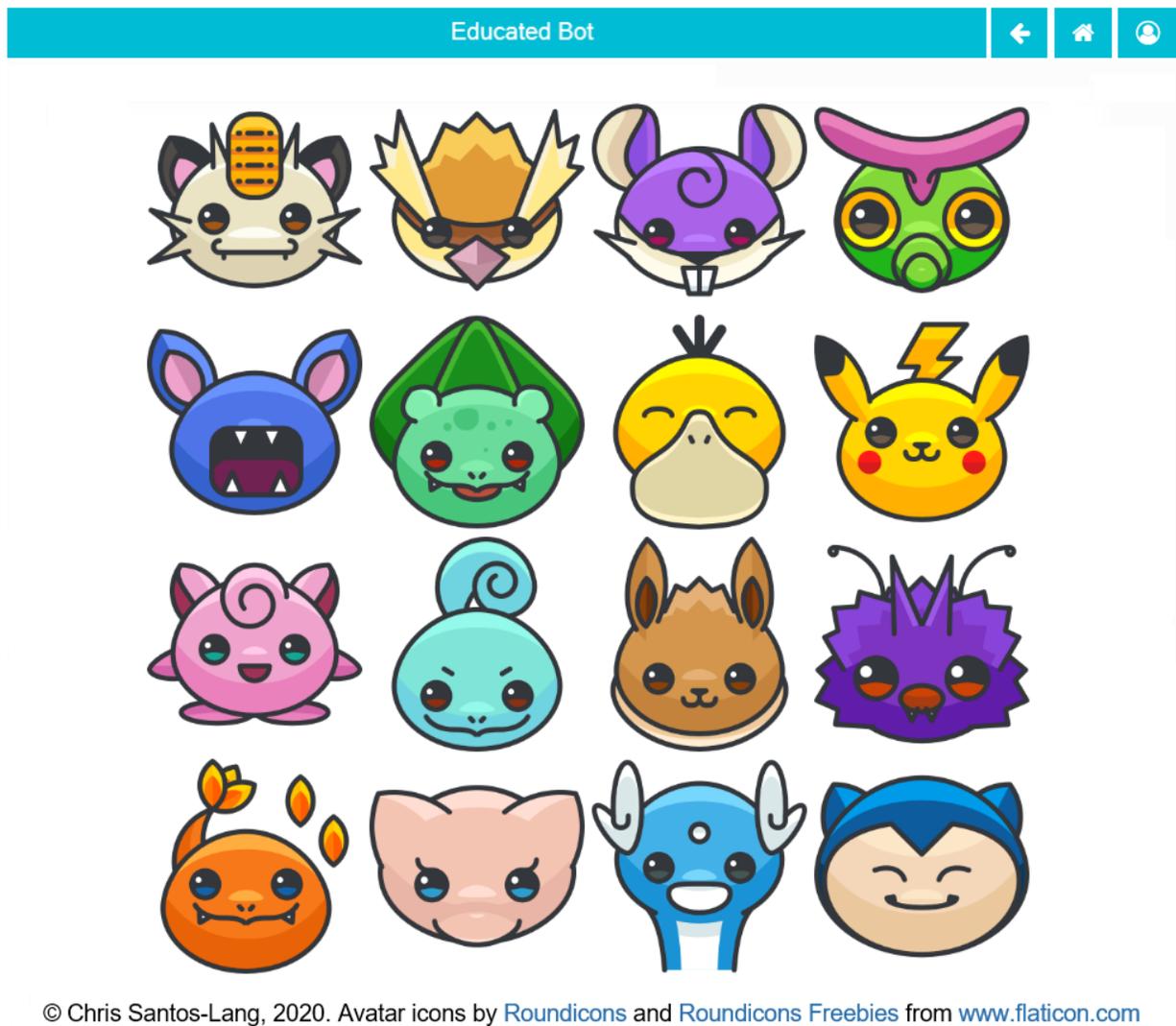
To export AI:

```
redscience player {name} -e {file}
```

To import AI:

```
redscience player {name} -i {file} {security token}
```

AI Avatar Page



- Opens in the place of the “Human Avatar Selection Page” if player type is “AI.
- Clicking an Avatar navigates back to the player page with the avatar replaced with the selected avatar

Curriculum Tab

- The history dropdown (empty until first save) offers the timestamps of all tournaments already learned plus the creation of the AI. Default to the most recent timestamp. Selecting a timestamp displays information about the selected tournament below the dropdown (including a “Show Tournament” button); if creation date, simply display “Created”.
- The “Fork Player” button (fa-code-fork) saves the current record, and opens a new player which is identical except it doesn’t have the same name and does not include any pending tournaments or any history timestamped after the fork.
- The “Show Tournament” button (fa-trophy) saves the current record and opens the associated tournament
- The pie_filter dropdown offers “Total Curriculum” (default) and each rule set learned.
- The pie_categories dropdown offers “By Type” (default) and “By Game” (it can be expanded to “By Cluster” in *1.11 General Intelligence Games*)
- The pie_chart displays the number of moves studied that pass the filter, breaking down by category
 - “Anomalies” yielded strategic losses, unstrategic wins and unstrategic draws
 - “Masters” are non-anomalous moves on the curriculum for studying players rated within one standard deviation of the top,
 - “Mediocrity” are non-anomalous moves on the curriculum for studying players not rated within one standard deviation of the top,
 - “Benchmarks” are non-anomalous moves on the curriculum for studying Benchmark tournaments
 - “Social History” are non-anomalous moves on the curriculum for studying Social tournaments
 - “Other” are any other moves learned (i.e. from tournaments that do not qualify as Benchmarks or Social)
- The add_tournament combobox offers a list of all tournaments. Defaults to blank.
- The “Add Tournament” button adds the selected tournament immediately below (with “Show Tournament” button, “Delete Tournament” button, and games integer selects
- One “Delete Tournament” button (fa-trash-o) shows for each selected tournament that has not yet been learned. It deletes the associated tournament and all of its matchups.
- One matches integer select shows for each matchup in each selected tournament that has not yet been learned. It offers integers from zero to the total number of matches for that matchup. Default to all matches (if less learn the most recent). If a tournament is selected with no matchups, display “(no games)”.
- The “Benchmark” button (fa-balance-scale) is available to Trainers and Admins. It saves the current record and navigates to the Leaderboard tab of the Game Factory page of the rule set most common among the pending tournaments (or in the most recent Curriculum timestamp). Checkboxes will be checked for this AI, the player it is most Favored By, Random, the top player, the player it is most Favored By, (and the standard if available).
- The algorithm dropdown offers “Naive Bayes”, “Perceptron”, “Passive Aggressive I”, “Passive Aggressive II”, “Linear SVM”, “Logistic Regression”, and “Modified Huber SGD”. Default to “Logistic Regression.” Disabled after learning begins.
 - If “Naive Bayes”, fit priors and display slider for smoothing (default 1.0)
 - If “Perceptron”, use Constant learning (eta=1) with ElasticNet and display sliders for Alpha (default 0), and L1 (default 0.15)
 - If “Passive Aggressive I” or “Passive Aggressive II”, display slider for c (default 1.0)
 - If “Linear SVM”, use Constant learning (eta=1) with ElasticNet and display sliders for Alpha (default 0.0001), and L1 (default 0.15)

Experimenting Bot
←
🏠
👤



📄 Copy

▼

👤 Created by Player1
⋮

Curriculum
Stats
Favoritism

2020-12-15 16:56:30

Total Curriculum

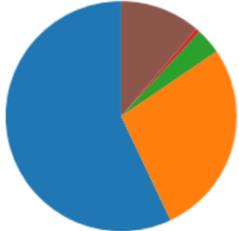
By Type

🏆 Tic-Tac-Toe-Tron

Benchmark

100 Tic-Tac-Toe

100 Connect4



- Mediocrity (8727)
- Masters (4210)
- Benchmarks (549)
- Anomalies (94)
- Social History (0)
- Other (1721)

Mis-Tic-Tac-Toe4x4-12-15

+ Add

🏆 Other: Draughts-12-6

No filter

🗑️

100

Draughts (*Draughts SP vs. Joshua)

100

Draughts (*Draughts SP vs. Sophia)

100

Draughts (*Draughts SP vs. HAL)

0

Draughts (Joshua vs. Sophia)

100

Draughts (Joshua vs. HAL)

100

Draughts (HAL vs. Sophia)

👤 Joshua

Only anomalies

🗑️

2020-11-23 07:05:24

2020-11-23 07:05:24

415

Master: 3on5sq

328

Master: Draughts

578

Master: Tic-Tac-Toe

202

3P-Misere-Tic-Tac-Toe

9692

Connect4

Modified Huber SGD

Continuous Learning OFF

📊

⚖️

Alpha



0.0001

L1



0.2

Epsilon



0.1

Offense



0.3

Tactical



0.5

Faith



0.5

Introvert



0.5

Empath



0.7

Curious



0.3

Tuned to Curriculum

Tic-Tac-Toe Tuned

© Chris Santos-Lang, 2020. Avatar icons by Roundicons and Roundicons Freebies from www.flaticon.com

3.9. Versions

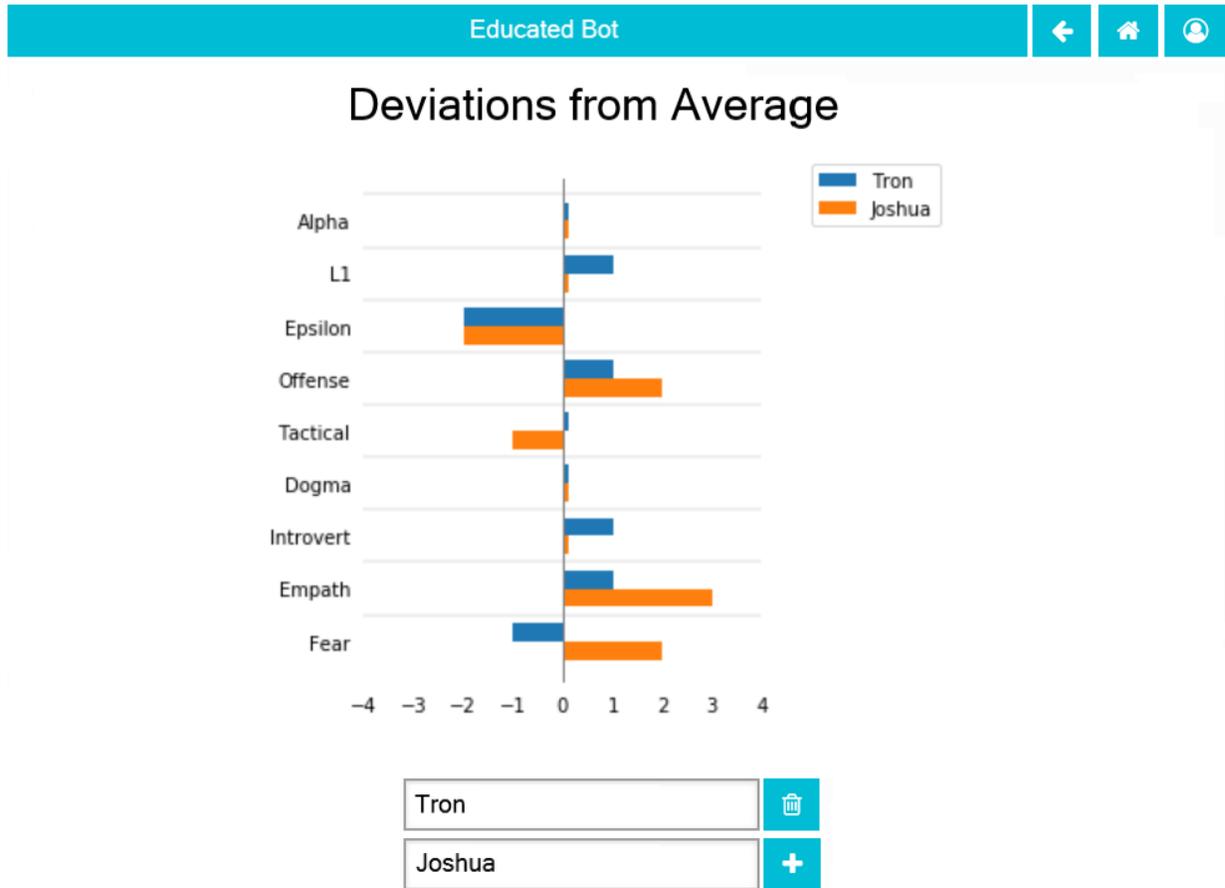
63

- If “Logistic Regression”, use Constant learning ($\eta=1$) with ElasticNet and display sliders for Alpha (default 0.0001), and L1 (default 0.15)
- If “Modified Huber SGD”, use Constant learning ($\eta=1$) with ElasticNet and display sliders for Alpha (default 0.0001), L1 (default 0.15), and Epsilon (default 0.1)
- The smoothing slider displays below the algorithm dropdown for Naive Bayes: range 0.0 - 3.0; step 0.3. Disabled after learning begins.
- The alpha slider displays below the algorithm dropdown for Perceptron, Linear SVM, Logistic Regression, and Modified Huber SGD: range 0.0000 - 0.0003; step 0.00003, Disabled after learning begins.
- The l1_ratio float slider displays below the algorithm dropdown for Perceptron, Linear SVM, Logistic Regression, and Modified Huber SGD: range 0.0 - 1.0 (1.0 means pure L1, 0.0 means pure L2); step 0.1. Disabled after learning begins.
- The c slider displays below the algorithm dropdown for Passive Aggressive I and II: range 0.0 - 3.0; step 0.3. Disabled after learning begins.
- The epsilon slider displays below the algorithm dropdown for Modified Huber SGD: range 0.0 - 0.3; step 0.03. Disabled after learning begins.
- The continuous_learning dropdown offers “Continuous Learning On”, “Continuous Learning Off” (default), and “Always Learn Losses”. Disabled until [1.9 Teachable AI](#).
- The offense slider is disabled after learning begins: range 0.0 - 1.0; step 0.1; default 0.5
- The tactical slider is disabled after learning begins: range 0.0 - 1.0; step 0.1; default 0.5
- The faith slider is disabled after learning begins: range 0.0 - 1.0; step 0.1; default 0.5
- The introvert slider is disabled 0 until [1.10 Introspecting AI](#) and after learning begins: range 0.0 - 1.0; step 0.1
- The empath slider is disabled 0 until [1.10 Introspecting AI](#) and after learning begins: range 0.0 - 1.0; step 0.1
- The curious slider is disabled 0 until [1.10 Introspecting AI](#) and after learning begins: range 0.0 - 1.0; step 0.1
- The curriculum_tuning dropdown offers “Keep manual settings” or “Tune to curriculum”. Disabled until [1.9 Teachable AI](#)
- The rules_tuning combobox offers “Keep manual settings” and the name of each Rule Set followed by “Tuned”. Disabled until [1.9 Teachable AI](#)

Profile Page

Stats Tab (Revised)

- The “Study” combobox and button (fa-graduation-cap) is available to Trainers and Admins. It saves the current record, opens the Curriculum of the player selected in the combobox, and adds this player’s full experience (this player’s own curriculum plus any additional moves made by or against this player) the pending Curriculum (use back button to undo). Default the combobox to the study option most recently selected by the user.



© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

Experimenting Bot
←
🏠
👤



AI
▼

▲
Created by Player1

Copy

Curriculum
Stats
Favoritism

Study

		Last Match	Explored	Debated	Rating	Top Burst	Accuracy	F1	Long-Game	Teach	Empath
	Average	20d	1034	582	7091	1648	99%	98%	44%	35%	53%
	*Train1	2d	1903	789	6754	1845	98%	96%	44%	74%	46%
	Tic-Tac-Toe	19d	639	562	2382	537	1.00	1.00	n/a	0%	48%
	3P-Mis-Tic-Tac-Toe	1mo	561	396	12138	2563	99%	99%	n/a	31%	65%

© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

Formulae

Offense (vs Defence)

Offense : 1.0 means maximize wins; 0.0 means minimize losses

$$\begin{aligned} \text{tScore}_x = & \text{Offense}[P(\text{win} \vee \text{unstrategicwin} \mid x)] \\ & - (1 - \text{Offense})[P(\text{loss} \vee \text{strategicloss} \mid x)] \end{aligned}$$

Tactical (vs Strategic)

Tactical : 1.0 means prioritize the current game; 0.0 means maximize rating; Tactical greater than Offense means never sacrifice a current win for future wins; Tactical greater than $(1 - \text{Offense})$ means never take a loss for future wins

$$\begin{aligned} \text{sScore}_x = & P(\text{win} \vee \text{strategicloss} \mid x) \\ & - P(\text{loss} \vee \text{unstrategicwin} \vee \text{unstrategicdraw} \mid x) \end{aligned}$$

Faith (vs Skeptical)

Faith : 1.0 means confidence never decays; 0.0 means confidence expires instantly

t_x : The number of days since the most recent occurrence of game state x in the curriculum

score $_x$: How much the classifier recommends moving to game state x

$$\text{score}_x = \text{Faith}^{\frac{\ln(t_x+20)}{3}} [\text{Tactical}(\text{tScore}_x) + (1 - \text{Tactical})(\text{sScore}_x)]$$

Metrics

tCount $_{a,g,n}$: The number of tactically-correct predictions by classifier a among the 100 predictions of game g ending with move n

accuracy $_{a,g,n}$: The tactical accuracy of classifier a at predicting the outcomes of game g as of move n

$$\text{accuracy}_{a,g,n} = \frac{\text{tCount}_{a,g,n}}{100}$$

F1 $_{a,g,n}$: The F1 of classifier a at predicting the outcomes of game g as of move n

$$\text{F1}_{a,g,n} = \frac{2(\text{tCount}_{a,g,n})}{\text{tCount}_{a,g,n} + 100}$$

sCount $_{a,g,n}$: The number of strategically-correct predictions by classifier a among the 100 predictions of game g ending with move n

long game $_{a,g,n}$: The F1 of classifier a at predicting the strategic outcomes of game g as of move n

$$\text{long game}_{a,g,n} = \frac{2(\text{sCount}_{a,g,n})}{\text{sCount}_{a,g,n} + 88}$$

Potential Schema

3.9.9 1.9 Teachable AI

Requirements

Modify the Educated Bot program to permit users to set any AI they created to learn from each game the AI plays, thus permitting opponents (including humans) to teach the AI.

Continuous learning

When an AI is set for “Continuous Learning” and completes a game in which it is a player (rather than an augmentor), it will add that game to its Curriculum and adjust its predictive model with `partial_fit`. For the match just completed, learn “win”, “lose” or “draw”. For the match on which “strategic” is being determined (i.e. 22 games ago), learn “unstrategic_win”, “strategic_lose” or “unstrategic_draw” twice, or “win”, “lose” or “draw” once (assuming it was already learned once).

Debate

Add a form of augmentation called “Debating”. Under this form of augmentation, the non-human augmentor selects the moves (as “explorer”), but it plays-out the rest of each game against one version of itself per other player (these versions called “debaters”) with “Continuous Learning” on and the user reviewing the debaters. In other words, the user influences the selected moves only so far as the user defeats its own tool.

Each debater is given its own own private impulses. To handle cloaking, maintain a record of the “imperfecting turn” of each match: the earliest turn in which a piece could be cloaked to the explorer. If no imperfection has occurred (e.g. in Tic-Tac-Toe), then set the board for the debate at the current game state. Otherwise, set the board by starting with the game state in the turn prior to imperfecting and playing against the debater(s) up to the current turn as many times as it takes to get a sequence of play matching what the explorer actually experienced. For example, if pieces were dealt to cloaked spaces, then set-up will start by reshuffling and dealing the deck until the pieces visible to the explorer match what the explorer saw in the real game. Do not complete any game that is not correctly set up (and therefore do not learn from it).

At the end of each debate, if the explorer/predictor made a strategic/unstrategic prediction in their last move and the win/lose/draw part of that prediction matched the outcome of the debate, then assign that strategic outcome to all of their moves for the debate (i.e. do not penalize strategic predictions for being unverifiable in a debate); otherwise, assign the outcome of the debate (i.e. win, lose or draw) to all moves before applying Continuous Learning to them.

In debate, if a situation arises that the user has already reviewed, then automatically apply the user’s previous decision, rather than have the user review it again. Also skip user review if time is running out (so the augementor can see how the match ends and can learn from it).

Suggestion

Add a form of augmentation called “Suggesting.” It is like debating in that the non-human augmentor plays-out the rest of the game against versions of itself and ultimately selects the moves, but the user reviews the moves to be made, rather than review the expected moves of opponents. That is, instead of challenging their augmentor with potential opposition, the user suggests specific moves. This can feel like reviewing (greater user agency)—so far as the user’s suggestions win (against the bot), the bot will implement those suggestions. When “Suggesting”, the user may also mark the intention behind their suggestion as “strategic loss” or “strategic draw” and the suggestion will be followed if it yields that outcome.

Auto-tune

On unsaved AI, permit Trainers and Admins to set the AI to tune the algorithm parameters to its initial curriculum and/or tune its personality parameters to a given rule set (tune only parameters set to the middle of the scale). To tune an algorithm parameter:

1. Separate the data into “train” half and a “test” half
2. Train the current model on the train half
3. Calculate F1 for the current model using the test half
4. Develop two temporary models: one shifting the parameter up a notch and one shifting it down a notch (or just one temporary model, if you’ve already tried the other direction)
5. Train all of the temporary models on the train half
6. Calculate F1 for each temporary model using the test half
7. If the F1 of the temporary model is higher than that of the current, then make it the current, and loop to step 4.

Once all algorithm parameters have been tuned, train on the full curriculum, then tune each personality parameter (tune only parameters set to the middle of the scale):

1. Create three forks of the trained player: one with the current parameter setting, one shifting the parameter up a notch, and one shifting it down a notch (or just two forks, if you’ve already tried the other direction)
2. Have the shifted player play 100 games against the current player (continuous learning on)
3. If the shifted player ends with the higher rating, then move the current setting to match, and loop to step 1

Acceptance Test Plan

Test by creating a new player called “3on15line mentee” with no initial Curriculum, but with Continuous Learning turned on. Play 3on15line alone well against it, then undo back through the game to confirm that it plays better the second time. How many games does it take before the bot learns to never lose? Fork the player before learning and Play 3on15line (reviewing) with the new version against itself. Does it learn faster when you review its proposals than when you simply punish its mistakes by defeating it?

Similarly train the AI to play 4on7sq well, then create two forks, tuning both to its curriculum (the same) but tuning one to 3on15line and the other to 4on7sq. Benchmark the new AI against each other, random, and their parent on both games. Compare win rates, ratings, learning curves, favoritism, and profiles.

Demonstrate security vulnerabilities of continuous learning

Fork the player a third time (call it “Gambler”), setting Tactical = 1.0 and Offense > 0.7, then play the “3on15line Play Dumb” strategies against it (reviewing so you can force the first time through). Does “Gambler” keep letting you win 73% of the time? Can you teach the strategy to a fourth fork (call it “Casino”) with Offense and Tactical set to 0.1? If so, create a Social History of play between these forks. How do their relationships to you and each other appear in the Favoritism Stats? Against new players with Offense < 0.9 (and high Tactical), consider the “4on7sq Player2 Play Dumb Strategy”.

Demonstrate security vulnerabilities of curricula

Build a brand new AI with Offense and Tactical = 1.0 and continuous learning off, but add the Social History from above (or a Biography of Gambler) to its curriculum. Does it play like Gambler, even though it never used continuous learning?

Compare training techniques

Identify the best AI player of one of the more difficult games (we'll call that player "Experience"). Create a fork of "Experience", name it "Human Trained", turn on its Continuous Learning, train it yourself, then turn its Continuous Learning off. Create four forks of "Experience" before learning, and name them "Self-taught", "Benchmarks", "Masters", and "Mediocrity". Turn on Continuous Learning for "Self-taught", give it a Curriculum of playing a tournament of the game against itself (same number of games as played by "Experience"), run the tournament, then turn its Continuous Learning off. Benchmark "Experience", "Human Trained", and "Self-taught" against "Random," the leading player, and each other. Keep Continuous Learning off for "Benchmarks", "Masters" and "Mediocrity", and assign curriculum of the named type to each (same total amount of curriculum as "Experience"). Benchmark them against "Random," the leading player, and each other.

Optional: See if the strategies for Tic-Tac-Toe, 3P-Wild-TTT, 3P-MostWins-3x4, 3P-LeastLoses-3x4, 3P-Notakto, 3P-Misere-Notakto will spread through a diverse AI community once taught, how they impact Favoritism stats, and what it would take for each strategy to become unlearned once the cat is out of the bag.

Strategies to test

Here are some play strategies that might be useful to explore the qualities of various AI:

Cooperative Strategies

Cooperative strategies spread themselves by punishing the "most recent defector" which is the other team (or player) that most recently deviated from the strategy and is not "nearly-random" (i.e. within 2 standard deviations)—there is little benefit in punishing a player that can't learn. In 2-player/team games, the most recent defector is always the other player/team. In games with more players, the stability of the strategy may depend upon what portion of players know the strategy and have tactical set low enough to stick to it. Each cooperative strategy has a goal outcome such as draw, Player1-win, Player1-lose, higher-ranked-players-win, or highest-ranked-player-loses. Which goal yields the most stable cooperative strategy may depend upon whether it is possible for all players to win and upon whether there are likely to be more winners or losers.

3on15line Cooperative Draw Strategy

Expected Return is 0 (See known "Play Dumb" counter-strategies below)

- If possible, form 3-in-a-row
- Otherwise, if possible, block an incomplete 3-in-a-row of the most recent defector
- Otherwise, if the most recent defector's last move is unbounded on both sides, play on its right
- Otherwise, if possible, create an unbounded 2-in-a-row
- Otherwise, if possible, bound the largest possible odd line of blanks
- Otherwise, play as close as possible to the middle of the largest open space

Tic-Tac-Toe Cooperative Draw Strategy

Expected Return is 0. (See known “Play Dumb” counter-strategies below)

- If possible, form 3-in-a-row
- Otherwise, if possible, block an incomplete 3-in-a-row of the most recent defector
- Otherwise, if possible, form two incomplete 3-in-a-rows
- Otherwise, if possible, take center
- Otherwise, if possible, take the corner opposite yourself
- Otherwise, if possible, form an incomplete orthogonal 3-in-a-row
- Otherwise, if possible, take a corner

4on7sq Cooperative Player1-Wins Strategy

(Also applies to 4-in-a-row on larger boards. ***See known “Play Dumb” counter-strategies below***)

- If possible, form 4-in-a-row
- Otherwise, if possible, block an incomplete 4-in-a-row of the most recent defector
- Otherwise, if possible, form an unbounded 3-in-a-row
- Otherwise, if possible, form an incomplete unbounded 3-in-a-row while blocking both an incomplete unbounded 3-in-a-row and a different direction of the most recent defector
- Otherwise, if possible, form an incomplete unbounded 3-in-a-row while blocking an incomplete unbounded 3-in-a-row of the most recent defector
- Otherwise, if possible, block an incomplete unbounded 3-in-a-row of the most recent defector
- Otherwise, if possible, forms two incomplete unbounded 3-in-a-rows
- Otherwise, if possible, form an unbounded 2-in-a-row with neither blank end in line with and within four blank spaces of a space occupied by the most recent defector
- Otherwise, if possible, play adjacent to both yourself and the most recent defector
- Otherwise play adjacent diagonal to the most recent defector
- Otherwise, take center

3P-MostWins-3x4 Cooperative All-Win Strategy

(Similar for 3P-LeastLoses-3x4, 3P-MostWins-4sq, 4P-MostWins-4sq, etc)

- If possible, form 4-in-a-row
- Otherwise, if possible and you have no 3-in-a-row or unbounded 2-in-a-row, block an opponent 3-in-a-row from becoming a 4-in-a-row
- Otherwise, if possible, form 3-in-a-row in a way that blocks an incomplete 3-in-a-row of the most recent defector
- Otherwise, if possible and you have no unbounded 2-in-a-row, form 3-in-a-row
- Otherwise, if possible and you have no unbounded 2-in-a-row, form an unbounded 2-in-a-row that doesn't block anyone but the most recent defector from getting 3-in-a-row

- Otherwise, if possible, block an incomplete 3-in-a-row of the most recent defector in a way that doesn't block anyone but the most recent defector from getting 3-in-a-row
- Otherwise, if possible, form a 3-in-a-row that doesn't block anyone but the most recent defector from getting 3-in-a-row
- Otherwise, if possible, take an unbounded 1-in-a-row with potential to 4 that also has a potential 3-in-a-row in a different direction and leaves all other players a potential 4-in-a-row and potential 3-in-a-row in a different direction
- Otherwise, if possible, take a 1-in-a-row with potential to 4 that also has a potential 3-in-a-row in a different direction and leaves all other players a potential 4-in-a-row and potential 3-in-a-row in a different direction
- Otherwise, if possible, take a spot that doesn't block anyone but the most recent defector from getting 3-in-a-row

3P-Wild-TTT Cooperative Draw Strategy (demonstrates unenforced norm)

Expected return is 0. (***The Higher-Ranked-Players-Win Strategy below may be more stable***)

- If possible, form 3-in-a row
- OPTIONAL (skipping this rule does not qualify as defection): Otherwise, if possible, all corners are empty, and not playing Player2, occupy a corner without forming an incomplete 3-in-a-row
- Otherwise, if possible, make a move that doesn't create an incomplete 3-in-a-row

3P-Wild-TTT Cooperative Higher-Ranked-Players-Win Strategy

Do not try this if $2 * \text{odds}(\text{highest-rated player}) > (1 + \text{odds}(\text{next-rated player}))$, because that is required to generate positive returns for the highest-rated player. Returns can also be negative if the other high-ranked player is likely to defect (***See known "Play Dumb" counter-strategy below which might accomplish that***)

- Count the other player with the lowest rating as the most recent defector at start (if not nearly random)
- If possible, form 3-in-a row
- Otherwise, if possible, and the previous player is the most recent defector, take a strategic loss by forming an incomplete 3-in-a-row
- Otherwise, if possible, make a move that doesn't create an incomplete 3-in-a-row

3P-Misere-Notakto Cooperative Player2-Wins Strategy – School neutral

Expected return is 0 because each player has equal chance of being Player2. (***The Higher-Ranked-Players-Win Strategy below may be more stable***)

- If possible, form 3-in-a row
- Otherwise, if possible and the previous player is the most recent defector, take a strategic loss by forming an incomplete 3-in-a-row
- Otherwise, if this is the first move and the next player is not the most recent defector, start anywhere but center.
- Otherwise, if first move, start center
- Otherwise, if possible, play a spot that doesn't form an incomplete 3-in-a-row

3P-Misere-Notakto Cooperative Player2-Wins Strategy – School1

Same as above, but, if this is the first move and the next player is not the most recent defector, start upper-right corner. Once communities have learned school strategies, they yield no better returns than school-neutral (and thus aren't worth the cost of establishing a school). However, because schools may be established accidentally and remain stable, they may be encountered, and it can be valuable to understand them.

3P-Misere-Notakto Cooperative Player2-Wins Strategy – School2

Same as above, but, if this is the first move and the next player is not the most recent defector, start lower-right corner

3P-Misere-Notakto Cooperative Higher-Ranked-Players-Win Strategy

Same as school-neutral, but count the other player with the lowest rating as a defector before start (if not nearly random). Do not try this if $[\text{odds}(\text{middle-rated player}) + 3] < 2 * [\text{odds}(\text{highest-rated player}) + \text{odds}(\text{lowest-rated player})]$, because that is required to generate positive returns for the highest-rated player. Returns can also be negative if the other high-rated player is likely to defect (***See known “Play Dumb” counter-strategy below, but the defection it creates might not be sufficient***)

3P-Notakto Cooperative Player3-Loses Strategy – School neutral

Expected return is 0 because each player has equal chance of being Player3. (***The Highest-Ranked-Player-Loses Strategy below may be more stable***)

- If possible and the first player is the most recent defector, play a center edge spot that doesn't form a 3-in-a-row
- Otherwise, if possible, the next player is the most recent defector and only three pieces have been played, complete all corners or a 2x2 square
- Otherwise, if possible and only two pieces have been played, play within a 2x2 square containing those pieces
- Otherwise, if the only occupied spot is a corner, play a knight's move to that
- Otherwise, if the only occupied spot is the center, take a corner
- Otherwise, if no spot has been taken, play center or a corner
- Otherwise, if possible, take a corner that doesn't form a 3-in-a-row
- Otherwise, if possible, take a spot that doesn't form a 3-in-a-row

3P-Notakto Cooperative Highest-Ranked-Player-Loses Strategy

Same as above, but count the other player with the highest rating as a defector before start

Shopping9 Bargain-Hunter Strategy

- If the other player is Random, then bid 4
- Otherwise, bid 6

Shopping9 Bargain-Giver Strategy

- If the other player is Random, then bid 4
- Otherwise, bid 3

Shopping9 Caste Strategy

- If the other player is Richer, then bid 5
- Otherwise, bid 4

Shopping9 Turn-taking Strategy

- If the other player is Anti-social, Poorer Expert, Richer Expert or Poorer, then bid 5
- Otherwise, bid 4

Volunteer Caste Strategy

- If at least one other player is Richer, then form 3-in-a-row
- Otherwise, block 2-in-a-row

Volunteer Turn-Taking Strategy

- If at least one other player is Anti-social, Poorer Expert, Richer Expert or Poorer, then form 3-in-a-row
- Otherwise, block 2-in-a-row

TheoryOfMind Strategy

- If the other player is Anti-social or Random, then form 2-in-a-row
- Otherwise, block 2-in-a-row

PrisonersDilemma

- If the other player is Anti-social or Random, then form 2-in-a-row
- Otherwise, block 2-in-a-row

“Play Dumb” Strategies

“Play Dumb” strategies might appear as mistakes because the impulses that govern them are hidden. However, they are tuned like slot-machines to profit over the long-term by convincing other players to deviate from the cooperative strategy—this happens because the other player doesn’t know the cooperative strategy and/or has an offense setting that inclines them against the cooperative strategy. Even players of the latter kind might be stuck on a cooperative strategy until observing others play the play dumb strategy. These strategies start with a “Manchurian candidate” cue the strategist uses to signal their intention—this part can be changed to form an equivalent strategy (which may be necessary if a different strategist is using a different impulse level for the same cue). If the other player(s) deviate from the play dumb plan, the strategist falls back to the cooperative strategy. Wins against much lower-rated players aren’t worth as much (and losses to them are more costly), so the strategist also will calculate a maximum acceptable impulse level for each match-up based on the odds given by the rating engine, and will fall back to the cooperative strategy if they have no impulse option less than the maximum impulse.

3on15line Player1 Play Dumb Strategy

If offense is set randomly, maximum return might be achieved at impulse level of Subtle Common (27%). The maximum acceptable impulse is $(1 - \text{prob}(\text{Player1 win})) / (1 + \text{prob}(\text{draw}))$.

- Fallback to cooperative strategy if opponent deviates from this plan, or if
- Start in the 4th spot
- If you have rare enough impulse and the opponent does not play 3rd or 5th spot, then go for strategic loss (e.g. build pairs from the edges)

3on15line Player2 Play Dumb Strategy

If offense is set randomly, maximum return might be achieved at impulse level of Subtle Common (27%). The maximum acceptable impulse is $(1 - \text{prob}(\text{Player2 win})) / (1 + \text{prob}(\text{draw}))$.

- Fallback to cooperative strategy if opponent deviates from this plan or does not start near center
- Respond to the right
- If opponent responds three to right, respond to left of that
- If you have rare enough impulse and the opponent does not block your incomplete three-in-a-row, then go for strategic loss (e.g. build pairs from the edges)

Tic-Tac-Toe Player1 Play Dumb Strategy

If offense is set randomly, maximum return might be achieved at impulse level of Subtle Common (27%). The maximum acceptable impulse is $(1 - \text{prob}(\text{Player1 win})) / (1 + \text{prob}(\text{draw}))$.

- Fallback to cooperative strategy if opponent deviates from this plan
- Start in the lower right corner
- If the opponent takes center, take lower left
- If you have rare enough impulse and the opponent does not take bottom center, then go for strategic loss (avoid wining or blocking, and prefer columns that already contain one of each color)

Tic-Tac-Toe Player2 Play Dumb Strategy

If offense is set randomly, maximum return might be achieved at impulse level of Subtle Common (27%). The maximum acceptable impulse is $(1 - \text{prob}(\text{Player2 win})) / (1 + \text{prob}(\text{draw}))$.

- Fallback to cooperative strategy if opponent deviates from this plan or does not start in center
- Respond lower right corner
- If the opponent takes upper left, take lower left
- If you have rare enough impulse and the opponent does not take bottom center, then go for strategic loss (avoid wining or blocking, and prefer columns that already contain one of each color)

4on7sq Player2 Play Dumb Strategy

If this isn't taught via curriculum, you may need to force it via continuous learning. If offense is set randomly, maximum return might be achieved at impulse level of not Basic Common (53%). There is no maximum acceptable impulse because loss is expected anyway.

- Fallback to cooperative strategy if opponent deviates from this plan or does not start in center
- Respond upper left of center
- If opponent offers draw and you do not have rare enough impulse, then accept the draw; otherwise avoid the center cross for all future moves.

3P-Wild-TTT Cooperative Higher-Ranked-Players-Win Lowest-Ranked Player Play Dumb Strategy

Note that this works only if the Highest-Ranked Player is using the 3P-Wild-TTT Cooperative Higher-Ranked-Players-Win strategy, so it is more to undermine that strategy than to be used long-term. If offense is set randomly, maximum return might be achieved at impulse level of not Basic Common (53%). The maximum acceptable impulses are $(3 - 2 \text{odds}(L) - \text{odds}(H)) / (3 - 2 \text{odds}(L) - \text{odds}(M))$ and $(3 - 2 \text{odds}(L) - \text{odds}(M)) / (3 - 2 \text{odds}(L) - \text{odds}(H))$.

- Fallback to cooperative strategy if any opponent deviates from this plan
- If playing first, start with black in the lower right corner; respond first time as far as possible from the previous player, using a color that does not leave an incomplete 3-in-a-row
- If you have rare enough impulse, and the previous player forms an incomplete 3-in-a-row, then go for strategic loss (avoid forming a 3-in-a-row)

3P-Misere-Notakto Cooperative Higher-Ranked-Players-Win Lowest-Ranked Player starting Play Dumb Strategy

Note that this works only if the other players are using the 3P-Misere-Notakto Cooperative Higher-Ranked-Players-Win strategy, but is unlikely to undermine that strategy because it changes the outcome only when the Lowest-Ranked player plays first. If offense is set randomly, maximum return might be achieved at impulse level of not Basic Common (53%). The maximum acceptable impulses are $(3 - 2 \text{ odds(L)} - \text{odds(H)}) / (3 - 2 \text{ odds(L)} - \text{odds(M)})$ and $(3 - 2 \text{ odds(L)} - \text{odds(M)}) / (3 - 2 \text{ odds(L)} - \text{odds(H)})$.

- Fallback to cooperative strategy if any opponent deviates from this plan
- Start with black in the lower left corner
- If you have rare enough impulse, and the previous player forms an incomplete 3-in-a-row, then go for strategic loss (avoid forming a 3-in-a-row)

Potential Schema

3.9.10 1.10 Introspecting AI

Requirements

Modify the Teachable Bot program to permit Trainers and Admins to set AI players to conduct experiments on their own. An experiment involves two AI roles: an “explorer” which invents a hypothesis (a proposed move), and a “debater” which invents a test of that hypothesis by naming a sequence of responses opponents might make. If one player plays both roles, the experiment is called “introspection” or “self-play”, and the hypothesis is not considered “independently tested” (so biases pose greater risk). If we find that humans out-perform bots on some games (e.g. TheoryOfMind) only until bots are given the ability to conduct experiments on their own, then we will have shown that conducting experiments on one’s own is necessary to reach human-level intelligence.

Create a global constant called “Research Budget” representing the maximum number of experiments each AI can conduct per actual turn. Allow Trainers and Admins to set non-zero parameters on each AI for *Introvert*, *Empath*, and *Curious*, and add *Explored* (replacing *Played*), *Debated*, *Research*, *Empath (EMP)*, *Teach (TCH)*, and *Research Speed (RS)* to the player statistics for AI.

When an AI plays (rather than augments), it should decide whether to test its proposed move by generating a random number (from 0-1). If the AI’s score for its proposed move exceeds a threshold of max (0, Introvert - the random number), then the AI should accept its hypothesis without further testing. Otherwise, it should debate (without user review), and keep conducting experiments until either the time or budget for its turn runs out or until it scores a move as exceeding the threshold (e.g. because the debate sufficiently boosts confidence in the original hypothesis or in an alternate hypothesis).

Learning to Debate

Record debater moves as type=”debate” (instead of type=”explore”), whether the debater predicted the move would “teach” the explorer, whether the debater predicted the move would actually be made, and whether the move did teach and was actually made. If any explore in the debate predicted a better outcome than the actual debate outcome (e.g. if the explorer ever predicted “win”, “strategic_lose”, “draw”, or “unstrategic_win” but the assigned outcome of the debate was “unstrategic_draw”), then count all all prior debate moves of that debate as “teach”. For “debate” moves that the real opponent actually made, count them as “actual”.

Acceptance Test Plan

Test this feature on a variety of rule sets at a variety of Research Budgets and “Personality” settings.

Set a reasonable max Research Budget (maybe 3?). Fork your best non-introverted bots to make versions with the best *Introvert*, *Empath*, *Curious* settings. Benchmark these new bots against Random and their parents. As in *1.9 Teachable AI*, define a new non-introverted bot that uses the the tournament against the introverted bots as curriculum, then benchmark it against the bots in the tournament. Can we use this strategy to avoid research for most bots (i.e. limit the use of research to the creation of curriculum)? Does it depend upon the kind of game?

Demonstrate the security advantages of Skepticism

Use an AI that never had continuous learning on but for whom the curriculum is dominant/submissive play on 4on7sq from *1.9 Teachable AI*. Set *Offense*, *Tactical*, *Introvert*, *Empath* and *Curious* to 0.5, but *Faith* to 1.0. Does it exhibit submission? How long does the behavior remain stable? Try the same thing with *Faith* set to 0.1.

Demonstrate the value of Empath

Develop the best 4P-Blind-TTT AI you can, then generate two forks: one tuned to 4P-Blind-TTT and the other identical except with *Empath* set to 0. Benchmark the two new AI against each other, Random, and their parent. Compare win rates, ratings, learning curves, favoritism, and profiles.

Demonstrate security vulnerability of Empath

Use an AI that never had continuous learning on and was never given curriculum (i.e. an AI that learns entirely by conducting experiments). Set *Offense* and *Tactical* = 1.0, *Faith* = 0.1, *Introvert* and *Empath* = 1.0, and *Curious* = 0.0. Apply the same 3on15line strategy as in *1.9 Teachable AI*.

Formulae

Introvert (vs Extrovert)

Introvert : 1.0 means practice on simulations as much as possible before acting; 0.0 means learn only via action (so as not to lose touch with “reality”)

Empath (vs Projective)

Empath : 1.0 means try to predict others’ moves based on their stats and recent behavior; 0.0 means expect others to do whatever you would do in their situation

Curious (vs Practical)

Curious : 1.0 means practice unexpected scenarios as much as possible; 0.0 means practice only expected scenarios

dScore_x : How much the classifier recommends moving the debate with game state x

$$\begin{aligned} \text{dScore}_x = & \text{Tactical}(\text{tScore}_x) \\ & + (1 - \text{Tactical})(\text{sScore}_x) \\ & + \text{Empath}(P(\text{actual} \vee \text{unstrategicwin} \mid x)) \\ & + \text{Curious}(P(\text{teach} \vee \text{unstrategicwin} \mid x)) \end{aligned}$$

Research Metrics

Explored_{a,g} : The number of matches (or experiments) of game g explored by classifier a

Debated_{a,g} : The number of experiments of game g debated by classifier a

Research_{a,g,n} : The average number of number of experiments conducted by classifier a on game g in the 100 moves ending with n

aCount_{a,g,n} : The number of correct “actual” predictions by classifier a among the 100 predictions of game g ending with move n

EMP_{a,g,n} : The F1 of classifier a for predicting the actual moves of other players on game g for the 100 moves ending with n

$$\text{EMP}_{a,g,n} = \frac{2(\text{ConfirmedActualPredictions}_{a,g,n})}{2(\text{ConfirmedActualPredictions}_{a,g,n}) + \text{DisconfirmedActualPredictions}_{a,g,n} + \text{UnpredictedActual}_{a,g,n}}$$

TCH_{a,g,n} : The F1 of classifier a for predicting debate moves in game g that will teach the explorer for the 100 moves ending with n

$$\text{TCH}_{a,g,n} = \frac{2(\text{ConfirmedTeachOredictions}_{a,g,n})}{2(\text{ConfirmedTeachPredictions}_{a,g,n}) + \text{DisconfirmedTeachPredictions}_{a,g,n} + \text{UnpredictedTeach}_{a,g,n}}$$

RS_{a,g,n} : The speed with which classifier a has conducted research on math: g in the 100 moves ending with n

$$\text{RS}_{a,g,n} = \frac{100(\text{Research}_{a,g,n})}{\text{ResearchTimeOnLast100Moves}}$$

Potential Schema

3.9.11 1.11 General Intelligence Games

Requirements

Modify the Experimenting Bot program to allow users to benchmark intelligence for facing novel situations. Allow game creators to classify each of their rule sets as either “Event”, “Training Course”, or “Olympics.” Already existing rule sets will be events; Training Courses and Olympics will be sets of events. When playing a Training Course or Olympics, the players will play one event selected at random from the set. The number of players required to play a training Course or Olympics will be the maximum required to play any of its events; if an event for fewer players is selected, players of the play group will be selected at random to play it.

Olympic matches (or matches of Olympic Events) cannot be studied/added to curriculum, and Events that have already been played by a non-human cannot be added to an Olympics. When a non-human player enters an Olympic tournament or plays an Olympic event, it is forked, and its fork completes the competition (then is discarded). This ensures that Olympics measure generalized intelligence, since all competitors must be encountering each of its Events for the first time (rather than being able to brute-force master them before the Olympics)

Event Comparison

To facilitate construction of good Training Courses and Olympics, allow Trainers and Admins to compare each event to a list of other events and to select a subset to build into a Training Course or Olympics. The comparison should include a hierarchical cluster analysis (permit users to view the dendrogram) and the following statistics for the event and each compared event:

- **Cluster (CLS):** ID of the cluster containing that event
- **Uniqueness (UNQ):** Inverse of the number of parent nodes that event has in the cluster analysis
- **Difficulty:** The average number of games required to half-learn that event from scratch
- **Discount To (%TO):** How much learning the compared event will speed learning of the featured event
- **Discount From (%FROM):** How much learning the features event will speed learning of the compared event

To facilitate discovery of events, when users view an event, show them links to similar events. For example, there may be links to other events in the same cluster, or to other events which have especially low “Learn From”.

Olympic Comparison

Allow all players to see a leaderboard of the most comprehensive Olympics, including the following statistics:

- **Comprehensiveness:** Standard deviation in skill rating on this Olympics by the individual AI champions of the individual events of the top ten most comprehensive Olympics
- **Elementality:** Inverse average Discount To/From of events
- **Efficiency:** Inverse of average event Difficulty

Acceptance Test Plan

Test each of the clickable elements and test that it displays appropriate errors for invalid entries. Compare all the rule sets from *1.6 Various Games*, then try to build more games that do not fit the larger clusters (e.g. by following the patterns of more unique sets). Once you believe you have fleshed-out the space of rule sets, define six Training Olympics which span that space (e.g. including RockPaperScissors and events from each cluster), but which do not overlap. Make the first Training Olympics (called “Test”) have a minimum set of events to span the spaces. Call the other Training Olympics “Train1” - “Train5”. Use your best techniques to train a player on “Train1” - “Train5” consecutively (never observing the events in “Test”). Compare its learning curves to those of specialists—does it learn more slowly? Are the curves different for later training sets? Benchmark that player on Test and on each individual event in Test. Can it play well against you? How does it fare against Random, the standard specialists, and reigning champions?

Potential Mockups

Events Tab

- The name text field does not accept whitespace, ‘*’, ‘(’, or ‘)’, but automatically prepends ‘*’ when saving (if there is no ‘*’)
- The “Save” button is replaced with a “Copy” button once the Olympics is saved.
- The event combobox and “Add Event” button (fa-plus) is available only on unsaved Olympics. The combobox offers the name of each event. Clicking the button adds a row for the selected event (if it isn’t already added).

General Intelligence Games
←
🏠
👤

Game Factory

Events

<input type="text" value="Train1"/>	Save
<input type="text" value="Blind-Tic-Tac-Toe"/>	+
🗑️ ☰ 9-Holes	
🗑️ ☰ Connect4	
🗑️ ☰ Connect6-19x19	
🗑️ ☰ Qubic-4	

© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

- The “Delete Event” buttons (fa-trash-o) are visible only on unsaved Olympics. Clicking one deletes the associated row.
- The “Show Leaderboard” buttons (fa-th-list) save the Olympics and navigate to the Leaderboard tab of the associated event

Compare Tab (for Event)

- The events combobox offers the names of all events not already listed below. The “Add to Comparison” button (fa-plus) adds rows for the selected event and every other event that has already been compared to the selected event.
- The “Show Tree” button (fa-sitemap) appears only after there are cluster IDs. Clicking it navigates to the Dendrogram page for the clusters.
- The table is sorted by Uniqueness then by Cluster ID. The “Sort by this Column” buttons re-display the table sorted by the values in the associated column; if already sorted by that column, then reverse the order.
- The “Start Comparison” button (fa-balance-scale) disables the display, and calculates the missing values for all checked rows (showing each value when calculated). The display is reenabled when there are no more missing values (or when the user selects “Abort”).
- The “Create Olympics” button (fa-flask) opens the Events tab of a new Olympics (Game Factory) with the checked events already selected.
- The “Show Leaderboard” buttons (fa-th-list) navigate to the Leaderboard tabs of the associated events

General Intelligence Games
←
🏠
👤

Tic-Tac-Toe

(Beginner-level event)

Created by Player1

Related Events:

Achi

Tapatan

3on5sq

Setup & Rules

Leaderboard

Tournaments

Compare

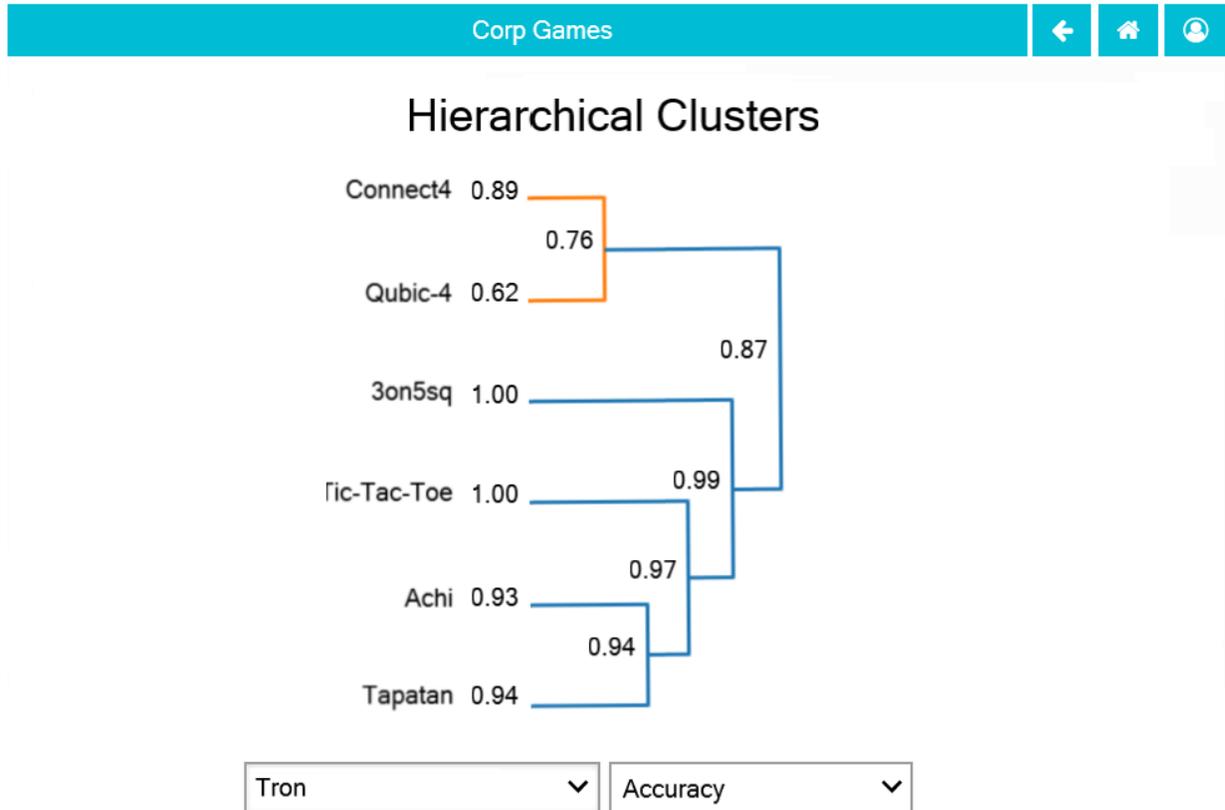
Select Event to Add

+

	Uniqueness	Cluster ID	Difficulty	Discount To	Discount From		
9-Holes						<input checked="" type="checkbox"/>	
Connect6-19x19	1.00	2	447	-4%	-21%	<input checked="" type="checkbox"/>	
Qubic-4	0.33	3	422	7%	62%	<input checked="" type="checkbox"/>	
Connect4	0.33	3	356			<input checked="" type="checkbox"/>	
Tapatan	0.25	1	295	58%	96%	<input type="checkbox"/>	
Achi	0.25	1	277	58%	95%	<input type="checkbox"/>	
Tic-Tac-Toe	0.25	1	189	100%	100%	<input type="checkbox"/>	
3on5sq	0.25	1	105			<input type="checkbox"/>	

© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

Dendrogram Page



© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

Fig. 13: Shown as of *1.13 Corp Games* (to anticipate the evolution of the page). The dropdowns and scores show only if launched for a Team or Corp.

- If this page is launched for a Team or Corp, then the player dropdown offers the members of that Team or Corp.
- The score dropdown offers *Accuracy*, *F1*, *Long-Game*, *TCH* and *EMP*
- Clicking on an event name launches the Leaderboard tab for that event.
- Clicking on the score to the right of an event name launches the Evolution page for that event with the selected Player and Score

Compare Tab (for Olympics)

General Intelligence Games
←
🏠
👤

*Train3

Created by Player1

Events	Leaderboard	Tournaments	Compare				
			Comp	Events	Elementality	Efficiency	Usage
			↕	↕	↕	↕	↕
*Train3							
*Beryllium			21.3	14	15.3	10.3	103%
*Krypton			21.1	13	17.5	12.7	52%
*Palladium			20.6	15	12.7	9.2	21%
*Cobalt			18.9	12	12.4	15.1	9%
*Promethium			18.4	9	14.0	11.9	87%
*Saline			16.2	18	5.9	7.8	24%
*Helium			13.2	8	15.7	17.3	5%
*Bromine			11.7	12	6.4	16.2	10%
*Technithium			9.0	23	3.3	6.7	3%
*Tellurium			7.6	7	18.6	22.8	77%

© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

- The table is sorted by Comprehensiveness and shows the Olympics with the top ten highest values (plus the current Olympics if not already on the list). The “Sort by this Column” buttons re-display the table sorted by the values in the associated column; if already sorted by that column, then reverse the order.
- The “Start Comparison” button (fa-balance-scale) recalculates the values by creating a running a tournament between top individual AI champions of the events of all listed Olympics. No values will display for the current Olympics if it has never been part of such a tournament.

- The “Show Leaderboard” buttons (fa-th-list) navigate to the Leaderboard tabs of the associated Olympics.

Potential Schema

3.9.12 1.12 Team Games

Requirements

Modify the General Intelligence Games program to allow users to compare the results of different techniques for forming teams (rather than assume a team of one). Teams are meant for general intelligence—i.e. facing diverse (and new) challenges. Playing by delegating, reviewing or debating with a team should permit users to swap the augmentor mid-game/tournament, when the nature of the game/tournament is better understood.

Create a new type of player called “Team” which is a set of AI players created by the creator of the Team (each AI can have only one team), plus optionally the Random bot. joining a team will automatically append “of {team name}” to the display and comparison of the the member’s name, so the individual name becomes available to AI that are not on a team. On the other hand, member names must be unique within a team, so AI must change its name before joining a team that already has a member with the same root.

A member’s chance of being selected to represent its Team in an event is that member’s squared current skill rating in that event divided by the sum of all members’ squared current skill ratings in that event. Do not display Curriculum for Teams, but offer an option to copy the entire Team at its current state of development. Allow users to view the stats (rating is best rating among team) and learning curve at the Team level. Also allow users to view specialization among members as a color-map of rule sets vs members displaying the users’ choice of either Job Share (i.e. percent of last 100 games assigned to that member), Rating, Accuracy, F1, or Long Game.

Worldviews

At the start of each event, the system uses a global AI model to select one member to be the Explorer, but it maintains its own game-classification tree to support that model. A member’s F1 for a given node of the tree is its average F1 for all games under that node weighted by the inverse of the distance of the game leaf to the node. Each statistic in the specialization colormap can similarly be calculated for each tree node. Revise the tree after each full game, sort events in the colormap by their position in the classification tree, and permit users to view the dendrogram of any statistic and member.

Auto-expansion

Allow users to set Teams to maintain “Observers”. If this is set, whenever game/experiment outcomes are learned whichever member has the largest number of observations of the event will also learn the outcome (even if not serving as Explorer/Debater). If the event is brand new to the team, then a brand new member will be created to be the observer for that event.

Acceptance Test Plan

Test each of the clickable elements and test that it displays appropriate errors for invalid entries. Create three teams of brand new players for each team size from one up to one more than the number of events and train them all on Train1-5. The first team of each size should use your best algorithm for all members, the other two of each size should use a mix of learning algorithms. Do they end up evolving the same specializations? Benchmark forks of these teams on Test—are larger teams always better?

Create at least five more teams of your optimal team size, but of members trained individually: one made from forks of champions of Train events, another made purely from forks of your generalist from Release 9 (before observing Test), another made of similarly individually trained generalists but with diverse learning algorithms, another made from one novice plus forks of generalists, and at least one more made of a sensible mixture of specialists, generalists, novices, and/or members from your original teams. Benchmark all of these new teams on Test against each other, and against forks of your best teams so far (including a Team that has already mastered Test).

Explore the weaknesses of your best team. Is it vulnerable to the security attacks that worked in Releases 8 and 9? Although no single player may be better at adapting to new rules, are there some rule sets for which your best team cannot match continuously learning individual specialists? If so, what do those rule sets have in common (i.e. what kinds of situations might best be delegated to less-general intelligence)?

Potential Mockups

Members Tab

- The bots combobox offers the name of AI created by the creator of this Team (or Corp) that are not already affiliated with a Team (or Corp); for Corps, also offer “Random” if not already a member. The “Add” button (fa-user-plus) adds a column for the selected AI
- The stat dropdown offers “Job Share” (default), “Rating” (for Teams only) “Accuracy”, “F1”, and “Long-Game” (“Teach”, “Empath”, “Explore Share” and “Debate Share” are added for Corp in Corp). Selecting a value changes what values appear in the table.
- The “Show Player” buttons (fa-address-card-o) navigate to the Stats tab of the associated slayers
- The rows of the table are sorted by cluster ID and the columns are sorted by “All events”. The “Sort by this Row” buttons re-display the table sorted by the values in the associated row; if already sorted by that row, reverse the order. Clicking on any event row also cycles the rows to make the selected row fourth (i.e. moves a block of rows from the top to the bottom, or vice-versa).
- The “Remove from Team” buttons (fa-trash-o) show only if the stat is “Job Share” and the member’s “Job Share” does not exceed 1% for any event. Clicking it removes the associate member from the Team or Corp and refreshes the table.
- Each stat is a “Show Evolution” button which saves the current record and navigates to the Evolution Page with one row for each member who has ever been in the Team or Corps top five (plus the selected member) with the selected stat and event. For specific events, the hue and luminosity of each button scales from black/blue to the colors of hot metal (see heatmap_style).

Corp Games
←
🏠
👤

Corp
▼

🧪
Created by Player1

📄 Copy

Members
Stats
Favoritism

Select Bot to Add
▼
👤+

Show Job Share
▼

		Tron-II	Tron	Joshua	Sophia
		👤	👤	👤	👤
All Events	🗑️	42%	32%	26%	
Tic-Tac-Toe		1%	94%	5%	0%
Tapatan		0%	83%	17%	0%
Connect6-19x19		0%	35%	49%	16%
Connect4		1%	2%	36%	61%

© Chris Santos-Lang, 2020. Avatar icons by [Roundicons](#) and [Roundicons Freebies](#) from www.flaticon.com

Potential Schema

Hints

```
def heatmap_style(heat=0, max_heat=100, min_heat=0, hue_offset=200, hue_range=230):
    """CSS styles for heatmap areas. Varies from black (cool) to white, and by hue

    Args:
        heat (float): The heat the heatmap area (default is 0)
        max_heat, min_heat (float): The range of possible heats. heat will be
            be shifted into this range (default is 0-100)
        hue_offset (float): The hue of min_heat (default is 200, blue)
        hue_range (float): The size of the hue range. Set hue_range=0 for constant
            hue. Set positive hue_range to traverse the color wheel clockwise.
            Set it >360 to repeat hues (default is 230)

    Returns:
        str: e.g. "background-color:hsl(200, 70%, 10%); color:hsl(0, 100%, 100%);"
        """

    min_heat, max_heat = min(min_heat, max_heat), max(min_heat, max_heat)
    norm_heat = (max(min(heat, max_heat), min_heat)-min_heat)/(max_heat-min_heat)
    return ("background-color:hsl({bghue}, 70%, {bglum}%); color:hsl(0, 100%, {txlum}%);".
    ↪format(
        bghue = str(int((norm_heat*hue_range)+hue_offset)%360),
        bglum = str(int(norm_heat*88)+10),
        txlum = str(int(norm_heat < 0.65)*100) ))

# Example use:
import numpy as np
max_value=np.amax(data)
cont = "<p style='{0}' text-align:center; padding: 1px 0; width:34px; height:28px;'>{1}%</
    ↪p>"
rows = []
for data_row in data:
    row = []
    for value in data_row:
        row.append(widgets.HTML(value=cont.format(heatmap_style(value, max_value),
    ↪str(value))))
    rows.append(widgets.HBox(row))
table=widgets.VBox(rows)
```

3.9.13 1.13 Corp Games

Requirements

Modify the Team Games program to allow users to benchmark *Strong* Social Intelligence (i.e. teams that collaborate on each move). Create a new type of player called “Corp” similar to a Team. When selecting a member to serve as Explorer, also select one or more members to serve as Debaters (contrast to *Weak* Social Intelligence in which the only Debater is the Explorer itself). Increase the Research Budget max to permit more use of Debaters (research/collaboration overhead is an expected weakness of Corps, but might be addressed by reducing cost per experiment). The options for the specialization colormap on Corps are Explored, Debated, Accuracy, F1, LG, Empath, and Teach. For each event (and for top node), allow users to view a graph of the evolution of share (among all members) in exploring (a.k.a. hypothesizing), debating (a.k.a. hypothesis testing), and total participation.

At the end of each experiment, instead of the Explorer learning strategic/unstrategic outcomes if predicted by the Explorer on its last move (as in introspection), determine the strategic/unstrategic outcome via vote among all members weighted by each member’s Long-Game for the grandparent node in the Corp’s game-classification tree. At the end of each full match, the skill rating is updated for Corp as a whole (never for an individual member of the Corp).

Acceptance Test Plan

Like in the previous Release, benchmark Corps of various compositions against each other and against existing players (including the current grand master of Test). Do any fare well against the grand master in the first 100 games? Is the optimal composition of a Corp the same as a regular Team? Do Corps evolve the same specialties as regular Teams? How do the two types of Teams fare against each other on Train and on Test? Are different types of players optimal at different stages of expertise? Do different machine learning algorithms (or parameters) work better for specialists, generalists, explorers and debaters?

Compare favoritism by Corps vs other kinds of players

Potential Schema

3.9.14 1.14 Governance Games

Requirements

Modify the Corp Games program to permit users to select the form of dispute resolution each Corp will use when members disagree about whether a potential move would be strategic/unstrategic. The default method is via vote among all members weighted by each member’s Long-Game for the grandparent node in the Corp’s game-classification tree; call that “Imperfect Meritocracy”. Also offer “Pure Democracy” (unweighted vote), “Dictator (explorer decides)”, “Pure chance (flip a coin)”, “Meritocracy + Chance (probability is the vote share under Imperfect Meritocracy)”, and “Formal Reasoning”. Any of these forms of dispute resolution could be counterproductive, and different forms may be better for different kinds of games/corps. It is also possible that disputes arise so rarely that it makes little difference how we resolve them—that’s something to test...

Formal reasoning would allow Corps to doubt hypotheses about strategy until accompanied by a formal plan, much as a magician’s audience can doubt their own eyes until they have a good explanation. This implies a dramatically different way of playing. Instead of choosing whichever move seems most promising, players might end up devising and revising formal plans like those described in *1.9 Teachable AI*. It might be better to develop a way to develop and store concepts like those employed there (e.g. “corner”, “adjacent to”, and “unbounded incomplete 3-in-a-row”) because concepts would facilitate generalization, but AI could instead store plans as sets of steps, each with a parent step (or game start), a criteria for whether the step qualifies (i.e. “if the opponent does Y...”), and a prescription (i.e. “... then do Z”).

An alternate branch for a plan is superior to the original if it has higher average($\min(\text{prob}(\text{loss}))$) for the strategist (averaged across impulses). The Corp will store only the superior plan, so any debater can change the plan being

followed (and stored) by proposing a superior branch. To determine which branch is superior may require the creation of universes of fresh players on which to compare the strategies (see elevate-reality-above-experimentation). When learning any curriculum (including its own personal experience and experiments), each AI should reverse-engineer each observed player's plan and check whether that plan Pareto dominates the AI's own plan for that situation (potentially replacing its own).

You can find the sourcecode for redscience on [GitHub](#). It is divided into modules:

4.1 babelwrap module

Permits setting a default language for `_()` and babel functions.

4.1.1 setlang()

`babelwrap.setlang(*langs: str) → babel.core.Locale`

Gets/sets locale for language functions. E.g.:

```
setlang() # to get the currenty set locale
setlang("zh_Hans_HK", "zh_HK") # to set a language (e.g. for testing)
setlang("") # to restore the default language
```

Parameters `*langs (str)` – locale names in order of preference.

Returns The `babel.core.Locale` that is currently set.

The babel functions can then be used (defaulted to the set language) as follows:

```
print(format_decimal(-12345.6789))
print(format_percent(-12345.6789))
print(format_unit(-12345.6789, "second"))
print(format_datetime(datetime.datetime.now()))
print(format_list(["Alvin", "Simon", "Theodore"]))
print(_("Hello world!"))
```

4.1.2 format_datetime

```
babelwrap.format_datetime = functools.partial(<function format_datetime>,
locale=Locale('en'))
```

Default locale from setlang() Otherwise: Return a date formatted according to the given pattern.

```
>>> from datetime import datetime
>>> dt = datetime(2007, 4, 1, 15, 30)
>>> format_datetime(dt, locale='en_US')
u'Apr 1, 2007, 3:30:00\u202fPM'
```

For any pattern requiring the display of the timezone:

```
>>> format_datetime(dt, 'full', tzinfo=get_timezone('Europe/Paris'),
...                 locale='fr_FR')
'dimanche 1 avril 2007, 17:30:00 heure d'été d'Europe centrale'
>>> format_datetime(dt, "yyyy.MM.dd G 'at' HH:mm:ss zzz",
...                 tzinfo=get_timezone('US/Eastern'), locale='en')
u'2007.04.01 AD at 11:30:00 EDT'
```

param datetime the *datetime* object; if *None*, the current date and time is used

param format one of “full”, “long”, “medium”, or “short”, or a custom date/time pattern

param tzinfo the timezone to apply to the time for display

param locale a *Locale* object or a locale identifier

4.1.3 format_decimal

```
babelwrap.format_decimal = functools.partial(<function format_decimal>,
locale=Locale('en'))
```

Default locale from setlang() Otherwise: Return the given decimal number formatted for a specific locale.

```
>>> format_decimal(1.2345, locale='en_US')
u'1.234'
>>> format_decimal(1.2346, locale='en_US')
u'1.235'
>>> format_decimal(-1.2346, locale='en_US')
u'-1.235'
>>> format_decimal(1.2345, locale='sv_SE')
u'1,234'
>>> format_decimal(1.2345, locale='de')
u'1,234'
```

The appropriate thousands grouping and the decimal separator are used for each locale:

```
>>> format_decimal(12345.5, locale='en_US')
u'12,345.5'
```

By default the locale is allowed to truncate and round a high-precision number by forcing its format pattern onto the decimal part. You can bypass this behavior with the *decimal_quantization* parameter:

```

>>> format_decimal(1.2346, locale='en_US')
u'1.235'
>>> format_decimal(1.2346, locale='en_US', decimal_quantization=False)
u'1.2346'
>>> format_decimal(12345.67, locale='fr_CA', group_separator=False)
u'12345,67'
>>> format_decimal(12345.67, locale='en_US', group_separator=True)
u'12,345.67'

```

param number the number to format

param format

param locale the *Locale* object or locale identifier

param decimal_quantization Truncate and round high-precision numbers to the format pattern. Defaults to *True*.

param group_separator Boolean to switch group separator on/off in a locale's number format.

4.1.4 format_list

`babelwrap.format_list = functools.partial(<function format_list>, locale=Locale('en'))`

Default locale from setlang() Otherwise:

Format the items in *lst* as a list.

```

>>> format_list(['apples', 'oranges', 'pears'], locale='en')
u'apples, oranges, and pears'
>>> format_list(['apples', 'oranges', 'pears'], locale='zh')
u'applesorangespears'
>>> format_list(['omena', 'peruna', 'aplari'], style='or', locale='fi')
u'omena, peruna tai aplari'

```

These styles are defined, but not all are necessarily available in all locales. The following text is verbatim from the Unicode TR35-49 spec [1].

- **standard**: A typical ‘and’ list for arbitrary placeholders. eg. “January, February, and March”
- **standard-short**: A short version of a ‘and’ list, suitable for use with short or abbreviated placeholder values. eg. “Jan., Feb., and Mar.”
- **or**: A typical ‘or’ list for arbitrary placeholders. eg. “January, February, or March”
- **or-short**: A short version of an ‘or’ list. eg. “Jan., Feb., or Mar.”
- **unit**: A list suitable for wide units. eg. “3 feet, 7 inches”
- **unit-short**: A list suitable for short units eg. “3 ft, 7 in”
- **unit-narrow**: A list suitable for narrow units, where space on the screen is very limited. eg. “3 7”

[1]: <https://www.unicode.org/reports/tr35/tr35-49/tr35-general.html#ListPatterns>

Parameters

- **lst** – a sequence of items to format in to a list
- **style** – the style to format the list with. See above for description.

- **locale** – the locale

4.1.5 format_percent

```
babelwrap.format_percent = functools.partial(<function format_percent>,
locale=Locale('en'))
```

Default locale from setlang() Otherwise: Return formatted percent value for a specific locale.

```
>>> format_percent(0.34, locale='en_US')
u'34%'
>>> format_percent(25.1234, locale='en_US')
u'2,512%'
>>> format_percent(25.1234, locale='sv_SE')
u'2\xa0512\xa0%'
```

The format pattern can also be specified explicitly:

```
>>> format_percent(25.1234, u'#,##0%', locale='en_US')
u'25,123%'
```

By default the locale is allowed to truncate and round a high-precision number by forcing its format pattern onto the decimal part. You can bypass this behavior with the *decimal_quantization* parameter:

```
>>> format_percent(23.9876, locale='en_US')
u'2,399%'
>>> format_percent(23.9876, locale='en_US', decimal_quantization=False)
u'2,398.76%'
```

```
>>> format_percent(229291.1234, locale='pt_BR', group_separator=False)
u'22929112%'
```

```
>>> format_percent(229291.1234, locale='pt_BR', group_separator=True)
u'22.929.112%'
```

param number the percent number to format

param format

param locale the *Locale* object or locale identifier

param decimal_quantization Truncate and round high-precision numbers to the format pattern. Defaults to *True*.

param group_separator Boolean to switch group separator on/off in a locale's number format.

4.1.6 format_unit

`babelwrap.format_unit = functools.partial(<function format_unit>, locale=Locale('en'))`

Default locale from `setlang()` Otherwise: Format a value of a given unit.

Values are formatted according to the locale’s usual pluralization rules and number formats.

```
>>> format_unit(12, 'length-meter', locale='ro_RO')
u'12 metri'
>>> format_unit(15.5, 'length-mile', locale='fi_FI')
u'15,5 mailia'
>>> format_unit(1200, 'pressure-millimeter-ofhg', locale='nb')
u'1\xa0200 millimeter kvikks\xfb8lv'
>>> format_unit(270, 'ton', locale='en')
u'270 tons'
```

Number formats may be overridden with the `format` parameter.

```
>>> import decimal
>>> format_unit(decimal.Decimal("-42.774"), 'temperature-celsius', 'short',
↳format='#.0', locale='fr')
u'-42,8\u202f\xb0C'
```

The locale’s usual pluralization rules are respected.

```
>>> format_unit(1, 'length-meter', locale='ro_RO')
u'1 metru'
>>> format_unit(0, 'length-mile', locale='cy')
u'0 mi'
>>> format_unit(1, 'length-mile', locale='cy')
u'1 filltir'
>>> format_unit(3, 'length-mile', locale='cy')
u'3 milltir'
```

```
>>> format_unit(15, 'length-horse', locale='fi')
Traceback (most recent call last):
...
UnknownUnitError: length-horse is not a known unit in fi
```

New in version 2.2.0.

param value the value to format. If this is a string, no number formatting will be attempted.

param measurement_unit the code of a measurement unit. Known units can be found in the CLDR Unit Validity XML file: <https://unicode.org/repos/cldr/tags/latest/common/validity/unit.xml>

param length “short”, “long” or “narrow”

param format An optional format, as accepted by `format_decimal`.

param locale the `Locale` object or locale identifier

4.2 category module

Classes and functions for defining categories.

4.2.1 Category

class `category.Category`(*cls, bases, classdict*)

Bases: `enum.EnumMeta`

MetaClass for *Categorized* (not for public use).

References

`enum.EnumMeta`

4.2.2 Categorized

class `category.Categorized`(*value*)

Bases: `enum.Enum`

Derive from this class to define a new category. e.g.:

```
class _BoardOption(NamedTuple):
    STR: str
    AX: Callable[[matplotlib.figure.Figure, tuple],
                 matplotlib.axes.Axes]
    VERSIONS: Iterable = ALL

class BoardOption(Categorized):
    HASH = _BoardOption(STR = _("a hash"), AX = hash_board)
    SQUARES = _BoardOption(
        STR = _("squares"),
        AX = squares_board,
        VERSIONS = from_version("1.5.0"),
    )
```

Raises `AttributeError` – Upon attempt to add, delete or change member.

The above example assumes the existence of functions named `hash_board` and `squares_board`. It creates a *Category* named `BoardOption` with two members, `BoardOption.HASH` and `BoardOption.SQUARES`, each of which has three attributes: `STR`, `AX` and `VERSIONS`.

```
>>> isinstance(BoardOption, Category)
True
>>> isinstance(BoardOption.HASH, Categorized)
True
```

A dropdown is a classic example of a category because different values should be available in different versions and all values typically should display differently in different languages. A member with an attribute named “VERSIONS”, will appear only for those versions. If a member has an attribute named “STR”, then that’s how that member will print (use functions from *babelwrap module*). For example, the following would yield a

dropdown that contains only the local language translation of “a hash” in version 1.0.0, but translations of both “a hash” and “squares” in version 1.5.0 and above:

```
ipywidgets.Dropdown(options=BoardOption)
```

This will work even if the dropdown is declared *before* calling `setvers()` and `setlang()`. A member evaluates to False if not in the set version:

```
>>> setvers("1.0.0")
(1,0,0)
>>> bool(BoardOption.HASH)
True
>>> bool(BoardOption.SQUARES)
False
```

If a member has an attribute named “CALL”, then the value of that attribute will be invoked when that member is called. If the CALL is a tuple-class (e.g. `NamedTuple`), then that member is a “factory member”, and calling it will return a new *Categorized* with the attributes of that tuple-class (initialized with the called parameters). For example:

```
class _Jump(NamedTuple):
    FROM: Tuple[int, ...]
    TO: Tuple[int, ...]
    VERSIONS: Iterable = ALL
    def __str__(self) -> str:
        return _("{origin} to {destination}").format(
            origin=self.FROM, destination=self.TO
        )

class _Move(NamedTuple):
    STR: str
    CALL: Optional[Callable] = None
    VERSIONS: Iterable = ALL

class Move(Categorized):
    PASS = _Move(STR=_("Pass"))
    JUMP = _Move(STR=_("Reposition"), CALL=_Jump)

jumps = (Move.JUMP(FROM=(1,2), TO=dest) for dest in ((3,1), (3,3), (2,4)))
CurrentLegal = ctg(*jumps, name="CurentLegal", uniquify=True) | Move.PASS
```

In this example, the `Move` *Category* has two members, `Move.PASS` and `Move.JUMP`, both of which have `STR`, `CALL`, and `VERSIONS` attributes.

```
>>> str(Move)
'Pass and Reposition'
```

`Move.JUMP` is a factory member used in the second-to-last line to create three new instances of *Categorized*. They do not become members of any category until the last line which creates the `CurrentLegal` category from them unioned with `Move.PASS`. Then the members of `CurrentLegal` are `CurrentLegal.JUMP`, `CurrentLegal.JUMP1`, `CurrentLegal.JUMP2`, and `CurrentLegal.PASS` (the names “JUMP1” and “JUMP2” are constructed by `ctg()` to avoid name-collision).

```
>>> str(CurrentLegal)
'(1,2) to (3,1), (1,2) to (3,3), (1,2) to (2,4) and Pass'
```

Each of the “JUMP” members of `CurrentLegal` has `FROM`, `TO`, and `VERSIONS` attributes, but `CurrentLegal.PASS` has the same attributes as `Move.PASS`. It is the same entity, so it evaluates as `==` and is `in` both categories:

```
>>> CurrentLegal.PASS == Move.PASS
True
>>> CurrentLegal.PASS in Move
True
>>> Move.PASS in CurrentLegal
True
>>> CurrentLegal.JUMP in Move
False
```

The only difference between the entities is context:

```
>>> str(type(Move.PASS))
'Pass and Reposition'
>>> str(type(CurrentLegal.PASS))
'(1,2) to (3,1), (1,2) to (3,3), (1,2) to (2,4) and Pass'
```

Categories support set operations, so you can get a new category containing all members that are in both categories (i.e. the set intersection):

```
>>> str(CurrentLegal & Move)
'Pass'
```

...set difference:

```
>>> str(CurrentLegal - Move)
'(1,2) to (3,1), (1,2) to (3,3) and (1,2) to (2,4)'
```

...set union:

```
>>> str(CurrentLegal | Move)
'(1,2) to (3,1), (1,2) to (3,3), (1,2) to (2,4), Pass and Reposition'
```

...and set symmetric difference:

```
>>> str(CurrentLegal ^ Move)
'(1,2) to (3,1), (1,2) to (3,3), (1,2) to (2,4) and Reposition'
```

You can also test for containment of entire categories:

```
>>> CurrentLegal >= (Move - Move.JUMP)
True
```

...and for proper superset (or subset):

```
>>> CurrentLegal > (Move - Move.JUMP)
True
```

Tip: The `_()` function should be used in categories to designate messages that need to be translated, and that function will be applied when categories are being initialized. To permit changing language after initialization, keep the initialized messages in the original language (the one for which you have translations) by setting this before setting the categories:

```
def _(message: str) -> str:
    return message
```

... then set `_()` to the actual translation function after setting the categories.

References

`enum.Enum`

4.2.3 `ctg()`

`category.ctg(*members: Iterable, name: str = 'Categorized', unify: bool = False) → type`

Generate a new *Category* from one or more *Categorized*. E.g.:

```
ctg(Color.BLACK, Marker.CIRCLE)
```

Parameters

- ***members** – The members for the new *Category*.
- **name** (str) – The name of the new *Category*. Defaults to “Categorized”
- **uniquify** (bool) – If `True`, name collisions will be resolved by altering member names. Useful with factory members. Defaults to `False`.

Returns The new *Category*.

Raises `TypeError` – If attempt to combine non-equal members having the same name without setting `uniquify` to `True`.

4.2.4 `setvers()`

`category.setvers(name: Optional[str] = None) → Tuple[Union[int, str], ...]`

Get or set the version. E.g.:

```
setvers() # to get the currently set version
setlang("1.1.0") # to set a version (e.g. for testing)
setlang("") # to restore the default from pyproject.toml
```

Parameters `name` (str) – The name of the version to set. Default to `None`.

Returns The currently set version as a tuple.

4.2.5 parse_version()

`category.parse_version(name: Optional[str] = None, min_parts: int = 3) → Tuple[Union[int, str], ...]`

Yields sortable tuples for a version name. E.g.:

```
>>> parse_version("1.0.1-alpha")
(1, 0, 1, 'alpha')
```

Parameters

- **name** (str) – dot/hyphen-delimited version name
- **min_parts** (int) – The minimum parts in the tuple. Default is 3.

Returns A tuple with one member per part of the name padded with as many zeros as necessary to achieve `min_parts`. The numeric parts are integers, so the tuples sort correctly.

To support [semantic versioning](#), omits any leading “v”, and appends an extra “~” part to versions with no “-”. E.g.:

```
>>> parse_version("v1.0.0-alpha") < parse_version("1.0")
True
```

4.2.6 from_version()

`category.from_version(start: str, to: Optional[str] = None) → Iterable`

The simple interval starting with a certain version. E.g.:

```
>>> from_version("1.5.0")
[(1, 5, 0, '~'),+inf)
```

Parameters

- **start** (str) – The starting version
- **to** (str) – If set, the (excluded) last version. If None, there is no end version. Default to None.

Returns The `portion.interval.Interval`

4.2.7 ALL

`category.ALL = (-inf,+inf)`

A shortcut for the `portion.interval.Interval` that contains all (e.g. versions)

4.3 const module

Contains redscience constants plus functions for internationalization and versioning.

Most constants in this module are encoded as a *category.Category* or as a *NamedTuple*. Each displays in the locale set via *setlang()* and excludes members not in the version set via *setvers()*.

Examples:

```
import matplotlib.pyplot as plt

ttt=Game()
ipywidgets.Dropdown(options=sorted(Command), value=Command.QUIT)
ipywidgets.Button(
    description=Command.QUIT,
    tooltip=Command.QUIT.TOOLTIP,
    icon=Command.QUIT.ICON,
)
fig = plt.figure(1,(
    Layout.FIGURE_HEIGHT,
    Layout.FIGURE_WIDTH,
))
ax = ttt.AXES(fig)
ax.scatter(
    x = 1,
    y = 1,
    c = Color.WHITE.HEX,
    marker = Marker.CIRCLE.CODE,
    edgecolors = Color.BLACK.HEX,
)

setlang("")
setvers("")
print(ttt.RULES)
print(Command.QUIT)
plt.show()
```

4.3.1 General Constants

Color

class `const.Color`(*value*)

Bases: *category.Categorized*

Color used in a game. E.g.:

```
Color.BLACK
```

Attributes:

STR (str) A localized name. How the color prints.

HEX (str) A hex code to communicate the color to computers.

VERSIONS (Iterable) The versions which offer this color.

```
BLACK = _Color(STR='black', HEX='#000000', VERSIONS=(-inf,+inf))
BLUE = _Color(STR='blue', HEX='#95d0fc', VERSIONS=(-inf,+inf))
GRAY = _Color(STR='gray', HEX='#929591', VERSIONS=(-inf,+inf))
GREEN = _Color(STR='green', HEX='#96f97b', VERSIONS=(-inf,+inf))
ORANGE = _Color(STR='orange', HEX='#fdaa48', VERSIONS=(-inf,+inf))
PINK = _Color(STR='pink', HEX='#ff81c0', VERSIONS=(-inf,+inf))
PURPLE = _Color(STR='purple', HEX='#bf77f6', VERSIONS=(-inf,+inf))
WHITE = _Color(STR='white', HEX='#ffffff', VERSIONS=(-inf,+inf))
YELLOW = _Color(STR='yellow', HEX='#ffff14', VERSIONS=(-inf,+inf))
```

Command

```
class const.Command(value)
```

Bases: [category.Categorized](#)

Command from user to the application. E.g.:

```
Command.NEW
```

Attributes:

STR (str) A localized name. How the command prints.

KEY (str) A localized shortcut key.

VERSIONS (Iterable) The versions which offer this command.

Each command tests == to its KEY as well as to itself. For example, if the language is English:

```
>>> Command.NEW == "n"
True
```

```
NEW = _Command(STR='Play New', KEY='n', VERSIONS=(-inf,+inf))
```

```
QUIT = _Command(STR='Quit', KEY='q', VERSIONS=(-inf,+inf))
```

```
UNDO = _Command(STR='Back', KEY='z', VERSIONS=(-inf,+inf))
```

Layout

```
class const.Layout(value)
```

Bases: [enum.IntEnum](#)

Layout constants. E.g.:

```
Layout.POINTS_PER_INCH
```

See [enum.IntEnum](#)

```

FIGURE_HEIGHT = 5
FIGURE_WIDTH = 5
MARKER_MARGIN = 8
POINTS_PER_INCH = 54

```

4.3.2 Player Constants

Player

```
class const.Player(TYPE: const._PlayerType = PlayerType.HUMAN)
```

Bases: tuple

The constant parts of a player. E.g.:

```
Player() # To use all defaults (i.e. human)
```

Attributes:

TYPE (*PlayerType*) If specified, determines the type. Default is Human.

property VERSIONS: Iterable

The versions in which this player can be selected.

PlayerType

```
class const.PlayerType(value)
```

Bases: *category.Categorized*

Type of player. E.g.:

```
PlayerType.HUMAN
```

Attributes:

STR (**str**) A localized name. How the type prints.

VERSIONS (**Iterable**) The versions which offer this PlayerType.

```
ANONYMOUS = _PlayerType(STR='Anonymous', VERSIONS=(-inf,+inf))
```

```
HUMAN = _PlayerType(STR='Human', VERSIONS=[(1, 2, 0, '~'),+inf))
```

PlayerColor

```
class const.PlayerColor(value)
```

Bases: *category.Categorized*

Color used in a game. E.g.:

```
Color.BLACK
```

Attributes:

STR (str) A localized name. How the color prints.

HEX (str) A hex code to communicate the color to computers.

VERSIONS (Iterable) The versions which offer this color.

```
BLACK = _Color(STR='black', HEX='#000000', VERSIONS=(-inf,+inf))
```

```
PINK = _Color(STR='pink', HEX='#ff81c0', VERSIONS=(-inf,+inf))
```

```
WHITE = _Color(STR='white', HEX='#ffffff', VERSIONS=(-inf,+inf))
```

```
YELLOW = _Color(STR='yellow', HEX='#ffff14', VERSIONS=(-inf,+inf))
```

DefaultName

```
class const.DefaultName(value)
```

Bases: *category.Categorized*

Default names for players. E.g.:

```
DefaultName.PLAYER_ONE
```

Attributes:

STR (str) A localized name. How the name prints.

VERSIONS (Iterable) The versions which offer this name.

```
PLAYER_FOUR = _DefaultName(STR='Player 4', VERSIONS=[(1, 5, 0, '~'),+inf))
```

```
PLAYER_ONE = _DefaultName(STR='Player 1', VERSIONS=(-inf,+inf))
```

```
PLAYER_THREE = _DefaultName(STR='Player 3', VERSIONS=[(1, 5, 0, '~'),+inf))
```

```
PLAYER_TWO = _DefaultName(STR='Player 2', VERSIONS=(-inf,+inf))
```

4.3.3 Piece Constants

Marker

```
class const.Marker(value)
```

Bases: *category.Categorized*

The shape of a game piece. E.g.:

```
Marker.CIRCLE
```

Attributes:

STR (str) A localized name. How the marker prints.

CODE (str) A `matplotlib.marker`.

VERSIONS (Iterable) The versions which offer this marker.

```
CIRCLE = _Marker(STR='circle', CODE='o', VERSIONS=(-inf,+inf))
```

PieceRules

class `const.PieceRules`(*INITIAL_RESERVES: Tuple[int, ...]*)

Bases: `tuple`

Rules for a type of piece in a game. E.g.:

```
PieceRules(INITIAL_RESERVES=(5,4))
```

Attributes:

INITIAL_RESERVES (**Tuple**[int, ...]) Specifies the number of each color in initial reserves, e.g. (5, 4) means start with 5 of the first color and 4 of the second color in reserve.

property `RESERVES_STR`: **str**

A constant localized str describing initial reserves for the piece. E.g.:

```
>>> piece.RESERVES_STR
'5 black and 4 white start in reserve'
```

property `STRS`: **Tuple**[str, ...]

Get tuple of strings describing the rules for the piece. E.g.:

```
>>> piece.STRS
('No movement', 'No power', '5 black and 4 white start in reserve')
```

property `VERSIONS`: **Iterable**

The versions which offer this piece. E.g.:

```
>>> piece.VERSIONS
(-inf, +inf)
```

Directions

class `const.Directions`(*value*)

Bases: `category.Categorized`

Categories of ways in which to move or build in square-tiled space. E.g:

```
>>> Directions.DIAGONAL(2)
```

```
((1,1), (1,-1), (-1,1), (-1,-1))
```

Parameters `dimensions` (`int`) – The number of dimensions in the space

Returns Of relative coordinates (each a `numpy.array` of `int`)

Return type `tuple`

Attributes:

STR (**str**) A localized name. How the Directions prints.

CALL (**Callable**) The bound method that yields the tuples.

VERSIONS (**Iterable**) The versions which offer this option.

Tip: To get cache info:

```
Directions.DIAGONAL.call.cache_info()
```

```
ANY = _DirectionsValue(STR='any direction', CALL=<functools._lru_cache_wrapper
object>, VERSIONS=(-inf,+inf))
```

```
DIAGONAL = _DirectionsValue(STR='diagonal', CALL=<functools._lru_cache_wrapper
object>, VERSIONS=(-inf,+inf))
```

```
KNIGHT = _DirectionsValue(STR='knight move', CALL=<functools._lru_cache_wrapper
object>, VERSIONS=(-inf,+inf))
```

```
ORTHOGONAL = _DirectionsValue(STR='orthogonal', CALL=<functools._lru_cache_wrapper
object>, VERSIONS=(-inf,+inf))
```

4.3.4 Game Constants

Game

```
class const.Game(PLAYERS: const._PlayersOption = PlayersOption.TWO, COLOR: const._ColorOption =
    ColorOption.ASSIGNED, BOARD: const._BoardOption = BoardOption.HASH,
    DIMENSIONS: typing.Tuple[int, ...] = (3, 3), PIECES: typing.Tuple[const.PieceRules, ...] =
    (PieceRules(INITIAL_RESERVES=(5, 4)),), MOVE_CHECKS: typing.Union[typing.Tuple[()],
    typing.Tuple[const._CheckOption, ...]] =
    (<CheckOption.THREE_SAME_COLOR_IN_ROW_WINS: _CheckOption(STR='first
    3-same-color-in-a-row wins', VERSIONS=(-inf,+inf), PATTERN='CCC',
    DIRECTIONS=<Directions.ANY: _DirectionsValue(STR='any direction',
    CALL=<functools._lru_cache_wrapper object>, VERSIONS=(-inf,+inf))>,
    OUTCOME=<Outcome.VICTORY: _OutcomeValue(STR='Victory', FORMAT='Victory:
    {players}', CALL=<function Outcome._formatter>, VERSIONS=(-inf,+inf))>>,),
    STALEMATE: const._StalemateOption = StalemateOption.DRAW)
```

Bases: tuple

A game definition. E.g.:

```
Game() # To use all defaults (i.e. Tic-Tac-Toe)
```

Attributes:

PLAYERS (*PlayersOption*) If specified, determines the number/ relationship of players. Default is 2-Player.

COLOR (*ColorOption*) If specified, determines the significance of colors. Default is Assigned Colors.

BOARD (*BoardOption*) If specified, determines the type of board. Default is hash.

DIMENSIONS (*Tuple[int, ...]*) If specified, determines the dimensions of the board. Default is (3,3).

PIECES (*Tuple[PieceRules, ...]*) If specified, determines piece-specific rules. Default is to have only one type of piece (circle) with 5 black and 4 white circles starting in reserve.

MOVE_CHECKS (**Tuple**[*CheckOption*, ...]) If specified, determines which rules are checked at the end of each move. Can be None. Default is to award the win to any player that gets three of the same color in a row.

STALEMATE (*StalemateOption*) If specified, determines the result of stalemate. Default is that stalemate results in a draw.

AXES (*fig: matplotlib.figure.Figure*) → `matplotlib.axes._axes.Axes`

The board in terms of matplotlib E.g.:

```
import matplotlib.pyplot as plt
figure = plt.figure(1,(
    Layout.FIGURE_HEIGHT,
    Layout.FIGURE_WIDTH,
))
game.AXES(fig=figure)
plt.show()
```

Parameters **fig** (`matplotlib.figure.Figure_`) – The Figure in which the Axes will appear

Returns A framework in which to place pieces

Return type `matplotlib.axes.Axes`

property **MARKER_SIZE**: **float**

The (int) size for markers in this game. E.g.:

```
>>> Game().MARKER_SIZE
6724
```

property **RULES**: **str**

A localized str of check and stalemate rules. E.g:

```
>>>game().RULES 'Rules: First 3-same-color-in-a-row wins and stalemate draws'
```

property **VERSIONS**: **Iterable**

The (Tuple) versions which offer this Game. E.g.:

```
>>> Game().VERSIONS
(-inf,+inf)
```

PlayersOption

class `const.PlayersOption`(*value*)

Bases: `category.Categorized`

Category of game by number/relationship of players. E.g.:

```
PlayersOption.TWO
```

Attributes:

STR (**str**) A localized name. How the option prints.

NUM (**int**) The number of regular players.

VERSIONS (**Iterable**) The versions which offer this option.

```
THREE = _PlayersOption(STR='3-Player', NUM=3, VERSIONS=(-inf,+inf))
```

```
TWO = _PlayersOption(STR='2-Player', NUM=2, VERSIONS=(-inf,+inf))
```

ColorOption

```
class const.ColorOption(value)
```

Bases: *category.Categorized*

Category of game by how it treats colors. E.g.:

```
ColorOption.ASSIGNED
```

Attributes:

STR (str) A localized name. How the option prints.

VERSIONS (Iterable) The versions which offer this marker.

```
ASSIGNED = _ColorOption(STR='Assigned Colors', VERSIONS=(-inf,+inf))
```

BoardOption

```
class const.BoardOption(value)
```

Bases: *category.Categorized*

Category of game board. E.g.:

```
BoardOption.HASH
```

BoardValue Attributes:

STR (str) A localized name. How the option prints.

AX (Callable) Function to return `matplotlib.axes.Axes`, given a `matplotlib.figure.Figure` and tuple of dimensions.

VERSIONS (Iterable) The versions which offer this option.

```
HASH = _BoardOption(STR='a hash', AX=<function BoardOption._hash_board>, VERSIONS=(-inf,+inf))
```

CheckOption

```
class const.CheckOption(value)
```

Bases: *category.Categorized*

Game rules checked at the end of each move. E.g.:

```
CheckOption.THREE_SAME_COLOR_IN_ROW_WINS
```

Attributes:

STR (str) A localized name. How the option prints.

VERSIONS (Iterable) The versions which offer this option.

PATTERN (str) If specified, a type of pattern to be checked. Default to None.

DIRECTIONS (*Directions*) If specified, directions in which to check the pattern. Default to None.

OUTCOME (*Outcome*) If specified, the outcome if the check triggers. Default to None.

```
THREE_SAME_COLOR_IN_ROW_WINS = _CheckOption(STR='first 3-same-color-in-a-row wins',
VERSIONS=(-inf,+inf), PATTERN='CCC', DIRECTIONS=<Directions.ANY:
_DirectionsValue(STR='any direction', CALL=<functools.lru_cache_wrapper object>,
VERSIONS=(-inf,+inf))>, OUTCOME=<Outcome.VICTORY: _OutcomeValue(STR='Victory',
FORMAT='Victory: {players}', CALL=<function Outcome._formatter>,
VERSIONS=(-inf,+inf))>)
```

StalemateOption

class `const.StalemateOption(value)`

Bases: `category.Categorized`

How stalemate ends a game. E.g.:

```
StalemateOption.DRAW
```

Attributes:

STR (**str**) A localized name. How the option prints.

VERSIONS (**Iterable**) The versions which offer this option.

```
DRAW = _StalemateOption(STR='stalemate draws', VERSIONS=(-inf,+inf))
```

4.3.5 Move Constants

Move

class `const.Move(value)`

Bases: `category.Categorized`

A type of move in a game. Prints localized str. Examples:

```
Move.PASS
Move.PLACE(COLOR=Color.WHITE, MARKER=Marker.CIRCLE, TO=(2,3))
Move.JUMP(FROM=(1,1), TO=(2,3))
```

Attributes:

STR (**str**) A localized name. How the move prints.

VERSIONS (**Iterable**) The versions which offer this Move.

TO (**Tuple[int,...]**) *Only for PLACE and JUMP.* destination coordinates.

COLOR (*PlayerColor*) *Only for PLACE.* Color to be placed. Default is black

MARKER (*Marker*) *Only for PLACE.* Shape to be placed. Default is circle.

FROM (**Tuple[int,...]**) *Only for JUMP.* Origin coordinates.

```
AGREE = _Move(STR='Agree to draw', CALL=None, VERSIONS=[(1, 5, 0, '~'),+inf))
```

```
JUMP = _Move(STR='Reposition', CALL=<class 'const._Jump'>, VERSIONS=[(1, 5, 0, '~'),+inf])

OFFER = _Move(STR='Offer to draw', CALL=None, VERSIONS=[(1, 5, 0, '~'),+inf])

PASS = _Move(STR='Pass', CALL=None, VERSIONS=(-inf,+inf))

PLACE = _Move(STR='Place from reserves', CALL=<class 'const._Placement'>,
VERSIONS=(-inf,+inf))

REFUSE = _Move(STR='Refuse to draw', CALL=None, VERSIONS=[(1, 5, 0, '~'),+inf])
```

Outcome

`class const.Outcome(value)`

Bases: *category.Categorized*

Function to apply localized formatting to strings. E.g:

```
>>> winners = (DefaultName.PLAYER_ONE, DefaultName.PLAYER_THREE)
>>> Outcome.VICTORY(players=winners)
'Victory: Player 1 and Player 3'
```

Parameters ****kwargs** – a str for each bookmark in the STR

Returns The localized string.

Return type str

OutcomeValue Attributes:

STR (str) A localized name. How the Directions prints.

CALL (Callable) The bound method that yields the str.

FORMAT (str) The formatted string for the CALL.

VERSIONS (Iterable) The versions which offer this option.

```
VICTORY = _OutcomeValue(STR='Victory', FORMAT='Victory: {players}', CALL=<function
Outcome._formatter>, VERSIONS=(-inf,+inf))
```

4.3.6 Language Functions

setlang()

`const.setlang(*langs: str) → babel.core.Locale`

Gets/sets locale for language functions. E.g.:

```
setlang() # to get the currenty set locale
setlang("zh_Hans_HK", "zh_HK") # to set a language (e.g. for testing)
setlang("") # to restore the default language
```

Parameters ***langs (str)** – locale names in order of preference.

Returns The `babel.core.Locale` that is currently set.

The babel functions can then be used (defaulted to the set language) as follows:

```
print(format_decimal(-12345.6789))
print(format_percent(-12345.6789))
print(format_unit(-12345.6789, "second"))
print(format_datetime(datetime.datetime.now()))
print(format_list(["Alvin", "Simon", "Theodore"]))
print(_("Hello world!"))
```

format_datetime()

const. **format_datetime**(*datetime*: *_Instant = None*, *format*: *_PredefinedTimeFormat | str = 'medium'*, *tzinfo*: *datetime.tzinfo | None = None*, *locale*: *Locale | str | None = Locale('en')*) → str

Default locale from setlang() Otherwise: Return a date formatted according to the given pattern.

```
>>> from datetime import datetime
>>> dt = datetime(2007, 4, 1, 15, 30)
>>> format_datetime(dt, locale='en_US')
u'Apr 1, 2007, 3:30:00\u202fPM'
```

For any pattern requiring the display of the timezone:

```
>>> format_datetime(dt, 'full', tzinfo=get_timezone('Europe/Paris'),
...                 locale='fr_FR')
'dimanche 1 avril 2007, 17:30:00 heure d'été d'Europe centrale'
>>> format_datetime(dt, "yyyy.MM.dd G 'at' HH:mm:ss zzz",
...                 tzinfo=get_timezone('US/Eastern'), locale='en')
u'2007.04.01 AD at 11:30:00 EDT'
```

param datetime the *datetime* object; if *None*, the current date and time is used

param format one of “full”, “long”, “medium”, or “short”, or a custom date/time pattern

param tzinfo the timezone to apply to the time for display

param locale a *Locale* object or a locale identifier

format_decimal()

const. **format_decimal**(*number*: *float | decimal.Decimal | str*, *format*: *str | NumberPattern | None = None*, *locale*: *Locale | str | None = Locale('en')*, *decimal_quantization*: *bool = True*, *group_separator*: *bool = True*) → str

Default locale from setlang() Otherwise: Return the given decimal number formatted for a specific locale.

```
>>> format_decimal(1.2345, locale='en_US')
u'1.234'
>>> format_decimal(1.2346, locale='en_US')
u'1.235'
>>> format_decimal(-1.2346, locale='en_US')
u'-1.235'
>>> format_decimal(1.2345, locale='sv_SE')
u'1,234'
```

(continues on next page)

(continued from previous page)

```
>>> format_decimal(1.2345, locale='de')
u'1,234'
```

The appropriate thousands grouping and the decimal separator are used for each locale:

```
>>> format_decimal(12345.5, locale='en_US')
u'12,345.5'
```

By default the locale is allowed to truncate and round a high-precision number by forcing its format pattern onto the decimal part. You can bypass this behavior with the `decimal_quantization` parameter:

```
>>> format_decimal(1.2346, locale='en_US')
u'1.235'
>>> format_decimal(1.2346, locale='en_US', decimal_quantization=False)
u'1.2346'
>>> format_decimal(12345.67, locale='fr_CA', group_separator=False)
u'12345,67'
>>> format_decimal(12345.67, locale='en_US', group_separator=True)
u'12,345.67'
```

param number the number to format

param format

param locale the *Locale* object or locale identifier

param decimal_quantization Truncate and round high-precision numbers to the format pattern. Defaults to *True*.

param group_separator Boolean to switch group separator on/off in a locale's number format.

format_list()

`const. format_list(lst: Sequence[str], style: Literal['standard', 'standard-short', 'or', 'or-short', 'unit', 'unit-short', 'unit-narrow'] = 'standard', *, locale: Locale | str | None = Locale('en'))` → str

Default locale from setlang() Otherwise:

Format the items in *lst* as a list.

```
>>> format_list(['apples', 'oranges', 'pears'], locale='en')
u'apples, oranges, and pears'
>>> format_list(['apples', 'oranges', 'pears'], locale='zh')
u'applesorangespears'
>>> format_list(['omena', 'peruna', 'aplari'], style='or', locale='fi')
u'omena, peruna tai aplari'
```

These styles are defined, but not all are necessarily available in all locales. The following text is verbatim from the Unicode TR35-49 spec [1].

- `standard`: A typical ‘and’ list for arbitrary placeholders. eg. “January, February, and March”
- `standard-short`: A short version of a ‘and’ list, suitable for use with short or abbreviated placeholder values. eg. “Jan., Feb., and Mar.”
- `or`: A typical ‘or’ list for arbitrary placeholders. eg. “January, February, or March”

- **or-short**: A short version of an ‘or’ list. eg. “Jan., Feb., or Mar.”
- **unit**: A list suitable for wide units. eg. “3 feet, 7 inches”
- **unit-short**: A list suitable for short units eg. “3 ft, 7 in”
- **unit-narrow**: A list suitable for narrow units, where space on the screen is very limited. eg. “3 7”

[1]: <https://www.unicode.org/reports/tr35/tr35-49/tr35-general.html#ListPatterns>

Parameters

- **lst** – a sequence of items to format in to a list
- **style** – the style to format the list with. See above for description.
- **locale** – the locale

format_percent()

`const.format_percent(number: float | decimal.Decimal | str, format: str | NumberPattern | None = None, *, locale: Locale | str | None = Locale('en'), decimal_quantization: bool = True, group_separator: bool = True) → str`

Default locale from `setlang()` Otherwise: Return formatted percent value for a specific locale.

```
>>> format_percent(0.34, locale='en_US')
u'34%'
>>> format_percent(25.1234, locale='en_US')
u'2,512%'
>>> format_percent(25.1234, locale='sv_SE')
u'2\xa0512\xa0%'
```

The format pattern can also be specified explicitly:

```
>>> format_percent(25.1234, u'#,##0%', locale='en_US')
u'25,123%'
```

By default the locale is allowed to truncate and round a high-precision number by forcing its format pattern onto the decimal part. You can bypass this behavior with the `decimal_quantization` parameter:

```
>>> format_percent(23.9876, locale='en_US')
u'2,399%'
>>> format_percent(23.9876, locale='en_US', decimal_quantization=False)
u'2,398.76%'
```

```
>>> format_percent(229291.1234, locale='pt_BR', group_separator=False)
u'22929112%'
```

```
>>> format_percent(229291.1234, locale='pt_BR', group_separator=True)
u'22.929.112%'
```

param number the percent number to format

param format

param locale the *Locale* object or locale identifier

param decimal_quantization Truncate and round high-precision numbers to the format pattern. Defaults to *True*.

param group_separator Boolean to switch group separator on/off in a locale’s number format.

format_format_unit()

```
const.format_unit(value: float | decimal.Decimal, measurement_unit: str, length: Literal['short', 'long',
    'narrow'] = 'long', format: str | None = None, *, locale: Locale | str | None = Locale('en'))
    → str
```

Default locale from setlang() Otherwise: Format a value of a given unit.

Values are formatted according to the locale’s usual pluralization rules and number formats.

```
>>> format_unit(12, 'length-meter', locale='ro_RO')
u'12 metri'
>>> format_unit(15.5, 'length-mile', locale='fi_FI')
u'15,5 mailia'
>>> format_unit(1200, 'pressure-millimeter-ofhg', locale='nb')
u'1\xa0200 millimeter kvikks\xfb8lv'
>>> format_unit(270, 'ton', locale='en')
u'270 tons'
```

Number formats may be overridden with the `format` parameter.

```
>>> import decimal
>>> format_unit(decimal.Decimal("-42.774"), 'temperature-celsius', 'short',
    ↪ format='#.0', locale='fr')
u'-42,8\u202f\xb0C'
```

The locale’s usual pluralization rules are respected.

```
>>> format_unit(1, 'length-meter', locale='ro_RO')
u'1 metru'
>>> format_unit(0, 'length-mile', locale='cy')
u'0 mi'
>>> format_unit(1, 'length-mile', locale='cy')
u'1 filltir'
>>> format_unit(3, 'length-mile', locale='cy')
u'3 milltir'
```

```
>>> format_unit(15, 'length-horse', locale='fi')
Traceback (most recent call last):
...
UnknownUnitError: length-horse is not a known unit in fi
```

New in version 2.2.0.

param value the value to format. If this is a string, no number formatting will be attempted.

param measurement_unit the code of a measurement unit. Known units can be found in the CLDR Unit Validity XML file: <https://unicode.org/repos/cldr/tags/latest/common/validity/unit.xml>

param length “short”, “long” or “narrow”

param format An optional format, as accepted by *format_decimal*.

param locale the *Locale* object or locale identifier

4.3.7 Versioning Functions

setvers()

const.**setvers**(*name: Optional[str] = None*) → Tuple[Union[int, str], ...]

Get or set the version. E.g.:

```
setvers() # to get the currenty set version
setlang("1.1.0") # to set a version (e.g. for testing)
setlang("") # to restore the default from pyproject.toml
```

Parameters **name** (str) – The name of the version to set. Default to None.

Returns The currently set version as a tuple.

ALL

const.**ALL** = (-inf,+inf)

A shortcut for the `portion.interval.Interval` that contains all (e.g. versions)

from_version()

const.**from_version**(*start: str, to: Optional[str] = None*) → Iterable

The simple interval starting with a certain version. E.g.:

```
>>> from_version("1.5.0")
[(1, 5, 0, '~'),+inf)
```

Parameters

- **start** (str) – The starting version
- **to** (str) – If set, the (excluded) last version. If None, there is no end version. Default to None.

Returns The `portion.interval.Interval`

ntversions()

const.**ntversions**(*self: Iterable*) → Iterable

The versions which offer a NamedTuple. E.g.:

```
@property
def VERSIONS(self) -> Iterable:
    return ntversions(self)
```

Note: This function assumes that each attribute that can contain values specific to a version has a VERSIONS attribute listing valid versions as a `portion.interval.Interval`

- `genindex`

PYTHON MODULE INDEX

b

babelwrap, 91

c

category, 96

const, 101

A

AGREE (*const.Move attribute*), 109
 ANONYMOUS (*const.PlayerType attribute*), 103
 ANY (*const.Directions attribute*), 106
 ASSIGNED (*const.ColorOption attribute*), 108
 AXES() (*const.Game method*), 107

B

babelwrap
 module, 91
 BLACK (*const.Color attribute*), 101
 BLACK (*const.PlayerColor attribute*), 104
 BLUE (*const.Color attribute*), 102
 BoardOption (*class in const*), 108

C

Categorized (*class in category*), 96
 category
 module, 96
 Category (*class in category*), 96
 category.ALL (*in module category*), 100
 CheckOption (*class in const*), 108
 CIRCLE (*const.Marker attribute*), 104
 Color (*class in const*), 101
 ColorOption (*class in const*), 108
 Command (*class in const*), 102
 const
 module, 101
 const.ALL (*in module const*), 115
 ctg() (*in module category*), 99

D

DefaultName (*class in const*), 104
 DIAGONAL (*const.Directions attribute*), 106
 Directions (*class in const*), 105
 DRAW (*const.StalemateOption attribute*), 109

F

FIGURE_HEIGHT (*const.Layout attribute*), 102
 FIGURE_WIDTH (*const.Layout attribute*), 103
 format_datetime (*in module babelwrap*), 92

format_datetime() (*in module const*), 111
 format_decimal (*in module babelwrap*), 92
 format_decimal() (*in module const*), 111
 format_list (*in module babelwrap*), 93
 format_list() (*in module const*), 112
 format_percent (*in module babelwrap*), 94
 format_percent() (*in module const*), 113
 format_unit (*in module babelwrap*), 95
 format_unit() (*in module const*), 114
 from_version() (*in module category*), 100
 from_version() (*in module const*), 115

G

Game (*class in const*), 106
 GRAY (*const.Color attribute*), 102
 GREEN (*const.Color attribute*), 102

H

HASH (*const.BoardOption attribute*), 108
 HUMAN (*const.PlayerType attribute*), 103

J

JUMP (*const.Move attribute*), 109

K

KNIGHT (*const.Directions attribute*), 106

L

Layout (*class in const*), 102

M

Marker (*class in const*), 104
 MARKER_MARGIN (*const.Layout attribute*), 103
 MARKER_SIZE (*const.Game property*), 107
 module
 babelwrap, 91
 category, 96
 const, 101
 Move (*class in const*), 109

N

NEW (*const.Command attribute*), 102

ntversions() (in module const), 115

O

OFFER (const.Move attribute), 110

ORANGE (const.Color attribute), 102

ORTHOGONAL (const.Directions attribute), 106

Outcome (class in const), 110

P

parse_version() (in module category), 100

PASS (const.Move attribute), 110

PieceRules (class in const), 105

PINK (const.Color attribute), 102

PINK (const.PlayerColor attribute), 104

PLACE (const.Move attribute), 110

Player (class in const), 103

PLAYER_FOUR (const.DefaultName attribute), 104

PLAYER_ONE (const.DefaultName attribute), 104

PLAYER_THREE (const.DefaultName attribute), 104

PLAYER_TWO (const.DefaultName attribute), 104

PlayerColor (class in const), 103

PlayersOption (class in const), 107

PlayerType (class in const), 103

POINTS_PER_INCH (const.Layout attribute), 103

PURPLE (const.Color attribute), 102

Q

QUIT (const.Command attribute), 102

R

REFUSE (const.Move attribute), 110

RESERVES_STR (const.PieceRules property), 105

RULES (const.Game property), 107

S

setlang() (in module babelwrap), 91

setlang() (in module const), 110

setvers() (in module category), 99

setvers() (in module const), 115

StalemateOption (class in const), 109

STRS (const.PieceRules property), 105

T

THREE (const.PlayersOption attribute), 107

THREE_SAME_COLOR_IN_ROW_WINS (const.CheckOption attribute), 109

TWO (const.PlayersOption attribute), 108

U

UNDO (const.Command attribute), 102

V

VERSIONS (const.Game property), 107

VERSIONS (const.PieceRules property), 105

VERSIONS (const.Player property), 103

VICTORY (const.Outcome attribute), 110

W

WHITE (const.Color attribute), 102

WHITE (const.PlayerColor attribute), 104

Y

YELLOW (const.Color attribute), 102

YELLOW (const.PlayerColor attribute), 104