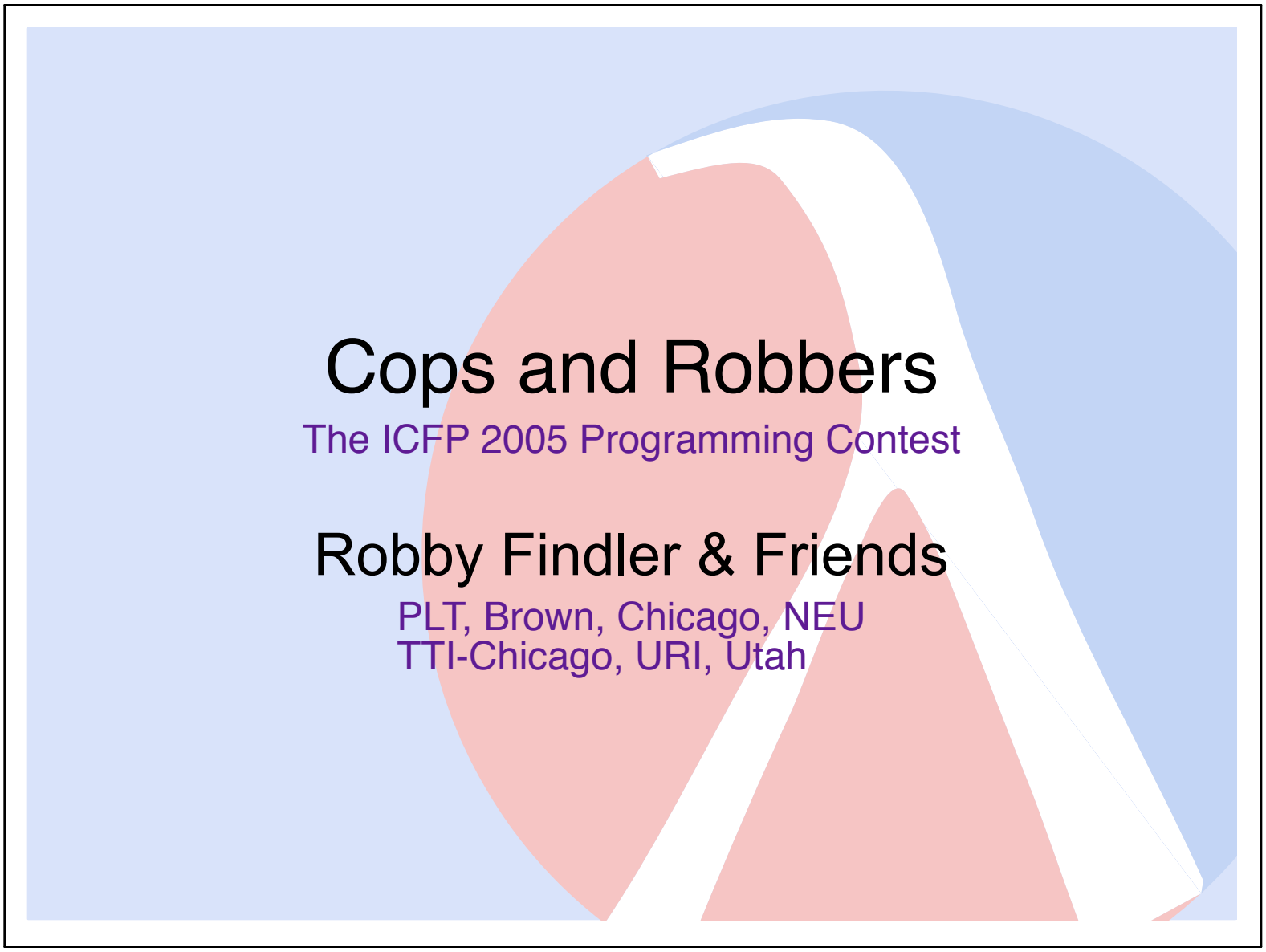

Cops and Robbers
The ICFP 2005 Programming Contest
Robby Findler & Friends
PLT, Brown, Chicago, NEU
TTI-Chicago, URI, Utah

Programming Contests

3 day sprint hacks

vs

Good programmming skills

One day in 2001, Jacob Matthews and I were talking about programming contests, and I was complaining how they purported to be measures of how good a programmer was, but that they were just hack-fests -- I would never hire a serial contest winner. The code such people produce is write once, read never.

So Jacob suggested a great idea: why not have two stages to a contest? The contestants begin as usual with the contest, but then we make them modify their code and test the modified code? This would, he argued, select for people who can write maintainable code, a true programming skill.

So, there I was. Put up or shut up. I'm not so good with shutting up, so here we are today.

Timeline

After much debate about how long the interval should be and how long the two phases should be, we settled on a two week interval with a standard three day icfp contest before and one day afterwards, to implement the twist.

Of course, we were afraid that only people who spent the entire two weeks working on their code would have a chance to win, but we counted on two things to counteract that.

First, we decided the judges' prize would be given to the team that demonstrated the best re-use of their submission. And I publicly committeed myself to reading their code, rather than trying to automate this via diff.

Second, we just tried to be sneaky enough in our twist design so that producing the world's best submission for the first round would only give a small advantage over competent submissions in the first round.

Desiderata, i

Twist:

- Easy to get a working solution
- Hard to get a good solution
- Implementing twist gives a clear advantage

Beyond those goals, we had a few other ideas in mind about how we wanted the competition to go.

First off, it should be easy to implement the twist if your code was structured well, but it should be hard to take advantage of it. And hopefully, it should be impossible to take avantage of the twist if your code is structured poorly -- just bringing your bots up to spec would take the entire 24 hours.

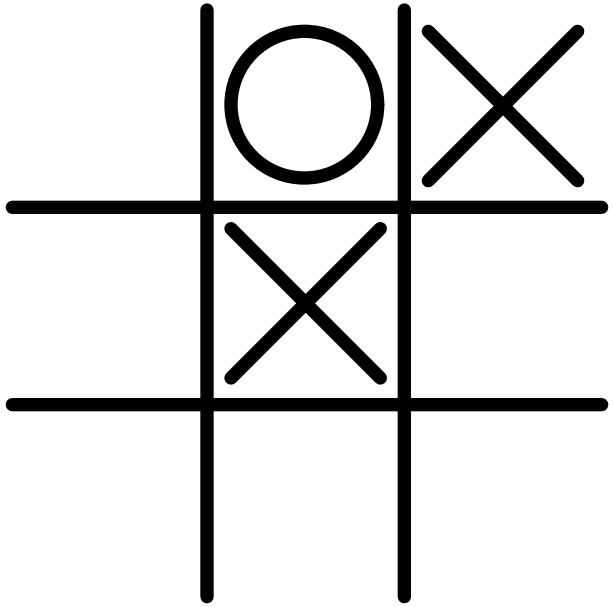
And yet, succeeding at implementing the twist should give a clear advantage.

Desiderata, ii

Too many puzzles succumb to brute force techniques and we wanted to make sure that ours would not. Instead, we wanted people to have to be able to synthesize something out of the rules of the contest and commit to some kind of a strategy up front.

Desiderata, iii

vs



I love board games and everyone who was interested in helping with the contest like them too, so it was decided early on that the contestants would implement players for some game.

For games where search is impractical or infeasible for some reason, what often happens is that the best strategy often depends on the strategies that your opponents play. In gaming circles, you sometimes hear this called the "meta-game".

To take an extreme example, assuming somehow no one knew ahead of time that all three strategies in rock-paper-scissors (ie, rock, paper, and scissors) were equally strong, and somehow people tended to gravitate towards rock -- you should definitely play paper. Some would even say that rock paper scissors truly has a vibrant meta-game. I encourage you to check out the World RPS society.

tic-tac-toe, however, has just a single viable game-time strategy -- block the opponent from getting three in a row while trying to do so yourself. There are no real strategy decisions to be made ahead of time.

So, tic-tac-toe-like games have an uninteresting meta-game, but games like rock-paper-scissors have an intersting one. We wanted our game to be in the rock-paper-scissors category.

We really didn't want the game to degenerate into who could produce the a move from the one strategy the quickest. Instead, we hoped that we could come up with a game where players would think about the rules and commit to particular strategies and, if you knew (or thought you knew) what your opponents were bringing, you could take advantage of that in your own strategy.

The game

The game

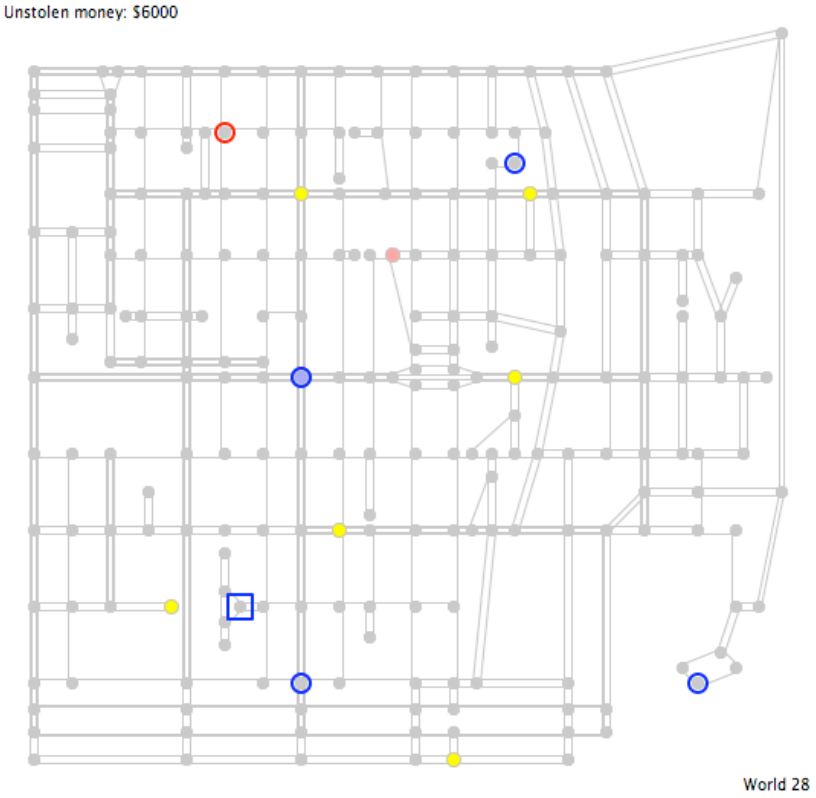
Cops and Robbers

- 6 players, one robber & 5 cops
- Each player a separate program
- Cops must catch robber while protecting banks
- Robber must evade cops while robbing banks

Each match in the game consists of six players: one robber and five cops. The cops must work together to be able to catch the robber. Each player is a separate program and will have to coordinate with other players in the game, in an at least partially cooperative way.

If the cops catch the robber, they are rewarded with the money still left in the banks. If the robber evades the cops for enough turns, it is rewarded with the money it stole from the bank.

Welcome to Hyde Park



The gray lines and circles are the streets and intersections in Hyde Park, my neighborhood (in Chicago).

The grey lines are offset from the centers of the circles to indicate directionality of the streets. Imagine you are driving on the right-hand side of the road; if you have an edge on your side the street goes in your direction and if not, it doesn't.

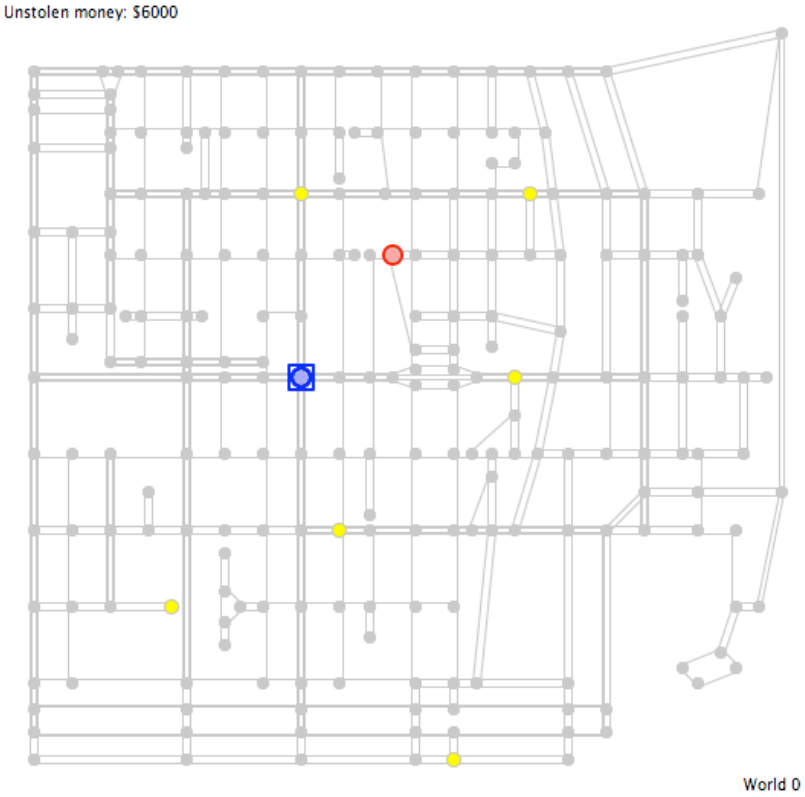
The blue circles & square are the locations of the cops and the red circle is the location of the robber.

The squared cop is the main quad of the University of Chicago. The cop in the top right is at my favorite restaurant in the neighborhood.

The yellow nodes are banks (The bottom-right one is not a bank in real life -- that's where TTI-Chicago is located).

The blue node -- in the center; a cop is standing on it -- is the cop headquarters and the red node is where the robber starts.

Initial locations

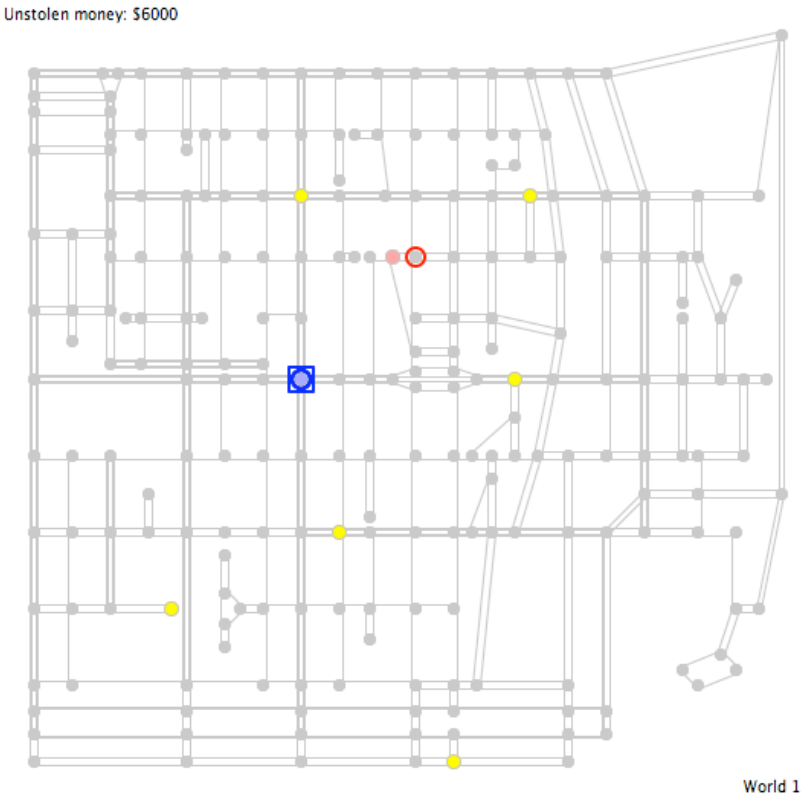


Initially, the cops are all at the cop headquarters and the robber is at the park, marked by the red dot.

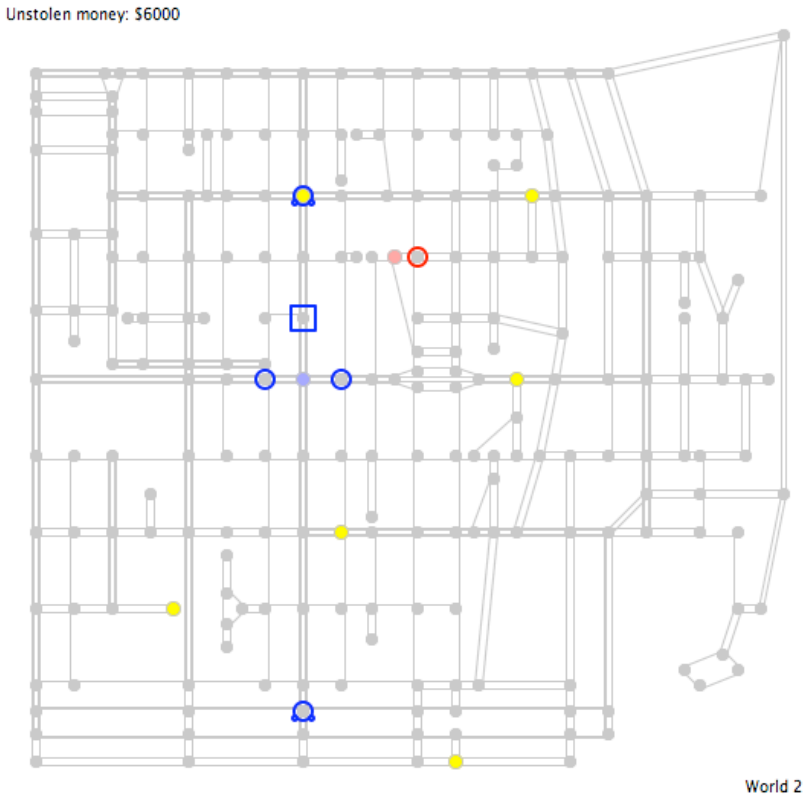
Gameplay is turn-based.

First, the robber moves.

Robber moves



Cops move



The cops communicate with each other to figure out a plan and then move simultaneously.

This pattern of moves repeats until the cops catch the robber, or 100 such pairs of moves occur, which ever comes first.

Sometimes the cops can skip over nodes, as you can see from the top-most and bottom-most cop here. Those cops are in cars and they have extra edges that let them skip ahead. But (unlike real life) cops in cars have to follow one way streets, but the foot cops and the robber do not.

Information

Public information: location of cops

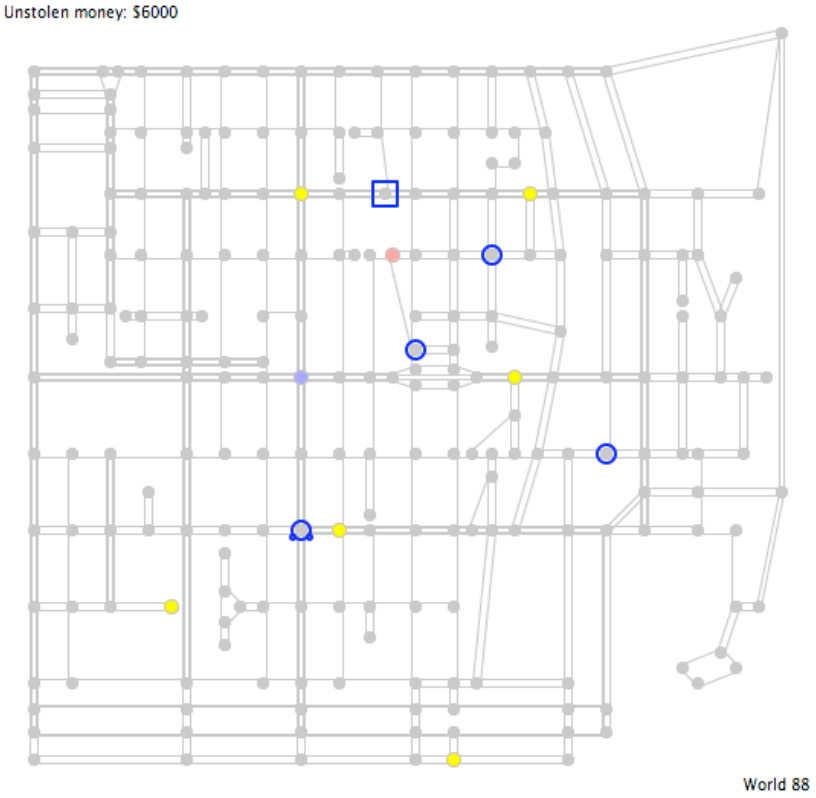
Private robber information: location

Private cop information: smells, evidence

Although everyone knows the locations of all the cops, not all of the information on the board is public knowledge.

The robber's location is private to the robber (unless the robber is on a bank). The cops each get their own partial information about the location of the robber, in two forms. First, the robber periodically drops evidence on nodes it passes. Only the cops that pass those nodes pick up the evidence. More interestingly, cops that are near to the robber can "smell" the robber. Each turn, each cop learns if it is one, two, or more steps from the robber. Once the cops learn their information, they can choose to pass that on to the other cops, or not.

Smell example



As an example, assume that the square cop is told that it can smell the robber within two steps.

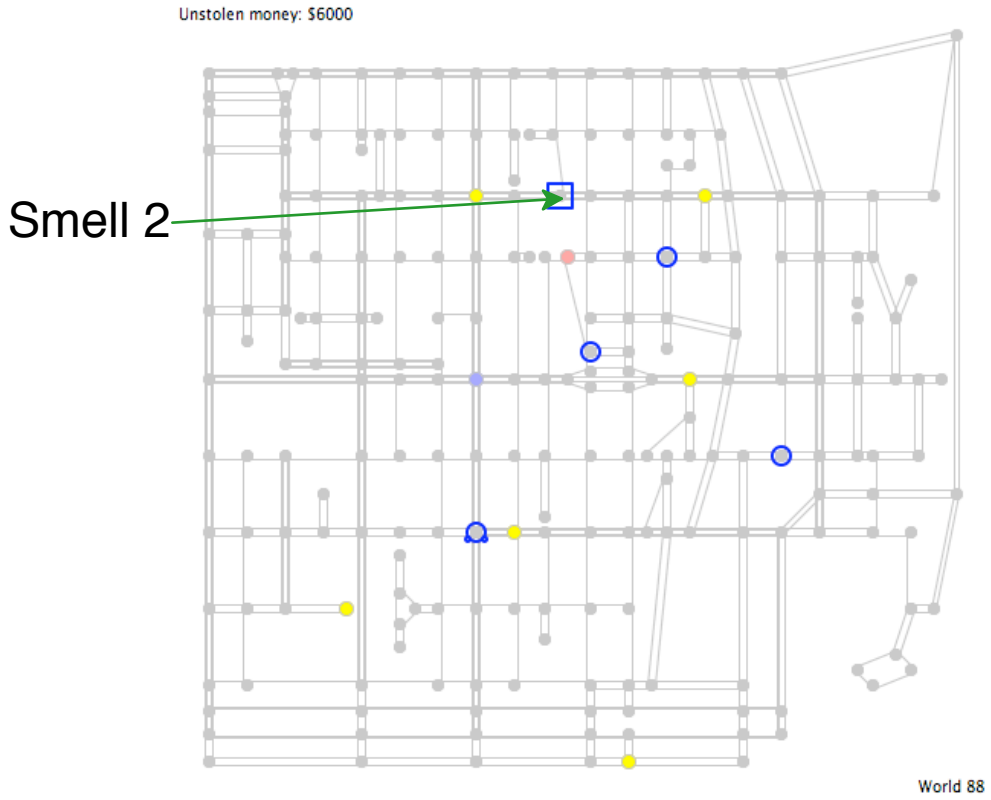
Now it can tell that the robber is on one of the red nodes.

One of the other cops can also smell the robber at a distance two and the rest of the cops cannot smell the robber.

Putting together the other cop's reported 2, we can narrow down the possible locations of the robber to just two.

But, we can also take the no smell information into account. Since the middle cannot smell the robber, the robber is not two away from it, so we can really narrow down the space to a single location for the robber.

Smell example



As an example, assume that the square cop is told that it can smell the robber within two steps.

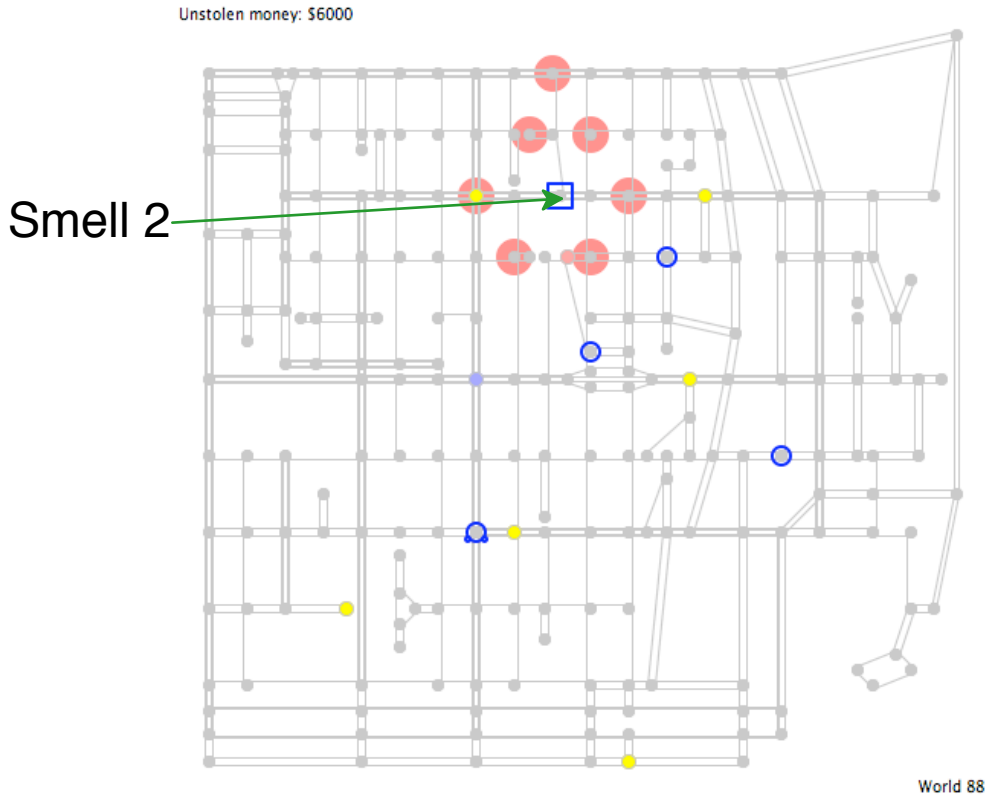
Now it can tell that the robber is on one of the red nodes.

One of the other cops can also smell the robber at a distance two and the rest of the cops cannot smell the robber.

Putting together the other cop's reported 2, we can narrow down the possible locations of the robber to just two.

But, we can also take the no smell information into account. Since the middle cannot smell the robber, the robber is not two away from it, so we can really narrow down the space to a single location for the robber.

Smell example



As an example, assume that the square cop is told that it can smell the robber within two steps.

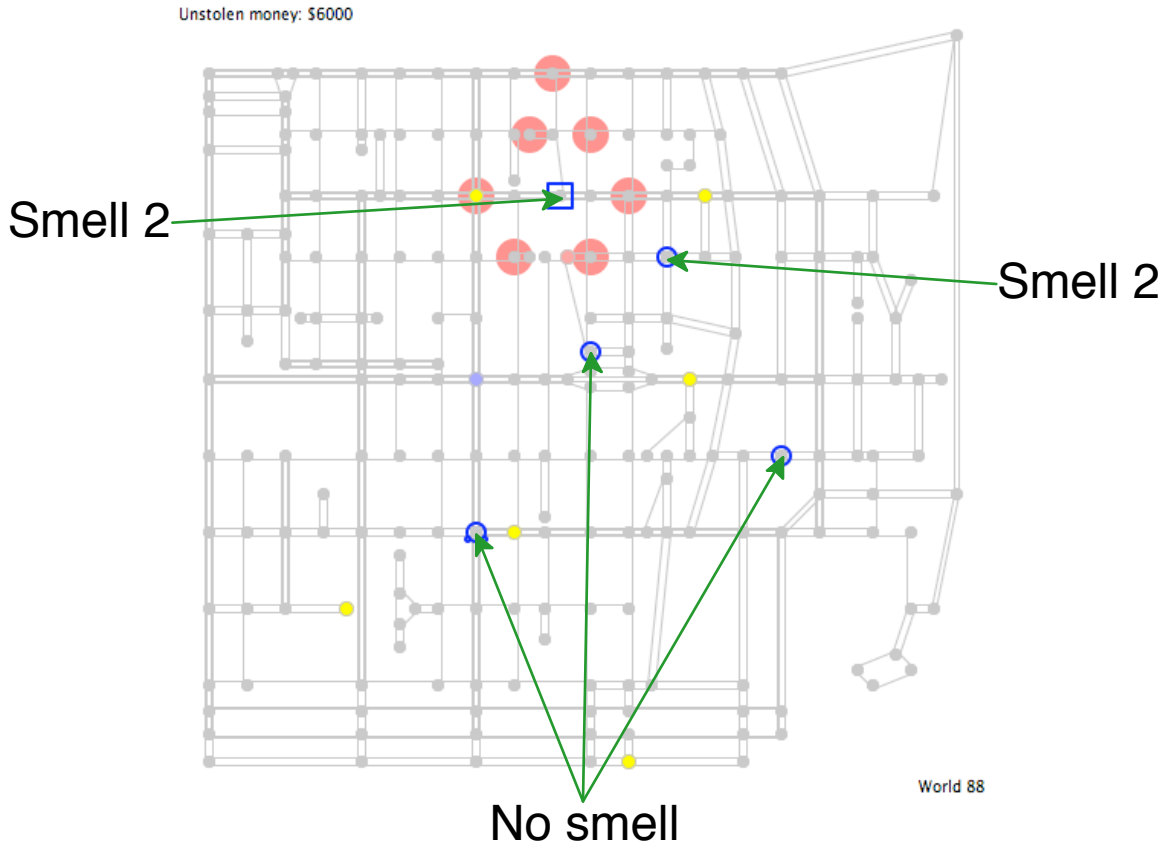
Now it can tell that the robber is on one of the red nodes.

One of the other cops can also smell the robber at a distance two and the rest of the cops cannot smell the robber.

Putting together the other cop's reported 2, we can narrow down the possible locations of the robber to just two.

But, we can also take the no smell information into account. Since the middle cannot smell the robber, the robber is not two away from it, so we can really narrow down the space to a single location for the robber.

Smell example



As an example, assume that the square cop is told that it can smell the robber within two steps.

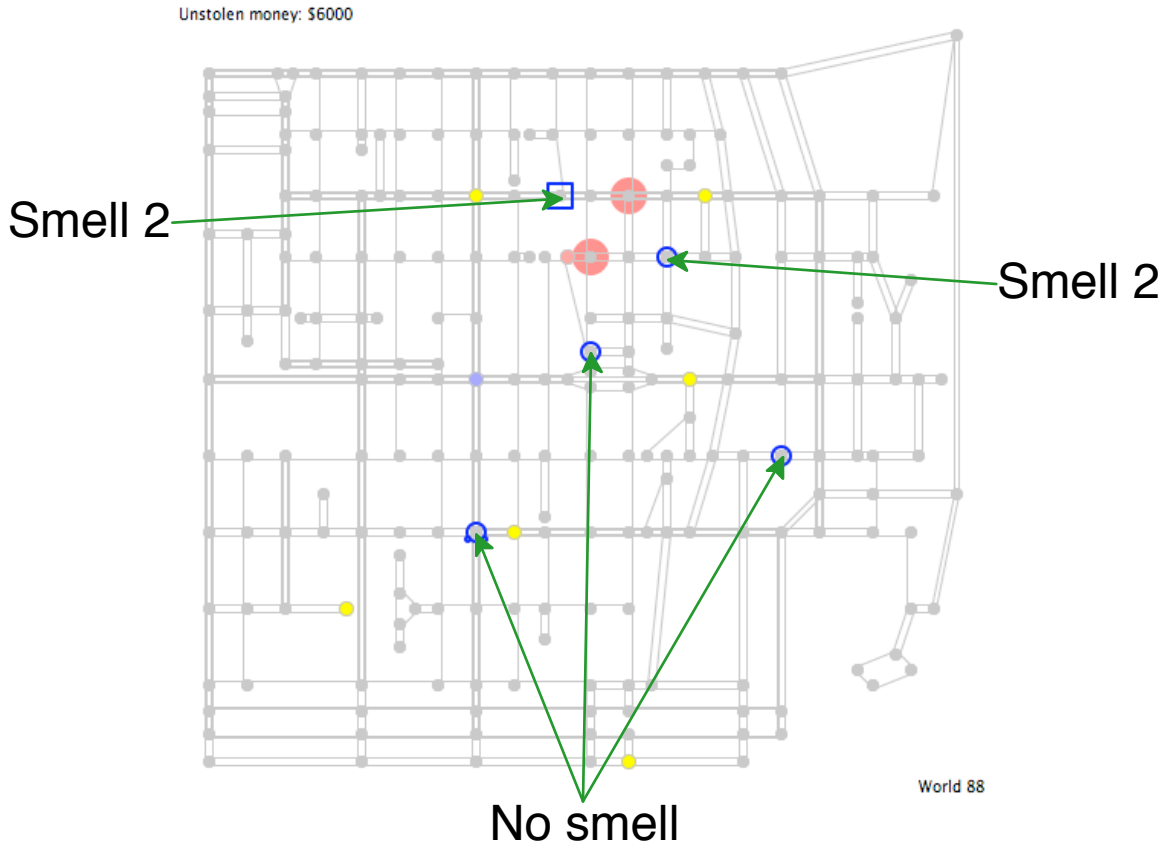
Now it can tell that the robber is on one of the red nodes.

One of the other cops can also smell the robber at a distance two and the rest of the cops cannot smell the robber.

Putting together the other cop's reported 2, we can narrow down the possible locations of the robber to just two.

But, we can also take the no smell information into account. Since the middle cannot smell the robber, the robber is not two away from it, so we can really narrow down the space to a single location for the robber.

Smell example



As an example, assume that the square cop is told that it can smell the robber within two steps.

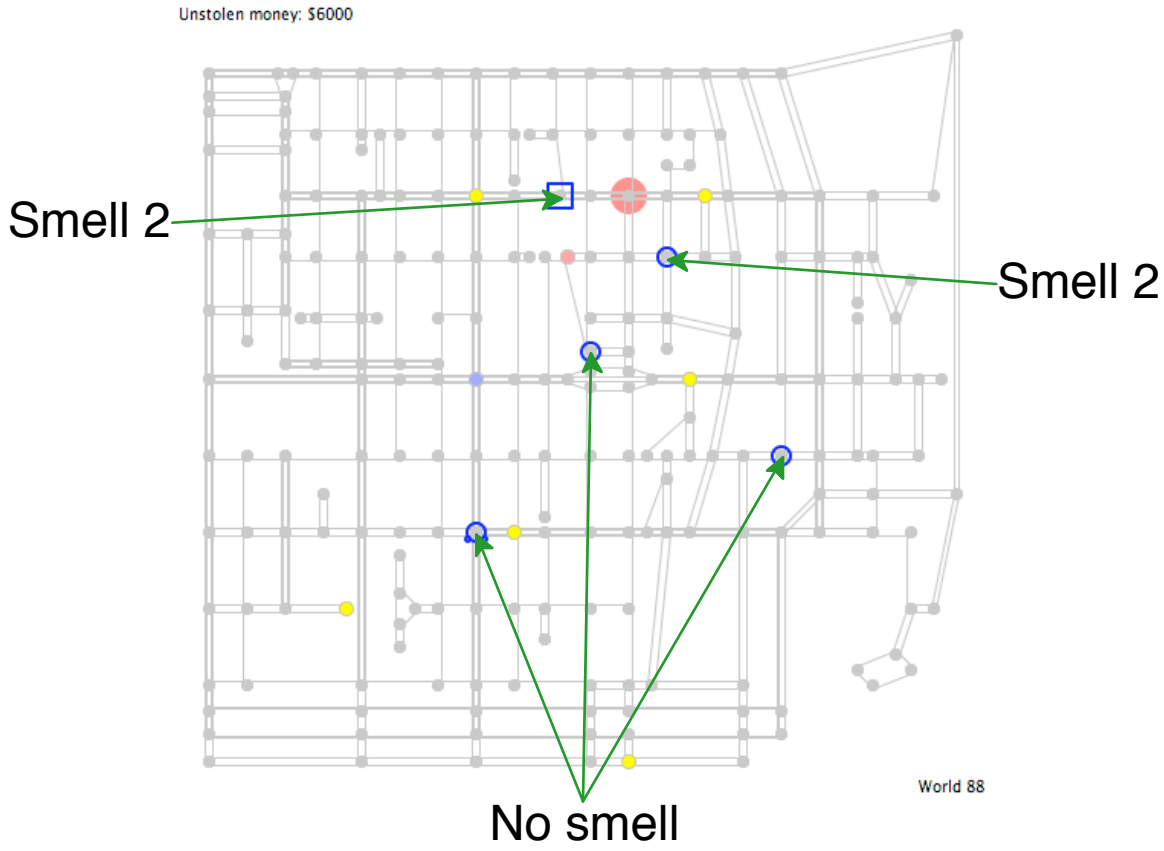
Now it can tell that the robber is on one of the red nodes.

One of the other cops can also smell the robber at a distance two and the rest of the cops cannot smell the robber.

Putting together the other cop's reported 2, we can narrow down the possible locations of the robber to just two.

But, we can also take the no smell information into account. Since the middle cannot smell the robber, the robber is not two away from it, so we can really narrow down the space to a single location for the robber.

Smell example



As an example, assume that the square cop is told that it can smell the robber within two steps.

Now it can tell that the robber is on one of the red nodes.

One of the other cops can also smell the robber at a distance two and the rest of the cops cannot smell the robber.

Putting together the other cop's reported 2, we can narrow down the possible locations of the robber to just two.

But, we can also take the no smell information into account. Since the middle cannot smell the robber, the robber is not two away from it, so we can really narrow down the space to a single location for the robber.

Cops turn, detail

Cops turn has several phases

- Cops learn their private information, and choose what to share
- Cops learn what other cops shared and formulate group plans
- Cops learn others plans and vote on plans
- Cops learn who won the vote and move

In order to make sure that computer programs have some hope of effectively communicating with each other, we designed a series of steps that the communication takes.

Each cop learns its own information and chooses what to share. It does not share the information it learns directly, however. Instead it can pass along assertions saying that a player is or is not on a location on a particular time, and they can indicate how sure they are of this fact.

The cops then learn what the others shared, and based on that formulate plans for the entire group of cops. Then, they receive everyone's plans and vote on them. The server computes the winner and reports that, and then the cops move.

The vote is non-binding, however, so the cops may move as they wish.

Scoring

If the cops catch the robber

- Cops split money in banks, 0 - 6000 pts
- Robber gets nothing
- Cops get three 60 pt bonuses

If the robber evades the cops

- Cops get two 60 pt bonuses
- Robber gets money stolen from banks

Cop goal: be selfless, but not too selfless

The match ends when after 200 different worlds have passed, or when the cops catch the robber, whichever comes first.

The game is scored in one of two different ways. If the cops win, they split the remaining cash in the bank. If the robber wins, it gets all of the money that it managed to liberate from the banks.

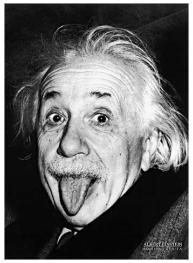
In either case, the cops receive some bonuses that serve to separate them from each other. The cops that actually land on the robber gets a bonus for doing so. The cops that collects the most evidence get a bonus, and the cop whose plans win votes the most often gets a bonus.

Tournament, i

Pods

- Each group of six plays six times, once each as robber



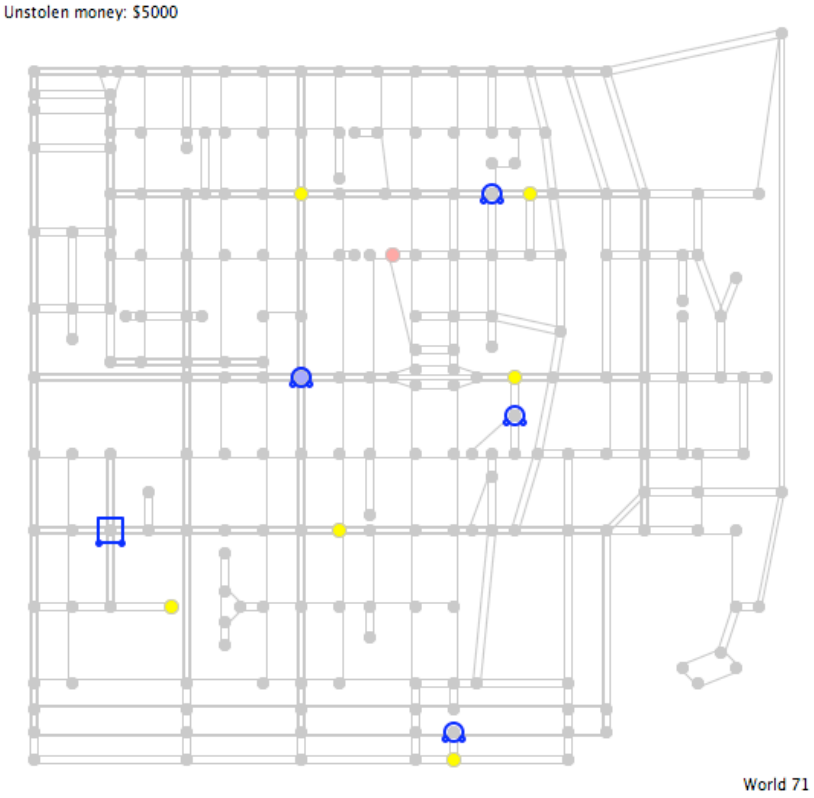



The tournament is organized into a series of pods. Each pod consists of six entries. Each entry plays as cop 5 times and each one plays as robber once.

Once problem we worried about early on was how to distinguish one genius from five idiots. If some incredibly good cop can come up with wonderful plans and even somehow gets them voted in, but no one follows the plans, and instead just wanders around randomly, the one good cop ends up chasing the robber around, but never catching the robber. So, everyone in the pod gets essentially the same scores and the genius doesn't stand out.

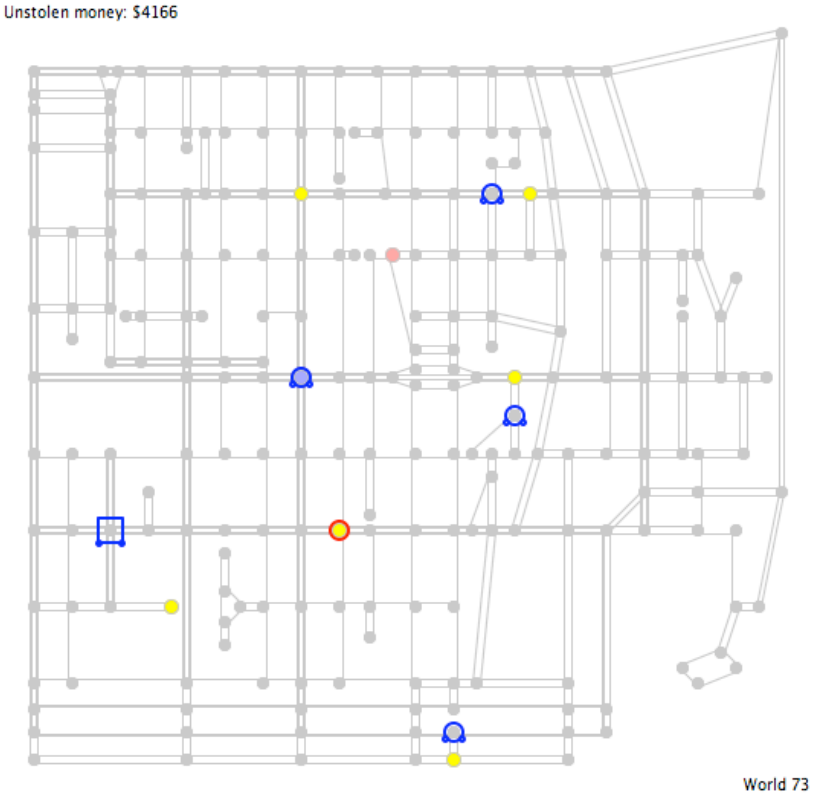
So, we built our own cop and robber and played each contestant against those.

Judge cop config 1



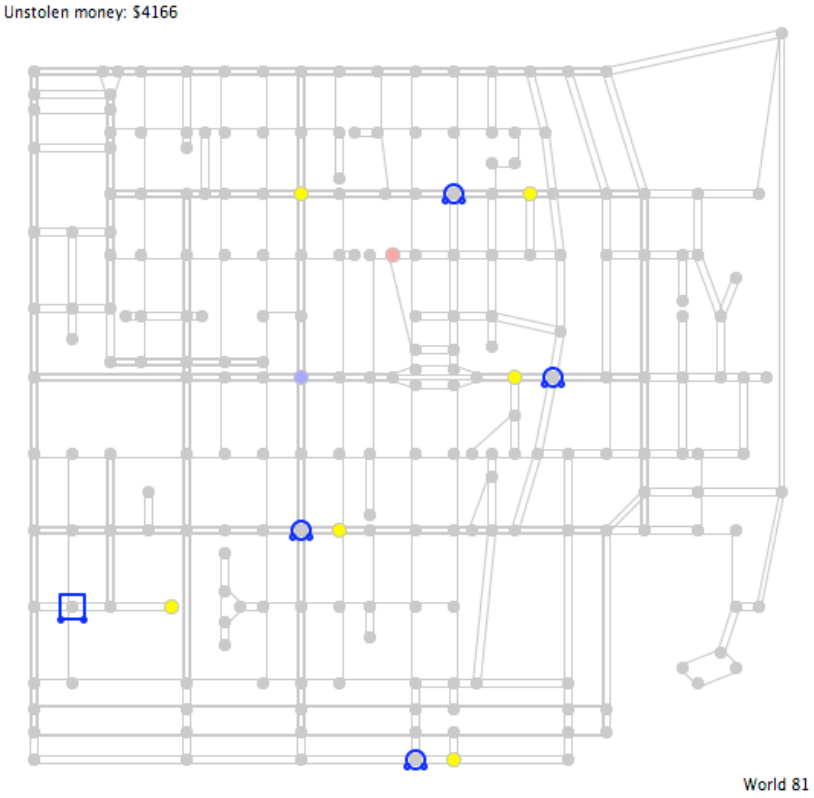
The judges cops are not very smart. They go to this configuration here which protects all but one bank.

Judge cop config 1



As soon as the robber steps on a bank, they move to the banks that they protect (ie, all but the one the robber is on here)

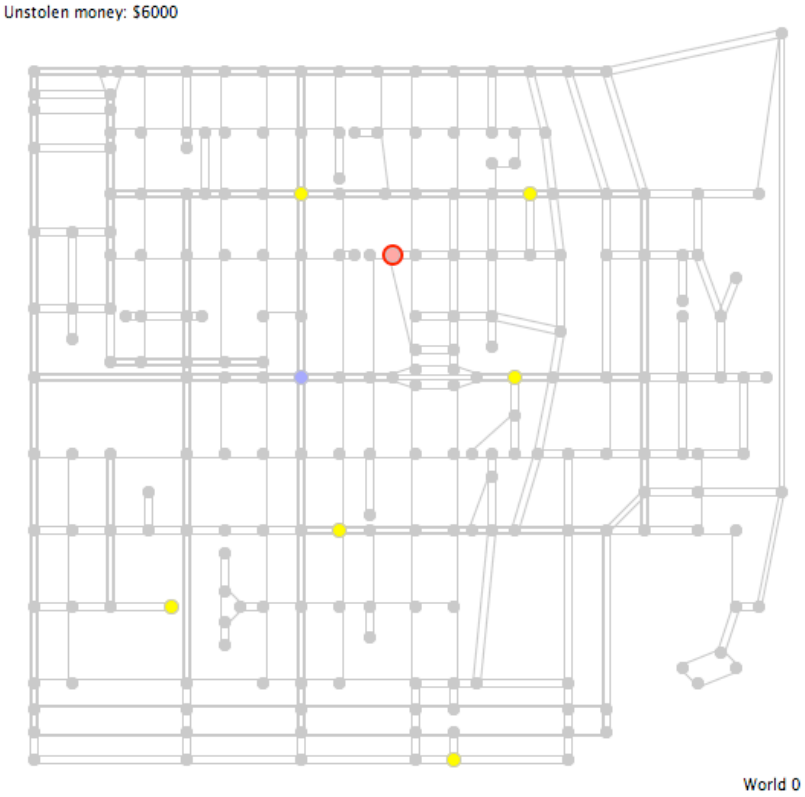
Judge cop config 2



and then they move to this configuration which, just like the other configuration protects all but one bank.

So, if your robber can sucessfully move between the unprotected banks, it will rob these cops blind.

Judge robber

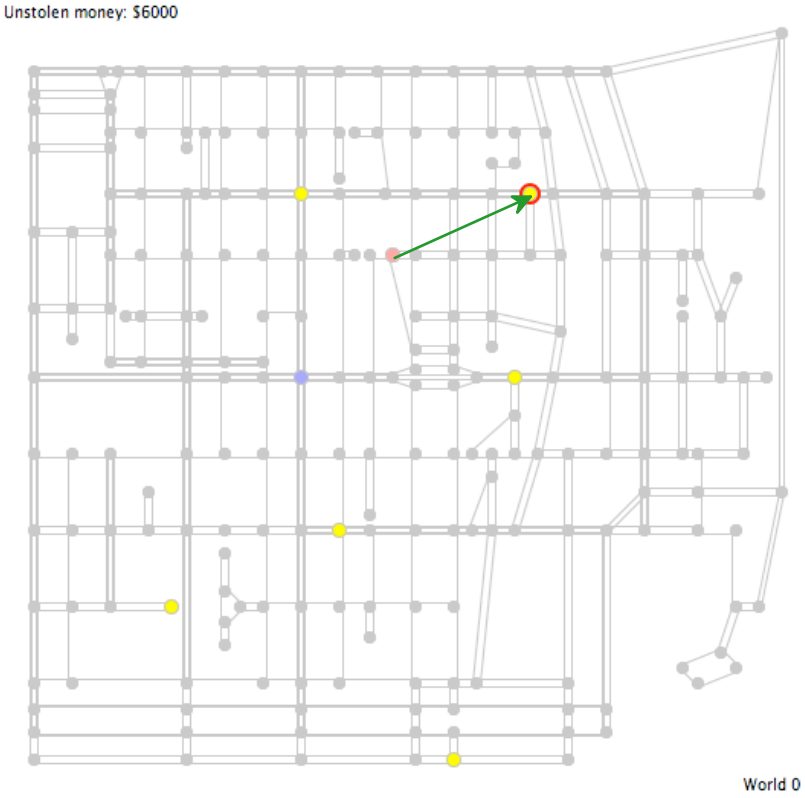


The robber is slightly smarter than the cops. It makes a tour of the banks in alphabetical order.

As it goes from between the banks, it avoids stepping next to a cop. If all moves would leave it next to a cop, it just doesn't move.

So, to catch this robber, the cops need to be able to coordinate and hem it in, but it is not too difficult to hem it in, because it goes directly between banks and so reveals its location fairly often.

Judge robber

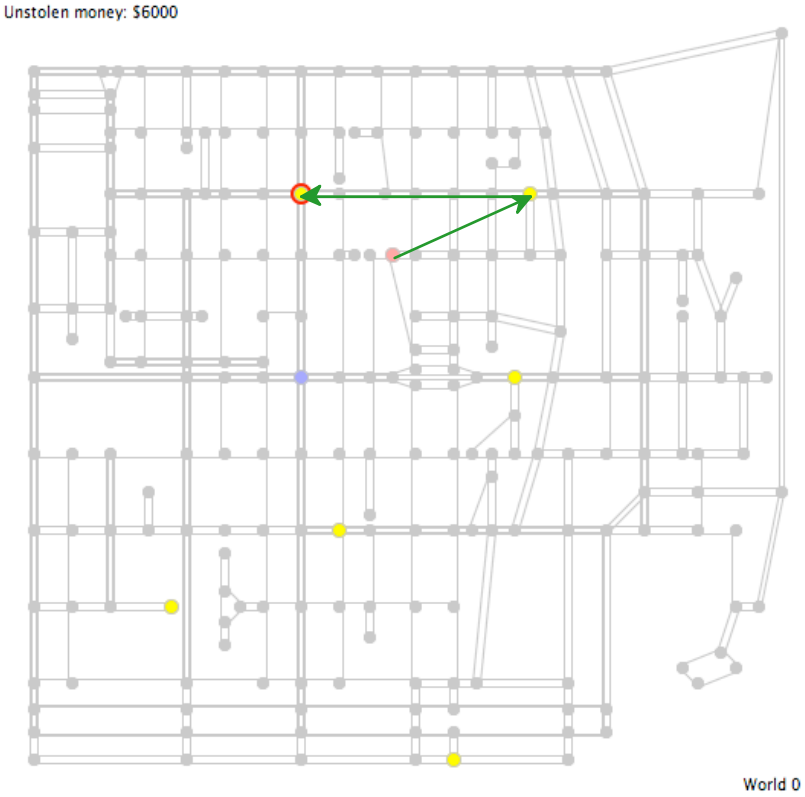


The robber is slightly smarter than the cops. It makes a tour of the banks in alphabetical order.

As it goes from between the banks, it avoids stepping next to a cop. If all moves would leave it next to a cop, it just doesn't move.

So, to catch this robber, the cops need to be able to coordinate and hem it in, but it is not too difficult to hem it in, because it goes directly between banks and so reveals its location fairly often.

Judge robber

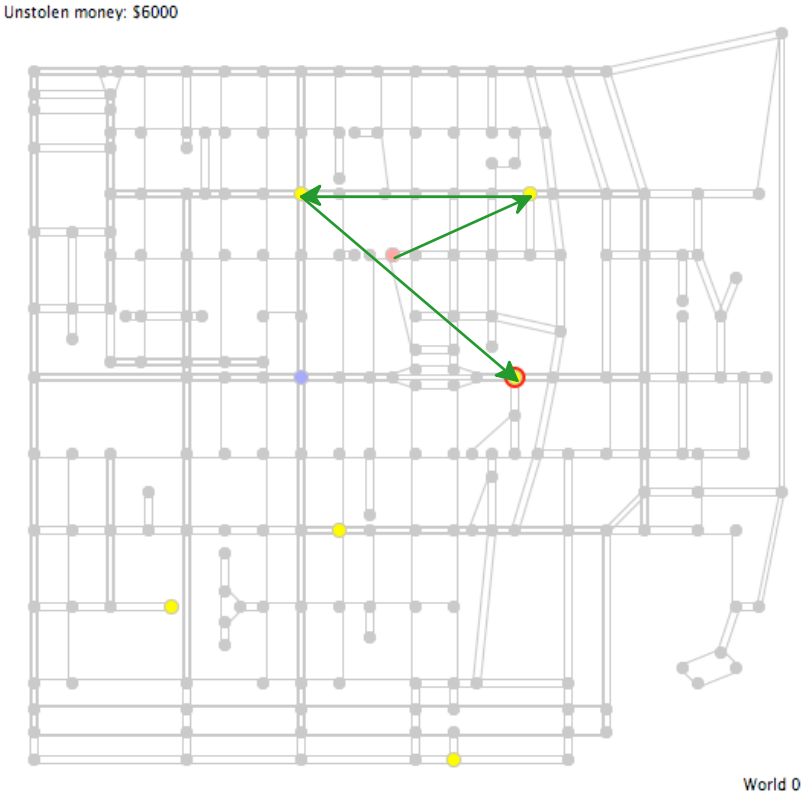


The robber is slightly smarter than the cops. It makes a tour of the banks in alphabetical order.

As it goes from between the banks, it avoids stepping next to a cop. If all moves would leave it next to a cop, it just doesn't move.

So, to catch this robber, the cops need to be able to coordinate and hem it in, but it is not too difficult to hem it in, because it goes directly between banks and so reveals its location fairly often.

Judge robber

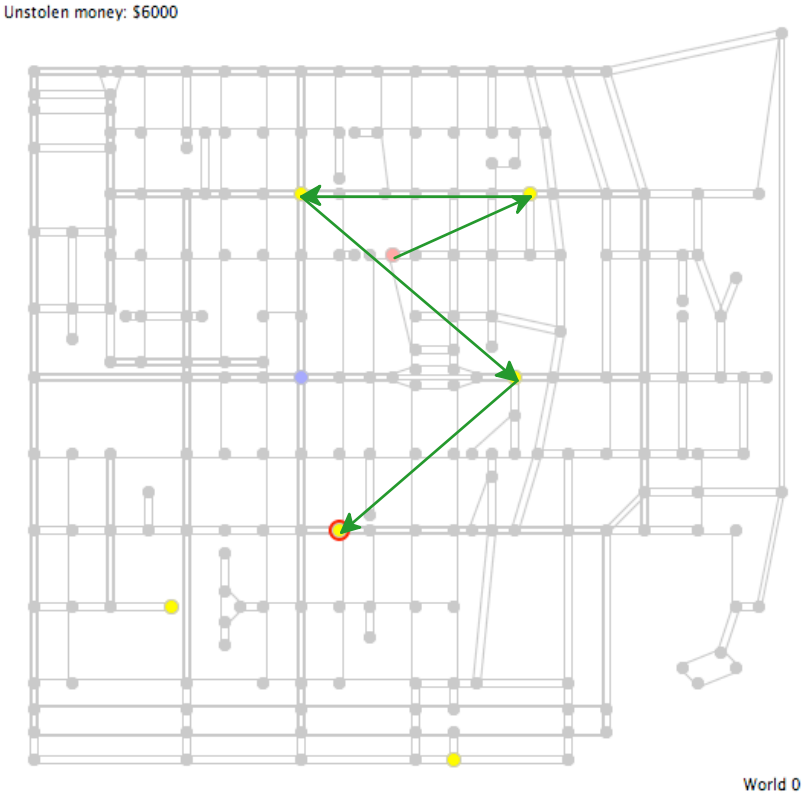


The robber is slightly smarter than the cops. It makes a tour of the banks in alphabetical order.

As it goes from between the banks, it avoids stepping next to a cop. If all moves would leave it next to a cop, it just doesn't move.

So, to catch this robber, the cops need to be able to coordinate and hem it in, but it is not too difficult to hem it in, because it goes directly between banks and so reveals its location fairly often.

Judge robber

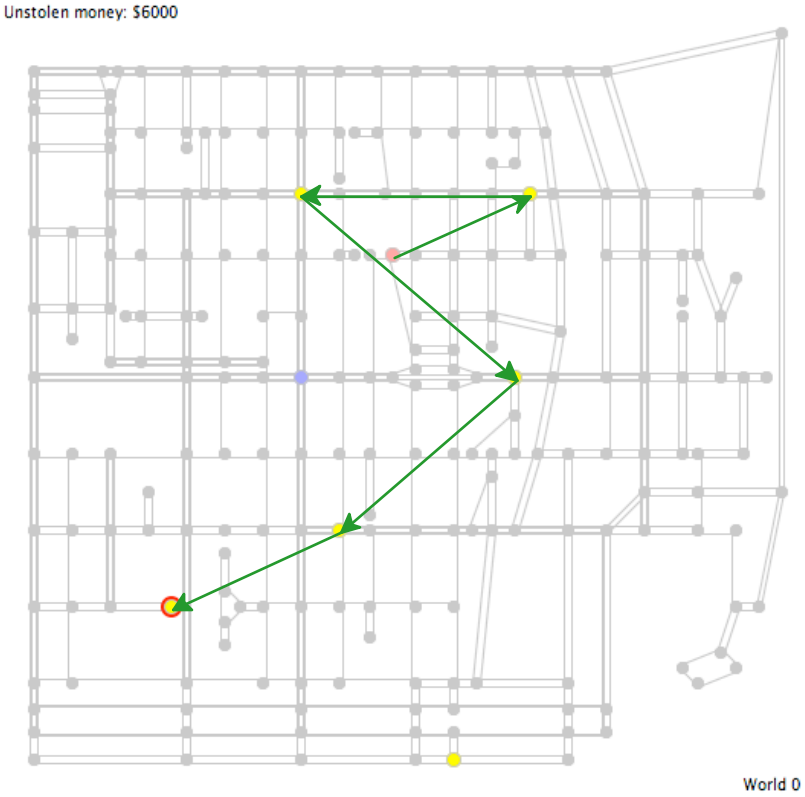


The robber is slightly smarter than the cops. It makes a tour of the banks in alphabetical order.

As it goes from between the banks, it avoids stepping next to a cop. If all moves would leave it next to a cop, it just doesn't move.

So, to catch this robber, the cops need to be able to coordinate and hem it in, but it is not too difficult to hem it in, because it goes directly between banks and so reveals its location fairly often.

Judge robber

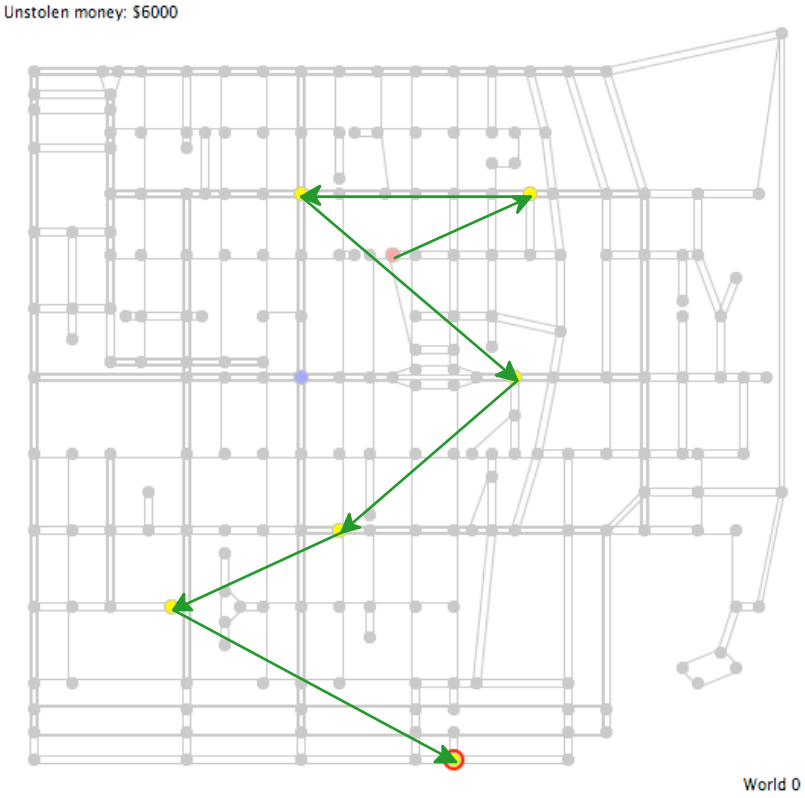


The robber is slightly smarter than the cops. It makes a tour of the banks in alphabetical order.

As it goes from between the banks, it avoids stepping next to a cop. If all moves would leave it next to a cop, it just doesn't move.

So, to catch this robber, the cops need to be able to coordinate and hem it in, but it is not too difficult to hem it in, because it goes directly between banks and so reveals its location fairly often.

Judge robber

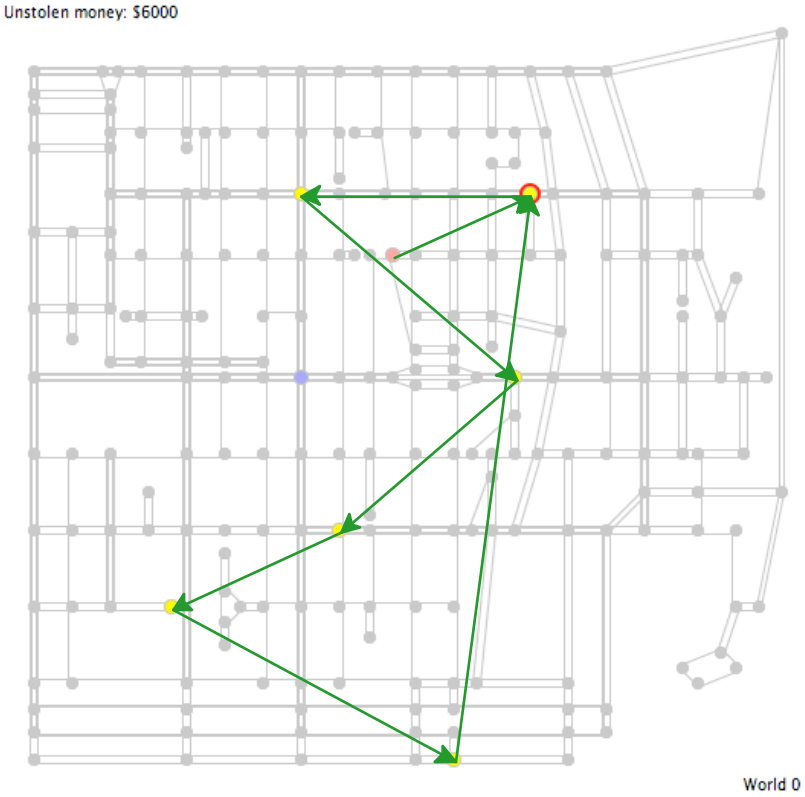


The robber is slightly smarter than the cops. It makes a tour of the banks in alphabetical order.

As it goes from between the banks, it avoids stepping next to a cop. If all moves would leave it next to a cop, it just doesn't move.

So, to catch this robber, the cops need to be able to coordinate and hem it in, but it is not too difficult to hem it in, because it goes directly between banks and so reveals its location fairly often.

Judge robber



The robber is slightly smarter than the cops. It makes a tour of the banks in alphabetical order.

As it goes from between the banks, it avoids stepping next to a cop. If all moves would leave it next to a cop, it just doesn't move.

So, to catch this robber, the cops need to be able to coordinate and hem it in, but it is not too difficult to hem it in, because it goes directly between banks and so reveals its location fairly often.

Tournament, ii

Regular season

- Play against judges cop & robber

Playoffs

- Original plan: single elimination tournament
- Revised plan: random pod selection, wait for quiescence

Once we'd eliminated all of the entries that could not beat that cop and robber, we thought we would run a single-elimination tournament but the tournament ended up being far too sensitive to the pod groupings (and to the random number seed, apparently).

So instead, we just pick random subsets of size six, run a pod, and record the rank (one thru six) of the entrants. We keep a running average of the ranks and keep picking pods until the ranks settle down.

Programming task

Implement a cop & a robber

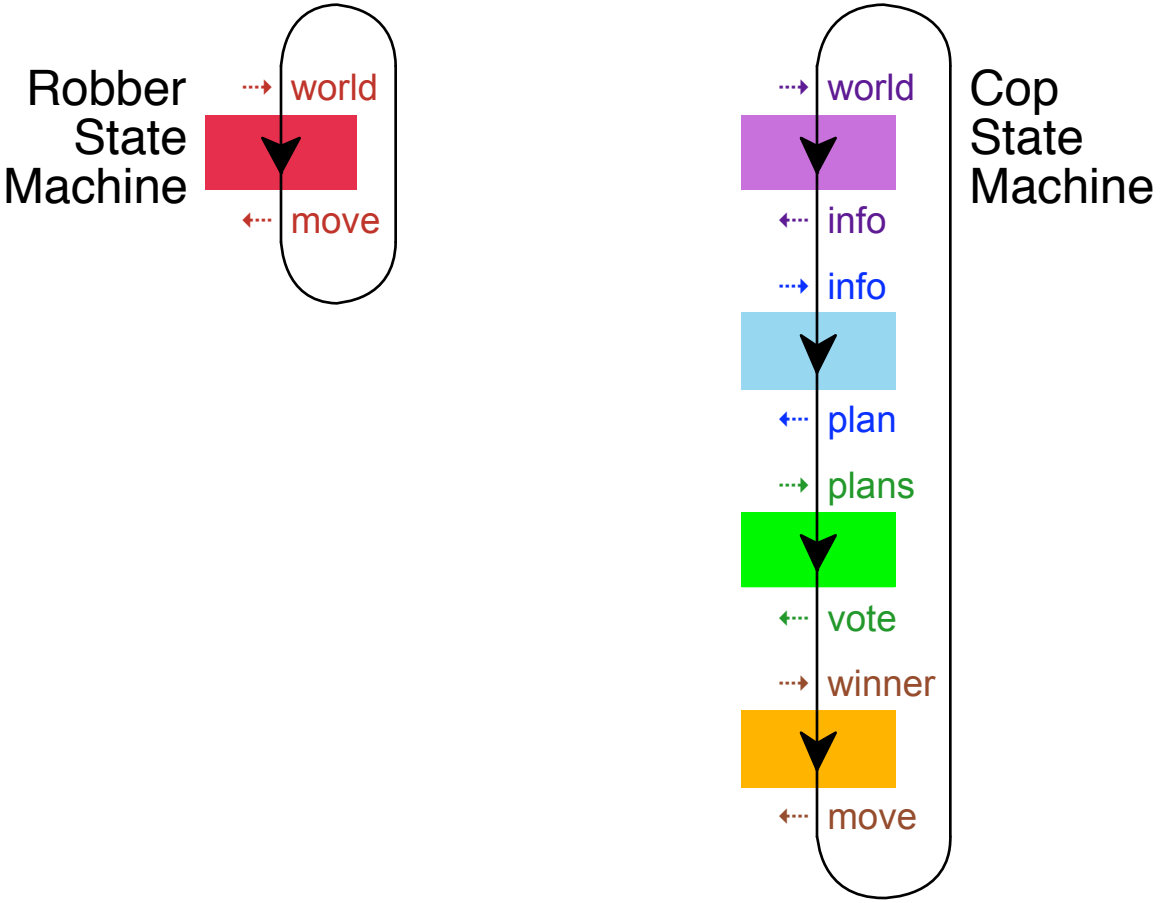
- Communicate via stdin/stdout
- Follow spec protocol

```
wor\
wor: 198
rbd: 5434
bv\
bv: 53-and-the-other-lake-park 94
bv: 53-and-woodlawn 96
bv: 55-and-harper 93
bv: 57-and-kimbark 92
bv: 58-culdesac 96
bv: 60-and-blackstone 95
bv/
```

```
ev\
ev/
smell: 0
pl\
pl: McGruff2 55-and-woodlawn cop-car
pl: McGruff3 55-and-woodlawn cop-car
pl: McGruff4 55-and-woodlawn cop-foot
pl: McGruff5 55-and-woodlawn cop-foot
pl: NoOpCop 55-and-woodlawn cop-car
pl/
wor/
```

The programming task is to implement a cop and a robber. They communicate via stdin and stdout, according to a set protocol. In black is an example messages and other messages look similar. This message coveys the private information that a cop receives at the beginning of the cops turn. In this case, it is near the end of the game (world 198) and the robber has been successful so far, robbing nearly all of the money in the banks. The smell line indicates that this cop cannot smell the robber this turn. The pl lines show the players that this cop can detect.

Programming task



These state machines show the order in which messages are sent and received for the two bots. They just show the main loop (there are also initialization and game over messages that are not shown) so you can get a sense of the kind of communication.

The robber is simple: it accepts a world and produces a move message, ad infinitum.

The cop machine follows the protocol discussed earlier: it gets private info, and then sends back what it wishes to share. This is then sent to all of the cops, and then each cop responds with a plan. The plans are shared and the cops vote. Finally the cops move, but with no obligations to follow the winning plans.

BDK

We supplied a BDK (bot development kit, natch)

- We wrote a rules game manager
- We wrote a GUI cop and robber
- We wrote record/replay transcript tools
- We wrote simple bots

Just like the real world, the problem we supplied this year is complex and has special cases that interact with each other in interesting ways, especially since we picked the map from Hyde Park! To make the contestants experience even more like the real world, we supplied programming tools to help manage the complexity.

We tried to make our rules manager have especially good error reporting when contestants fail to follow the protocol or violate some other of the rules (much of the code is error checking, in fact). Our GUI cop and GUI robber allow contestants to play against their own bots and experiment with various situations. Combined with the record and replying tools, they can even build automated test suites from situations they set up in the GUI. We also provided them a few simple bots: a cop and robber that just sit there, and a cop that always votes its own plans down, and always follows the winning plan to help contestants experiment with their own plan-making strategies.

The twist

We did a little bit of red-herring seeding in the original version. For example, during the startup of the bots, we supplied the entire map in the first message, rather than just leaving it in the spec, even though we never planned to change the map. Judging from contestant comments, we fooled a few. :)

The twist

Cops can choose to team up with the robber

- A cop offers itself to the robber, robber accepts ⇒ dirty cop
- A dirty cop wins if the robber wins and loses if the clean cops win

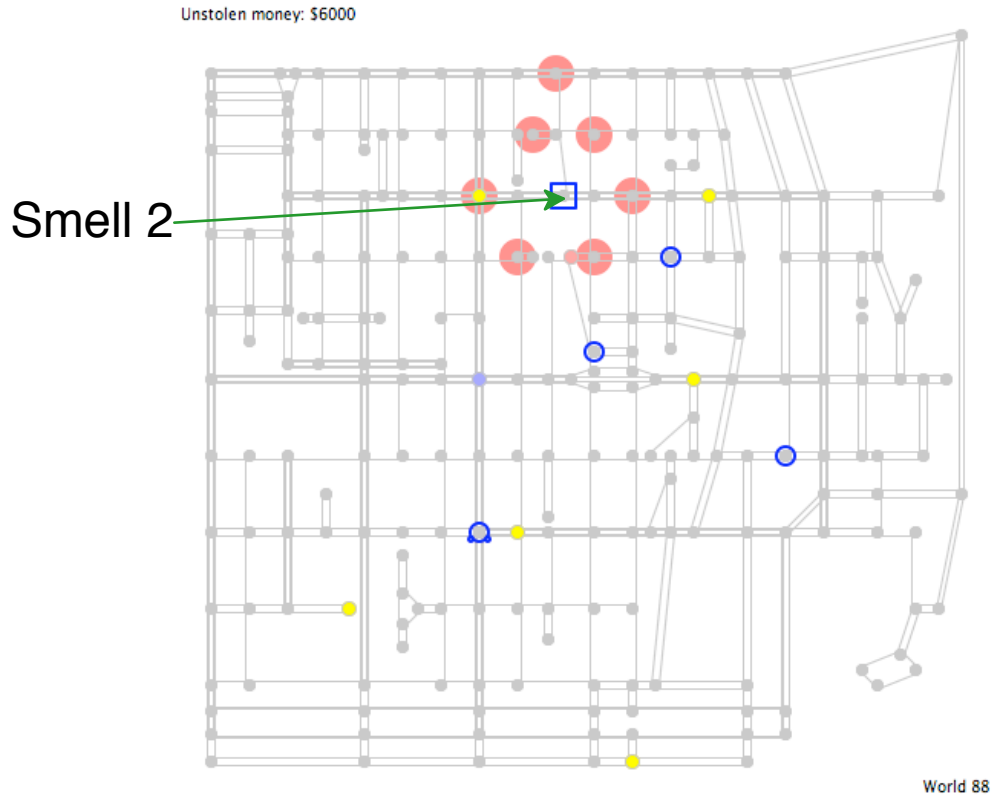
The twist was that cops can now voluteer themselves to team up with the robber.

The robber must agree in order for a cop to become officially dirty.

Once dirty, the cop has thrown its lot with the robber. It loses if the robber loses and if the robber wins, it gets a share of the loot.

Of course, once dirty, a cop wants to lie to its fellow cops to throw them off of the trail.

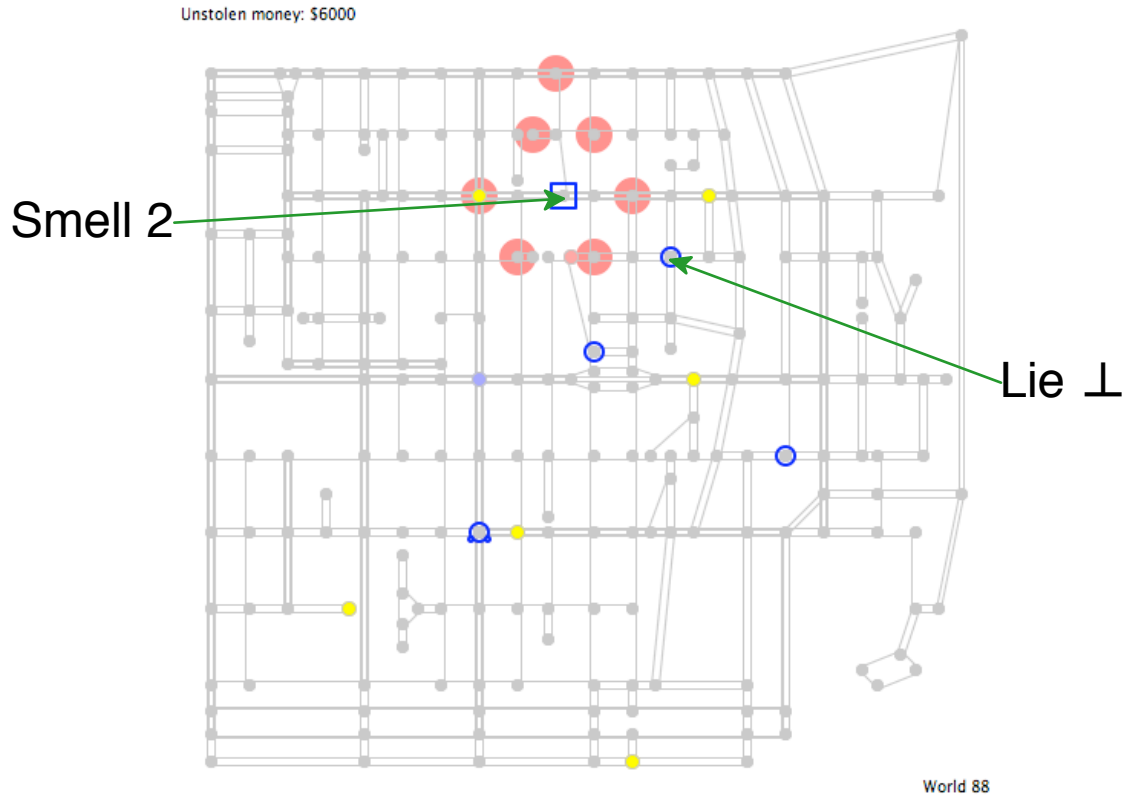
Smell lie



As before, imagine that we smell the robber two away.

But now, the neighboring cop lies and says that it does not smell the robber. If we believe it, we now are going to be sent in the wrong direction.

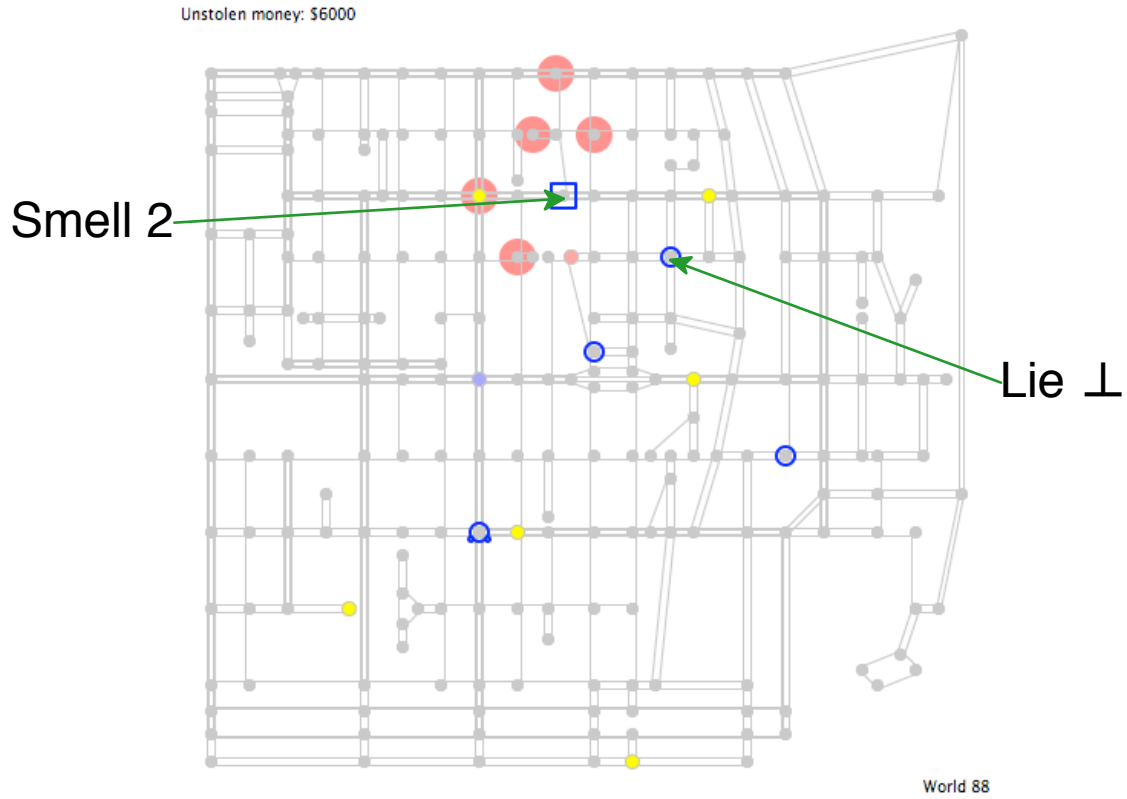
Smell lie



As before, imagine that we smell the robber two away.

But now, the neighboring cop lies and says that it does not smell the robber. If we believe it, we now are going to be sent in the wrong direction.

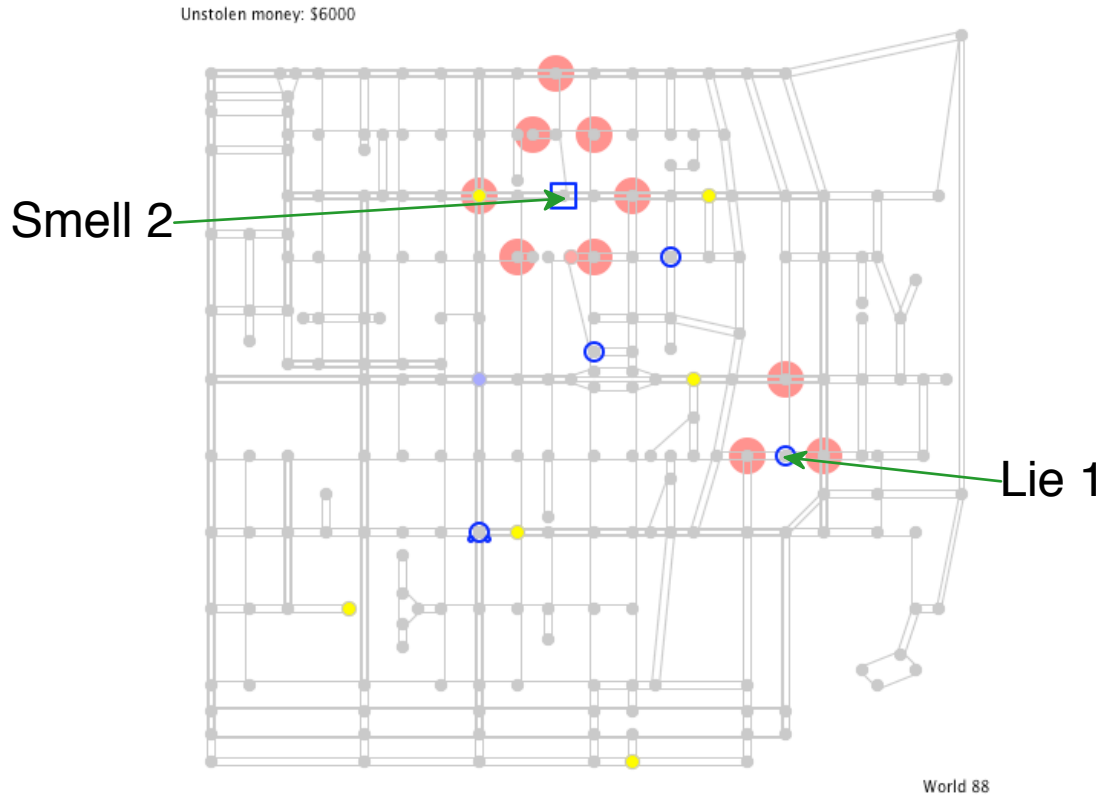
Smell lie



As before, imagine that we smell the robber two away.

But now, the neighboring cop lies and says that it does not smell the robber. If we believe it, we now are going to be sent in the wrong direction.

Bad smell lie



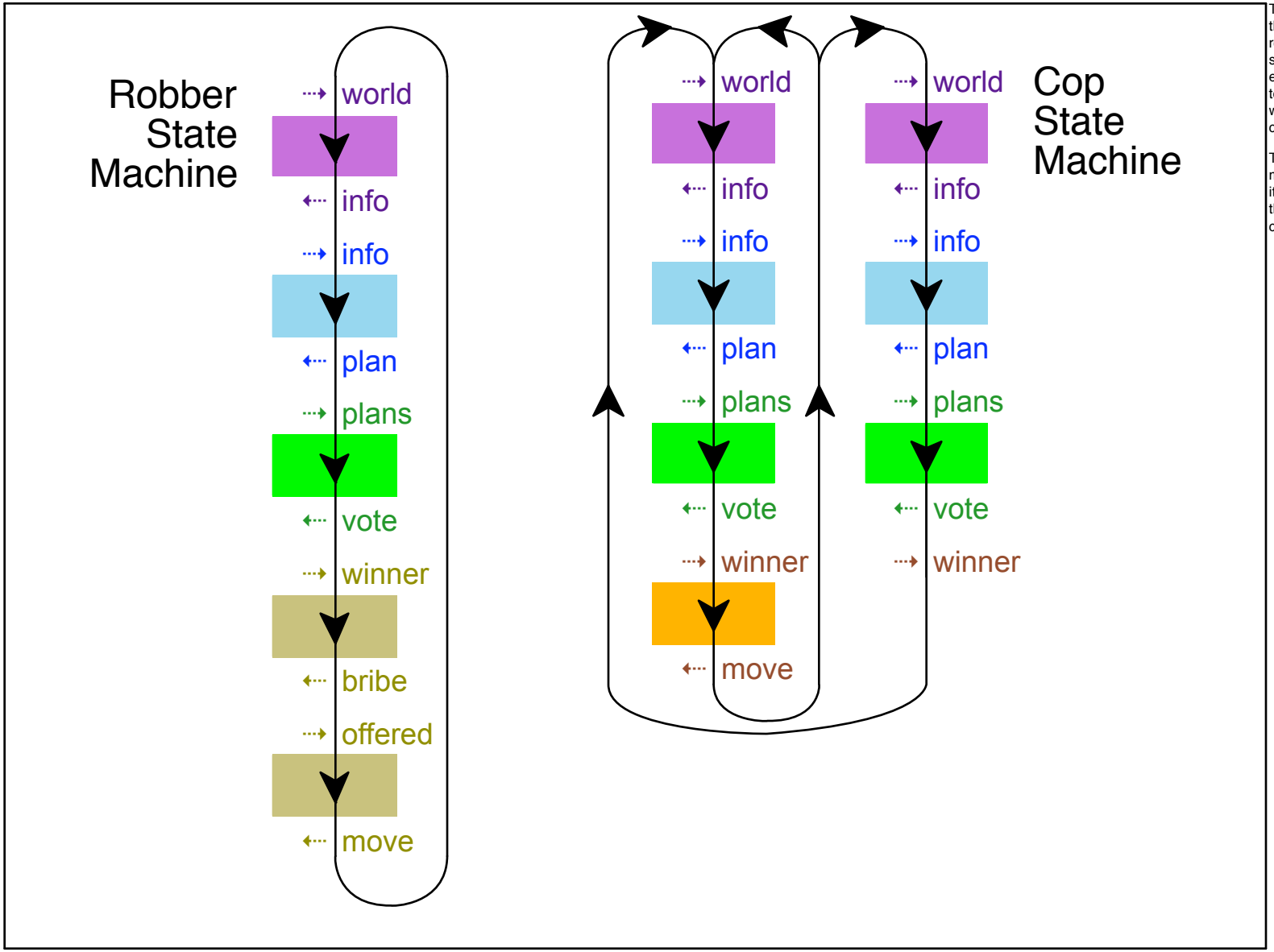
But, cops can also lie poorly. For example, if the cop to the bottom right says that it smells the robber one away from itself, we know that it is lying and we can then infer that it is dirty and accuse it.

Twist gameplay changes

- Robber now communicates with dirty cops
- Clean cops can accuse and take over dirty bots
- Minor change to map (removed degenerate strategy)

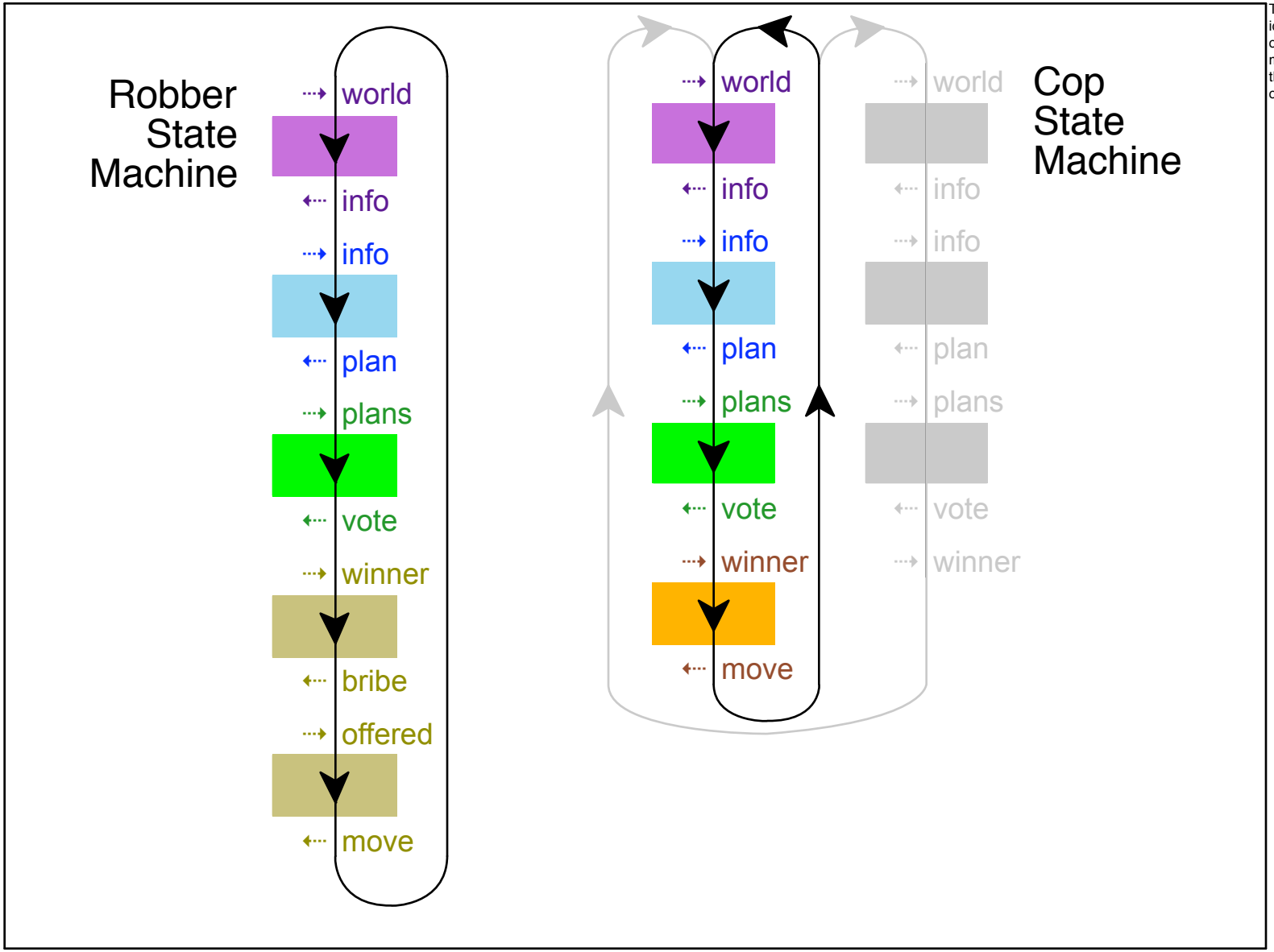
To fully support the notion of dirty cops, we must allow the robber to communicate with the dirty cops, just like the clean cops communicate. In addition, when a clean cop accuses a dirty cop, the clean cop is now in control of the dirty cop and must then submit multiple moves.

Finally, we did end up changing the map, but only in a very minor way, to remove a degenerate strategy.

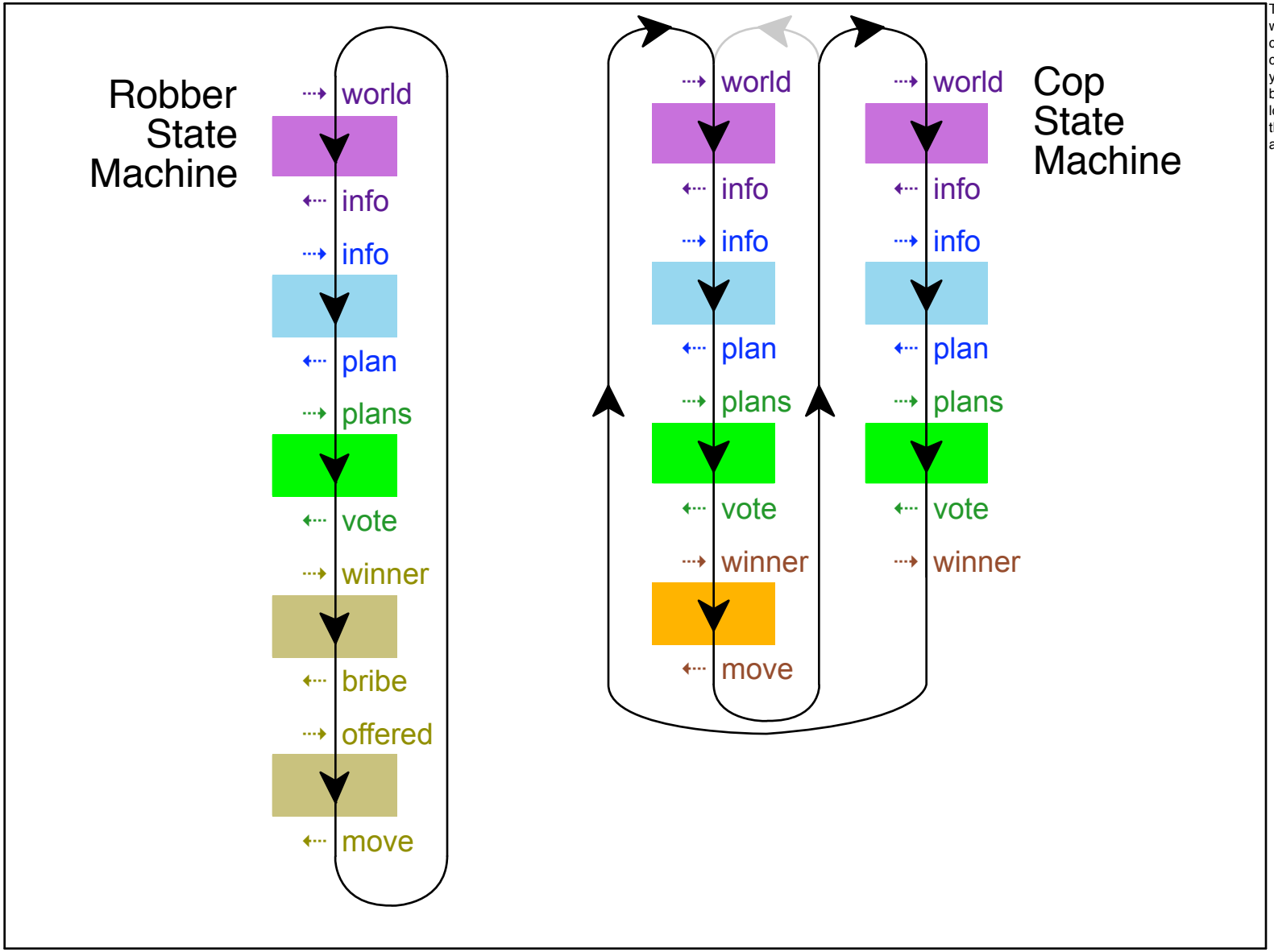


These are the new state machines for the robber and the cop bots. The robber state machine is now very similar to the old cop state machine, except that it has one extra round-trip to the server, where it indicates if it wants to bribe cops and is told which cops are available.

The new cop state machine looks a bit more more complex, but only because it has two state machines combined, the one for dirty cops and the one for clean cops.



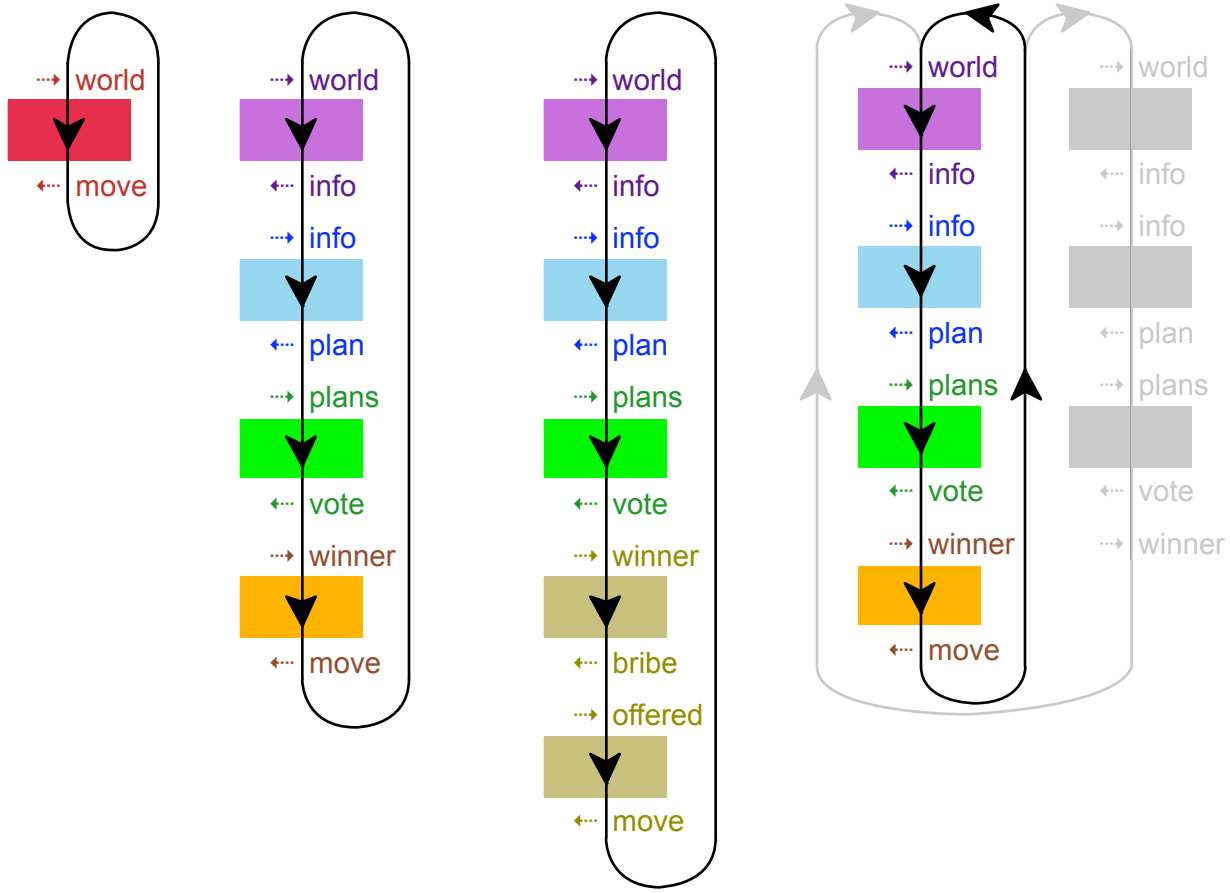
The state machine for clean cops is identical to the state machine for original cops. The content of the messages has changed a little bit, but the changes are only constant print outs, if the cop never becomes dirty.



The state machine for dirty cops begins with the same states as the clean cops, but then veers off for some communication with the robber. But as you will notice, the communication between the dirty cops and the robber looks much like the communication with the cops, except that it does not end in a move.

Desiderata revisited

Desiderata, i



Based on the state machines alone, you can see that it is easy to implement the basic structure of a twist cop if you have a working cop. I would say that it also should be easy to implement a naive dirty cop, since the conversation logic is very similar to the clean cop.

And, of course, if you can share your state machine between the cop and the robber, implementing the new robber spec is also not difficult.

Desiderata, i

Special bonuses for cops that try to become dirty

If robber looks for dirty cops and doesn't find any, (and there never have been any before) robber can push clean cops around

We added these special rules to encourage people to even implement the twist at all. The hope was that people who used these special features would win outright if no one else in their match implemented the twist.

Desiderata, ii

3.6 x 10^{13} different boards (just counting players positions)

games can be 100 moves, expect 5000 possible moves on average

barring clever search space collapses:

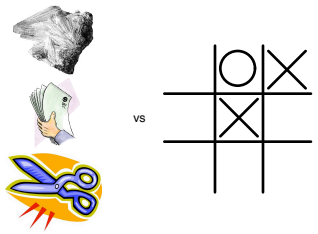
- two move lookahead: 122 gigabytes
- three move lookahead: ½ petabyte

These numbers were calculated under optimistic assumptions. For example, the search space for the board only counts boards that have the same players in different places once.

Typically, you expect to have an even mixture of car cops and foot cops to be able to move around the board effectively, which means that there are, on average, slightly less than 5,000 moves from a given board, which is what was used to compute the numbers you see there. The worst case, tho, is more than 16,000 moves which, of course, bumps up those numbers. And, I'm assuming only 47 bits per board which would probably be difficult to achieve.

In any case, it seems quite difficult to use search effectively with this game.

Desiderata, iii



- Too many credulous cops ⇒ dirty cops win
- Too many dirty cops ⇒ accusations win
- Too many accusations ⇒ sneaky cops win
- Too many sneaky cops ⇒ robber wins
- Five credulous cops catch the robber

Although we were not able to full explore the search space, it does seem likely that there is a rock-paper-scissors-like element to the game.

If you expect most of the cops to be clean and credulous, then by becoming dirty you disrupt their startegies and win with the robber. If you expect cops to be dirty and disruptive, than by detecting and accusing dirty cops, you can take them all over and win. If you expect cops to be making a lot of accusations, you can play stupid for a little while, get cops to use up their accusations and then become dirty and win. If too many cops are acting dirty without being dirty than you clearly expect the robber to win and, as you might guess, five credulous cops can catch the robber.

So, we hoped that the contestants would be discussing these issues and that a meta-game would develop and that the two week period might give a little bit of time for this to develop.


Warning

More work than I had guessed ...

Don't do this before you have a stable job

> 30k lines of code written

> 2,400 pages of code read

> 5.5 months of cpu-time

> 6 robby-months (full time)

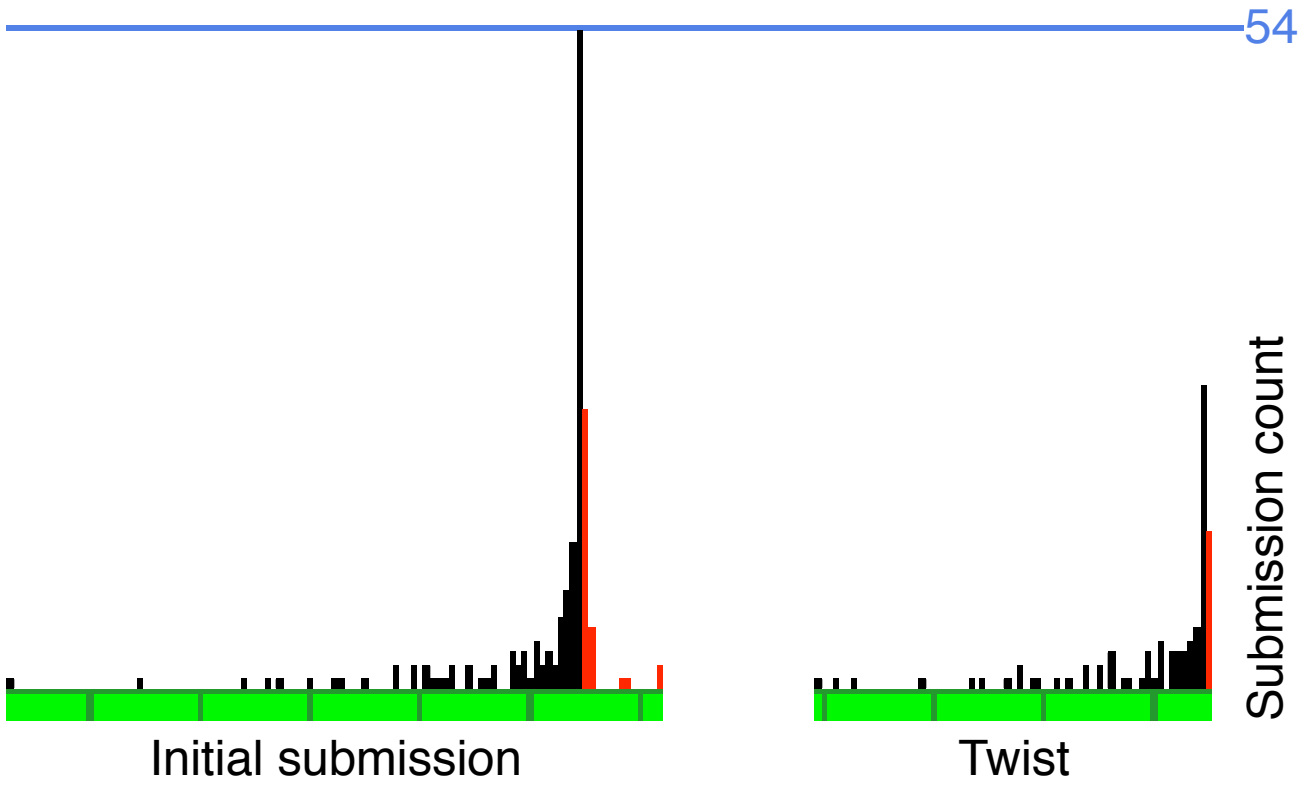
... don't do this before you have a stable job.

And really, the killer here is that last line. I don't know if I work more or less efficiently than you do, but I do know that a robby month is not just four weeks of M-F 9-5.

But enough depression.

The teams

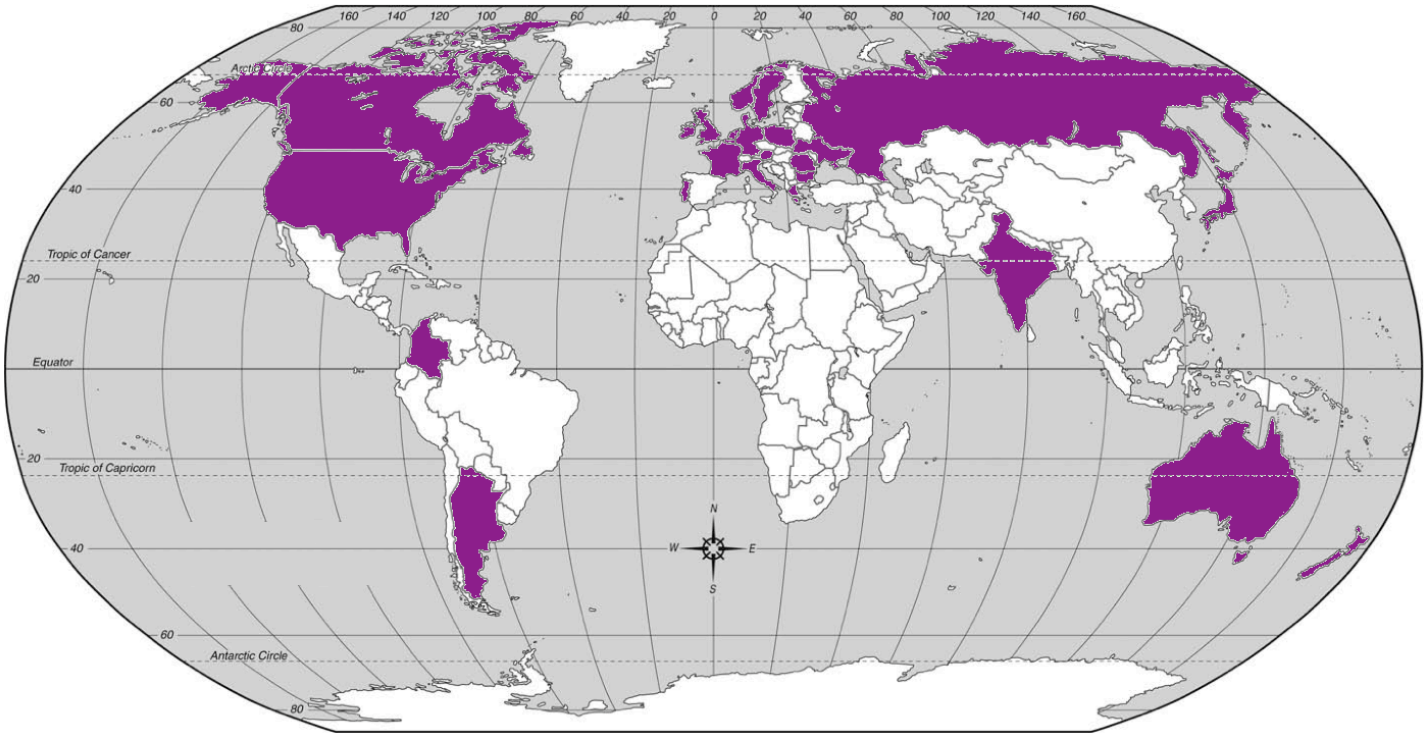
161 teams, 370 contestants



20 minutes per vertical bar
6 hours per green region

There were 161 teams, with the usual submission time graph. Here you see 6 hours in each of the light green regions.

Those last few very late teams were actually ones that we entered ourselves, since a few teams had trouble uploading their submissions and we had to enter them manually.



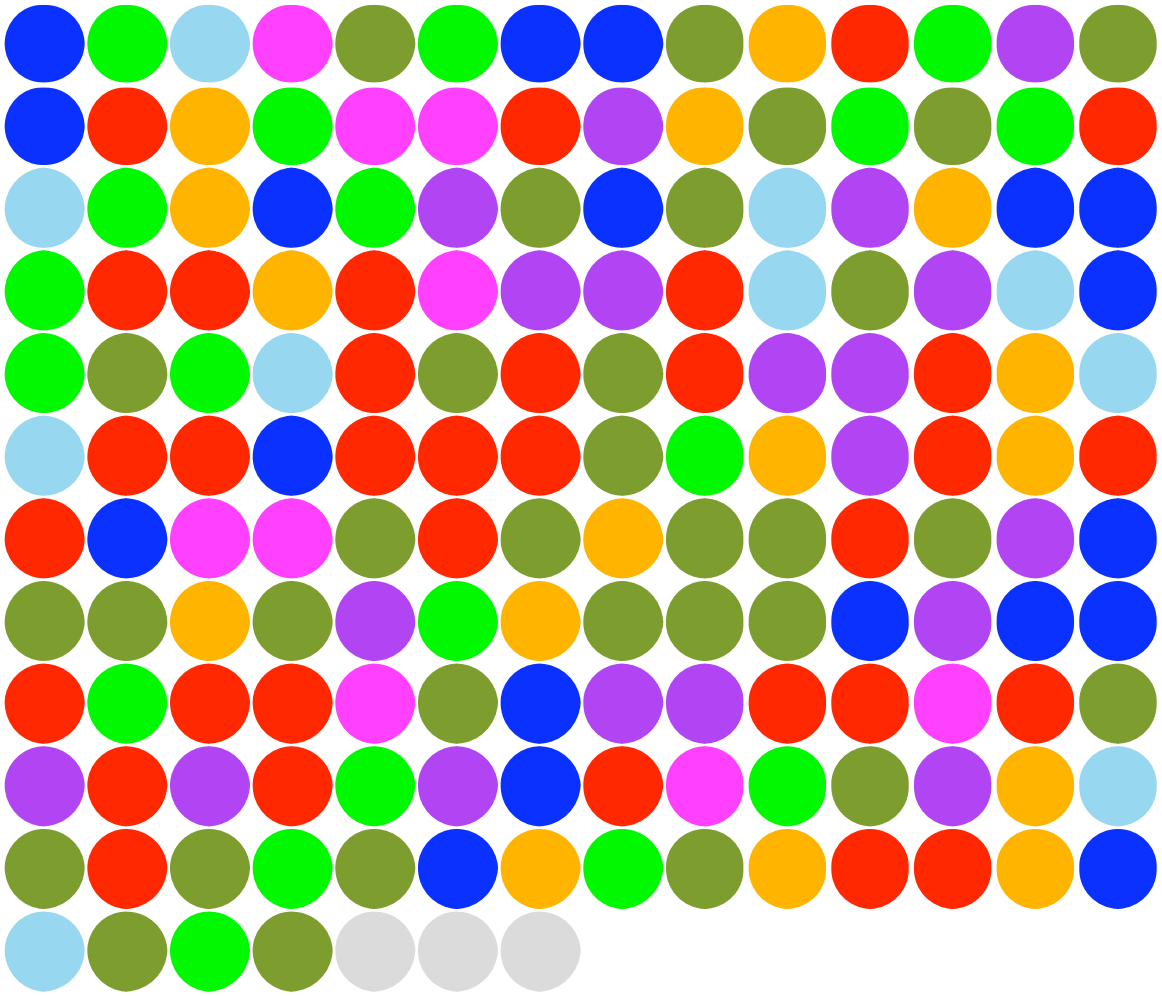
USA 33	UK 6	India 3	Greece 1
Germany 26	Denmark 5	New Zealand 3	Ireland 1
France 13	Poland 5	Sweden 3	Italy 1
Japan 10	Romania 4	Ukraine 2	Netherlands 1
Australia 7	Argentina 3	Bulgaria 1	Norway 1
Canada 6	Austria 3	Colombia 1	Portugal 1
Russia 6	Belgium 3		

We had submissions from all over the world.

We nearly had a submission from each occupied continent in the world, since a team from South Africa registered, but sadly they did not end up submitting anything to the tournament. Still, pretty good coverage, no?

The results

Initial submission



The dots here correspond to teams. Each team gets its own dot. The color of the dot indicates which programming language the team used. We will see a series of these slides where the dots will turn grey when the corresponding team drops out of the tournament, according to the tournament phases: the regular season and the playoffs.

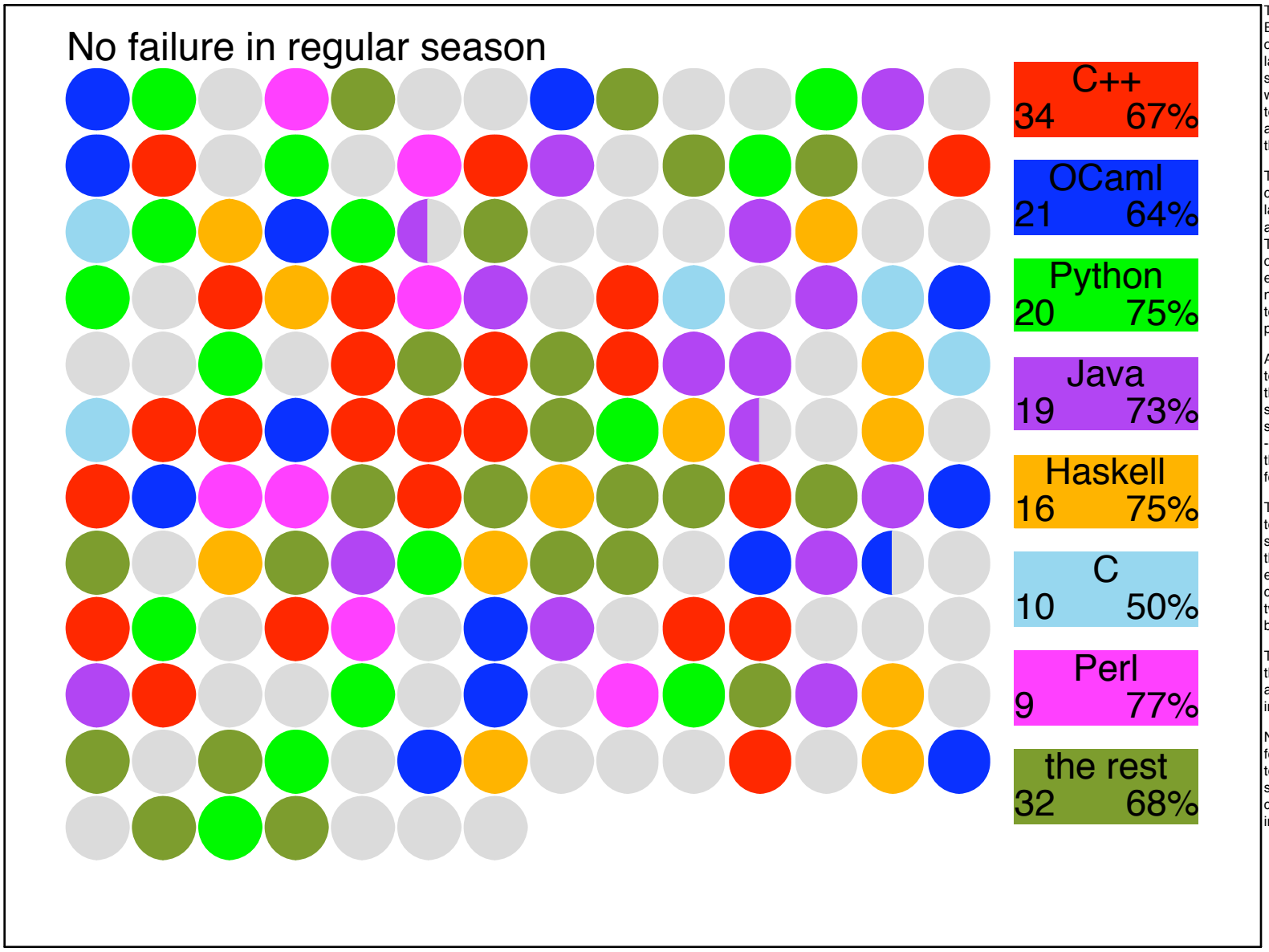
The right-hand column shows which colors match which teams. I cut off languages with 5 and a fewer entries and combined them into the last color. The numbers are the absolute counts of teams and will remain the same for each slide. the percentages show the number remaining at each stage of the tournament and will change during the progression.

Already on the first slide there are three teams that have dropped out here, at the bottom. those are teams that only submitted to the twist. I allowed this, since it didn't make sense to disallow it -- teams can (and did) submit entries that were bogus, just to reserve a slot for the second phase.

The second phase shows all of the teams that did not fail in the regular season. You see some half circles there -- teams were allowed to submit either one or two sets of players. Half a circle means that the team submitted two entries and one has dropped out by this point.

The third phase shows teams that beat the judges cops in the regular season and the fourth is teams that did not fail in the playoffs.

Next, we can see the same four slides for the twist: initial submission, those teams that did not fail in the regular season, those that beat the judges cops, and finally those that did not fail in the twist.



The dots here correspond to teams. Each team gets its own dot. The color of the dot indicates which programming language the team used. We will see a series of these slides where the dots will turn grey when the corresponding team drops out of the tournament, according to the tournament phases: the regular season and the playoffs.

The right-hand column shows which colors match which teams. I cut off languages with 5 and a fewer entries and combined them into the last color. The numbers are the absolute counts of teams and will remain the same for each slide. the percentages show the number remaining at each stage of the tournament and will change during the progression.

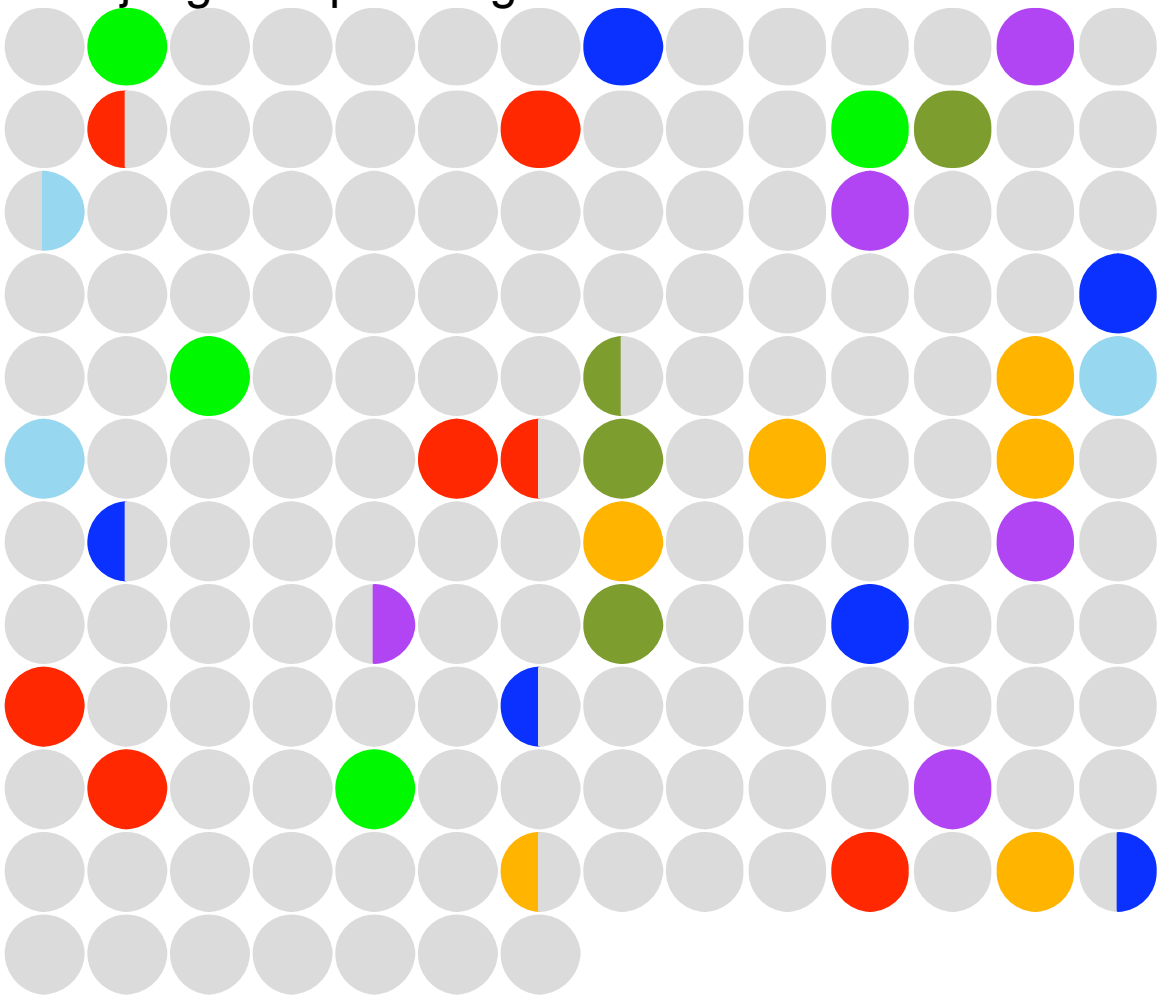
Already on the first slide there are three teams that have dropped out here, at the bottom. those are teams that only submitted to the twist. I allowed this, since it didn't make sense to disallow it -- teams can (and did) submit entries that were bogus, just to reserve a slot for the second phase.

The second phase shows all of the teams that did not fail in the regular season. You see some half circles there -- teams were allowed to submit either one or two sets of players. Half a circle means that the team submitted two entries and one has dropped out by this point.

The third phase shows teams that beat the judges cops in the regular season and the fourth is teams that did not fail in the playoffs.

Next, we can see the same four slides for the twist: initial submission, those teams that did not fail in the regular season, those that beat the judges cops, and finally those that did not fail in the twist.

Beat judges cops in regular season



The dots here correspond to teams. Each team gets its own dot. The color of the dot indicates which programming language the team used. We will see a series of these slides where the dots will turn grey when the corresponding team drops out of the tournament, according to the tournament phases: the regular season and the playoffs.

The right-hand column shows which colors match which teams. I cut off languages with 5 and a fewer entries and combined them into the last color. The numbers are the absolute counts of teams and will remain the same for each slide. the percentages show the number remaining at each stage of the tournament and will change during the progression.

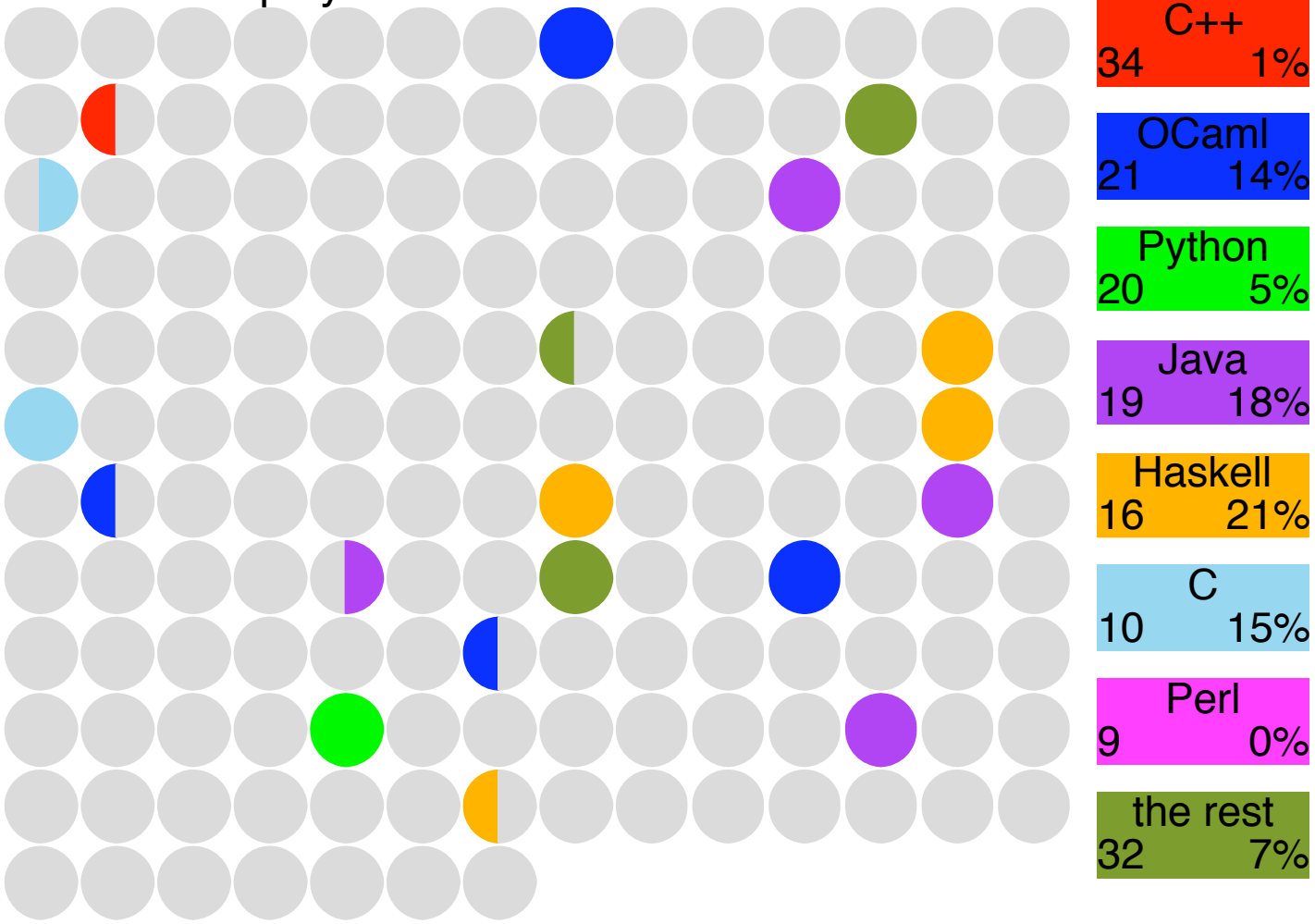
Already on the first slide there are three teams that have dropped out here, at the bottom. those are teams that only submitted to the twist. I allowed this, since it didn't make sense to disallow it -- teams can (and did) submit entries that were bogus, just to reserve a slot for the second phase.

The second phase shows all of the teams that did not fail in the regular season. You see some half circles there -- teams were allowed to submit either one or two sets of players. Half a circle means that the team submitted two entries and one has dropped out by this point.

The third phase shows teams that beat the judges cops in the regular season and the fourth is teams that did not fail in the playoffs.

Next, we can see the same four slides for the twist: initial submission, those teams that did not fail in the regular season, those that beat the judges cops, and finally those that did not fail in the twist.

No failure in playoffs



The dots here correspond to teams. Each team gets its own dot. The color of the dot indicates which programming language the team used. We will see a series of these slides where the dots will turn grey when the corresponding team drops out of the tournament, according to the tournament phases: the regular season and the playoffs.

The right-hand column shows which colors match which teams. I cut off languages with 5 and a fewer entries and combined them into the last color. The numbers are the absolute counts of teams and will remain the same for each slide. the percentages show the number remaining at each stage of the tournament and will change during the progression.

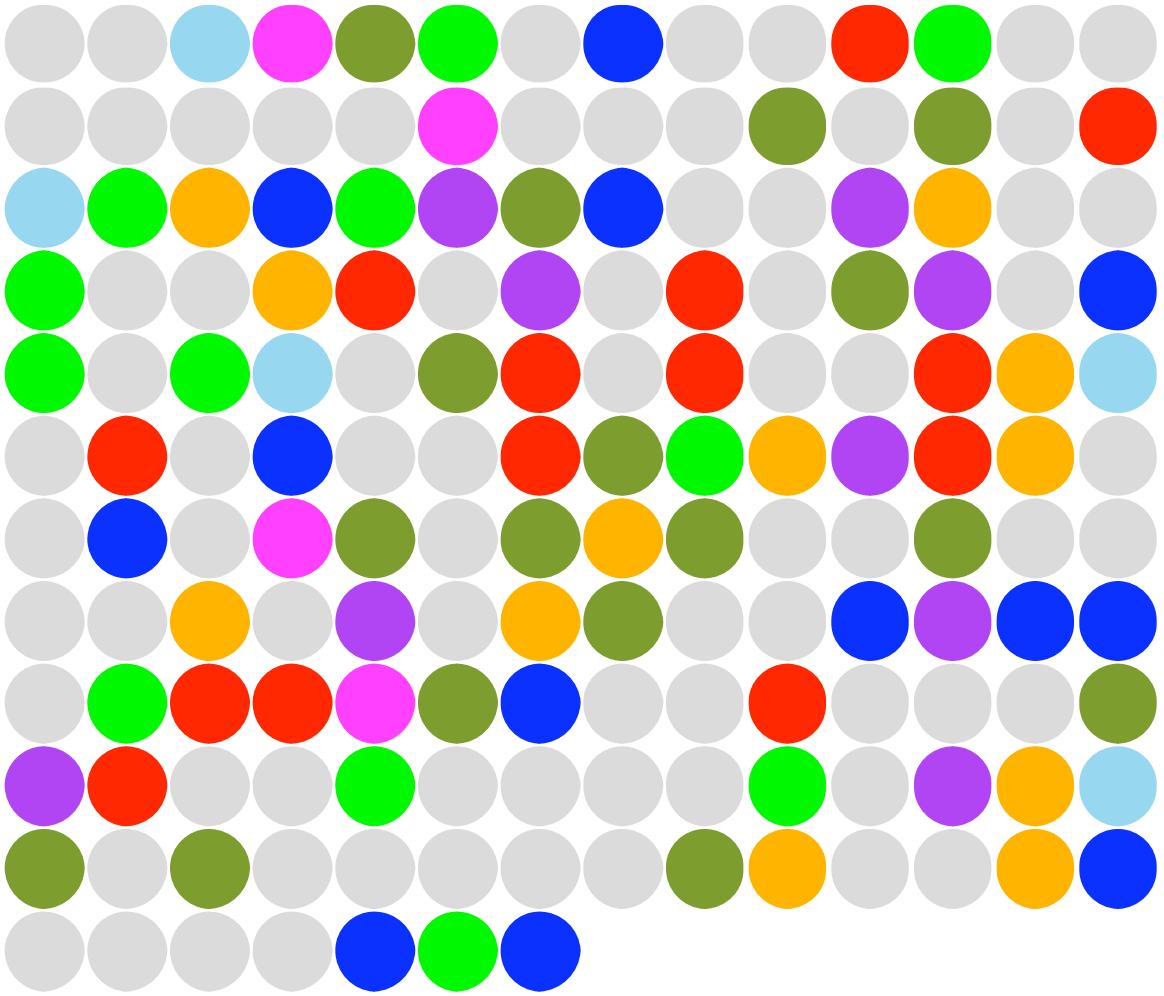
Already on the first slide there are three teams that have dropped out here, at the bottom. those are teams that only submitted to the twist. I allowed this, since it didn't make sense to disallow it -- teams can (and did) submit entries that were bogus, just to reserve a slot for the second phase.

The second phase shows all of the teams that did not fail in the regular season. You see some half circles there -- teams were allowed to submit either one or two sets of players. Half a circle means that the team submitted two entries and one has dropped out by this point.

The third phase shows teams that beat the judges cops in the regular season and the fourth is teams that did not fail in the playoffs.

Next, we can see the same four slides for the twist: initial submission, those teams that did not fail in the regular season, those that beat the judges cops, and finally those that did not fail in the twist.

Twist: initial submission



The dots here correspond to teams. Each team gets its own dot. The color of the dot indicates which programming language the team used. We will see a series of these slides where the dots will turn grey when the corresponding team drops out of the tournament, according to the tournament phases: the regular season and the playoffs.

The right-hand column shows which colors match which teams. I cut off languages with 5 and a fewer entries and combined them into the last color. The numbers are the absolute counts of teams and will remain the same for each slide. the percentages show the number remaining at each stage of the tournament and will change during the progression.

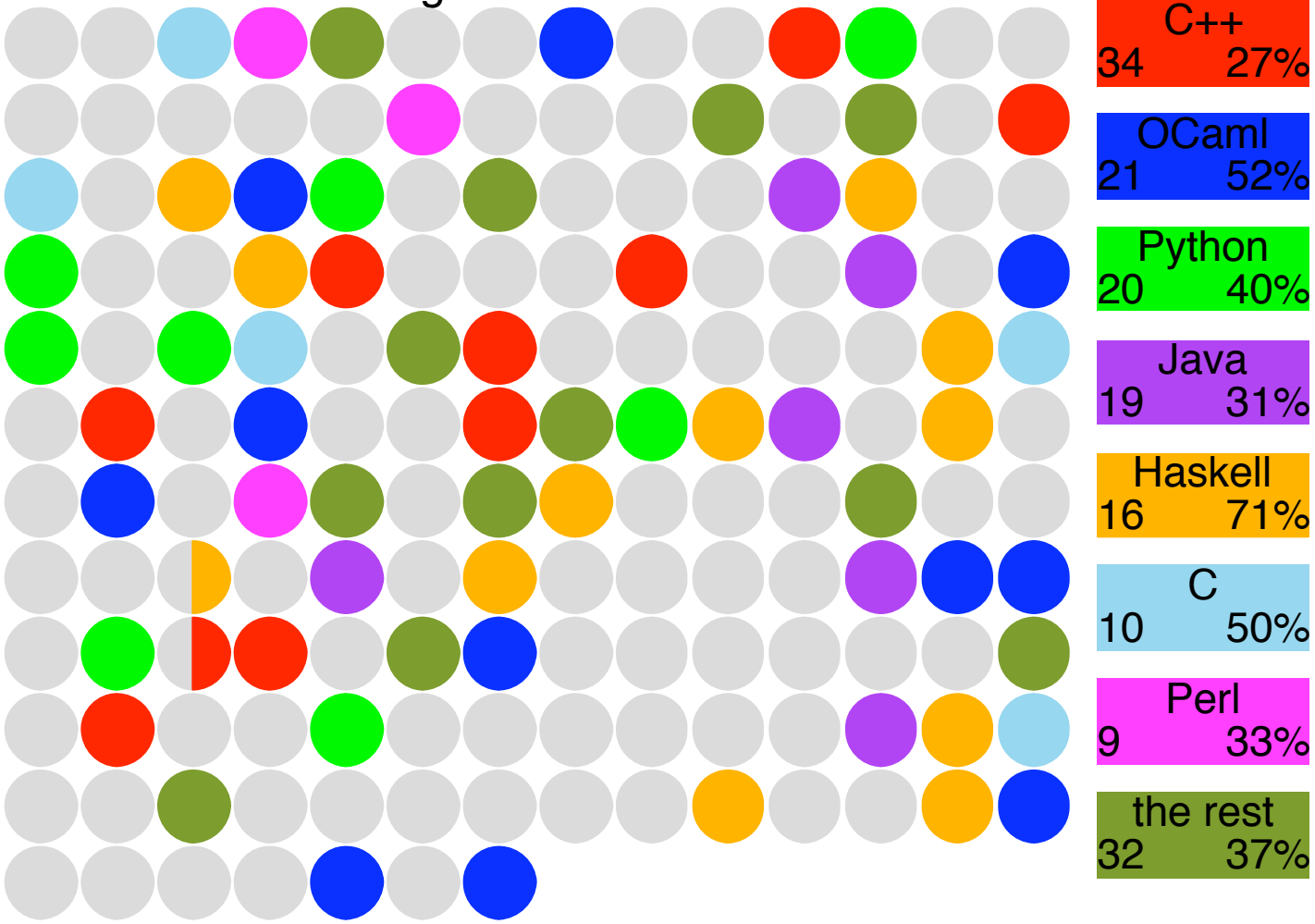
Already on the first slide there are three teams that have dropped out here, at the bottom. those are teams that only submitted to the twist. I allowed this, since it didn't make sense to disallow it -- teams can (and did) submit entries that were bogus, just to reserve a slot for the second phase.

The second phase shows all of the teams that did not fail in the regular season. You see some half circles there -- teams were allowed to submit either one or two sets of players. Half a circle means that the team submitted two entries and one has dropped out by this point.

The third phase shows teams that beat the judges cops in the regular season and the fourth is teams that did not fail in the playoffs.

Next, we can see the same four slides for the twist: initial submission, those teams that did not fail in the regular season, those that beat the judges cops, and finally those that did not fail in the twist.

Twist: no failure in regular season



The dots here correspond to teams. Each team gets its own dot. The color of the dot indicates which programming language the team used. We will see a series of these slides where the dots will turn grey when the corresponding team drops out of the tournament, according to the tournament phases: the regular season and the playoffs.

The right-hand column shows which colors match which teams. I cut off languages with 5 and a fewer entries and combined them into the last color. The numbers are the absolute counts of teams and will remain the same for each slide. the percentages show the number remaining at each stage of the tournament and will change during the progression.

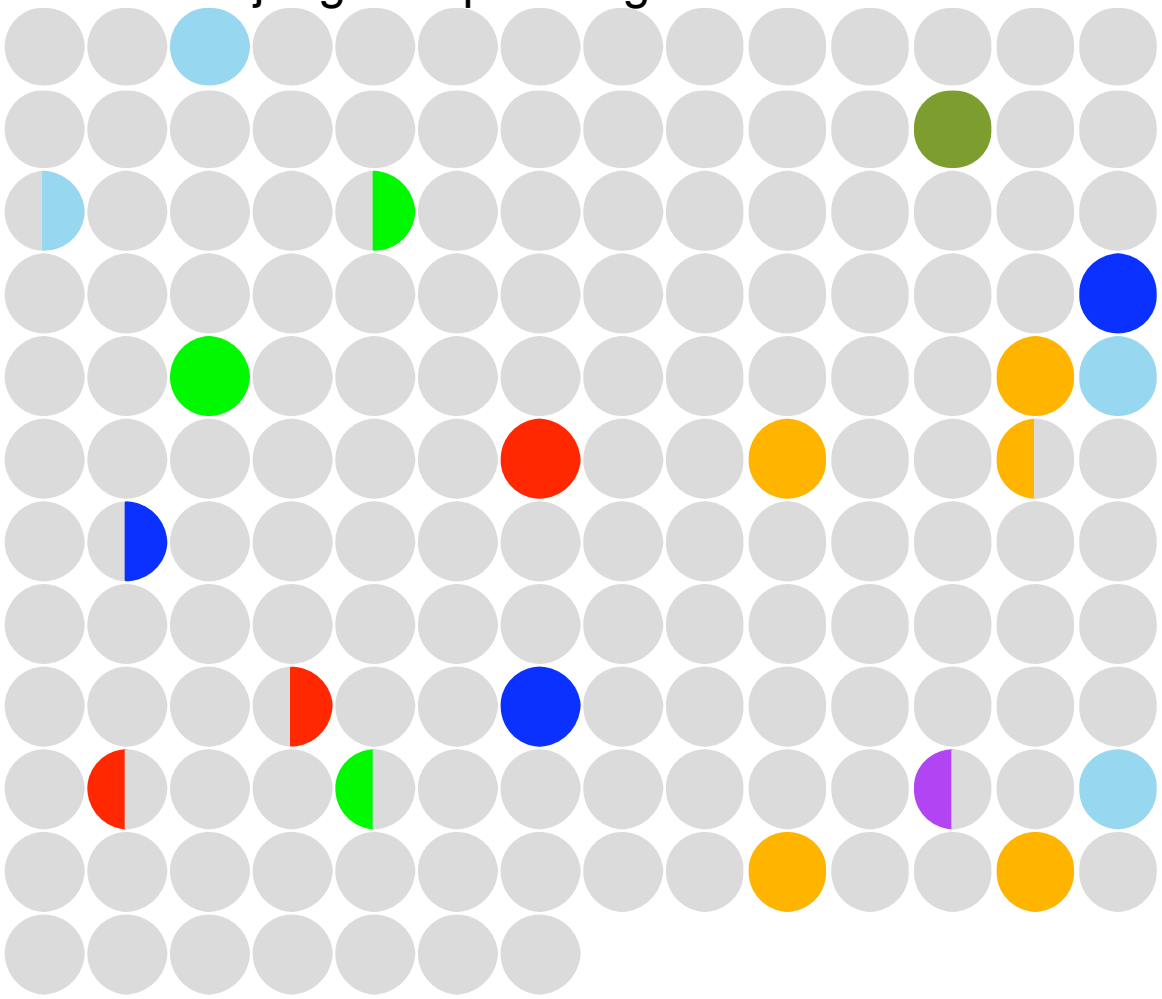
Already on the first slide there are three teams that have dropped out here, at the bottom. those are teams that only submitted to the twist. I allowed this, since it didn't make sense to disallow it -- teams can (and did) submit entries that were bogus, just to reserve a slot for the second phase.

The second phase shows all of the teams that did not fail in the regular season. You see some half circles there -- teams were allowed to submit either one or two sets of players. Half a circle means that the team submitted two entries and one has dropped out by this point.

The third phase shows teams that beat the judges cops in the regular season and the fourth is teams that did not fail in the playoffs.

Next, we can see the same four slides for the twist: initial submission, those teams that did not fail in the regular season, those that beat the judges cops, and finally those that did not fail in the twist.

Twist: beat judges cops in regular season



The dots here correspond to teams. Each team gets its own dot. The color of the dot indicates which programming language the team used. We will see a series of these slides where the dots will turn grey when the corresponding team drops out of the tournament, according to the tournament phases: the regular season and the playoffs.

The right-hand column shows which colors match which teams. I cut off languages with 5 and a fewer entries and combined them into the last color. The numbers are the absolute counts of teams and will remain the same for each slide. the percentages show the number remaining at each stage of the tournament and will change during the progression.

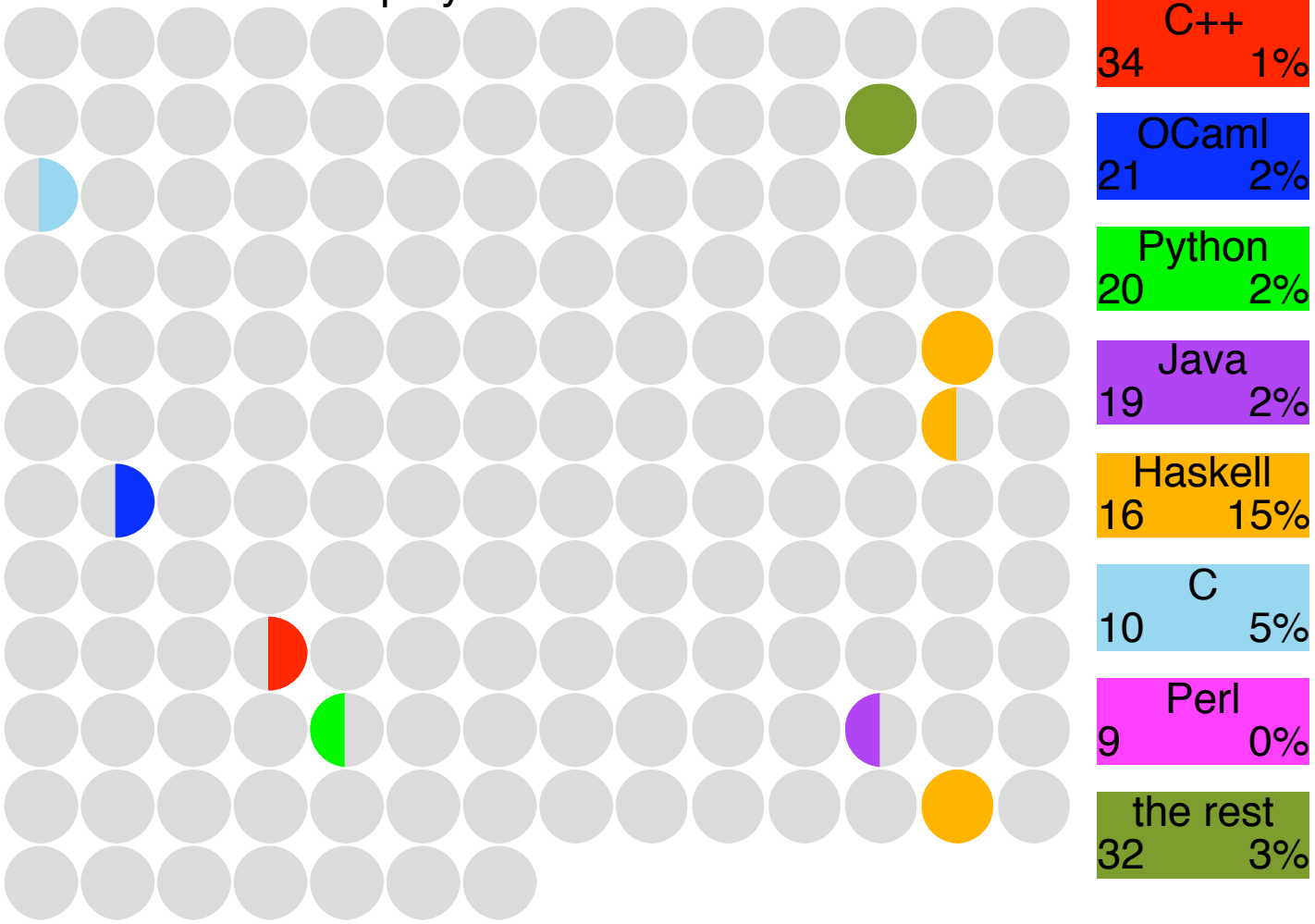
Already on the first slide there are three teams that have dropped out here, at the bottom. those are teams that only submitted to the twist. I allowed this, since it didn't make sense to disallow it -- teams can (and did) submit entries that were bogus, just to reserve a slot for the second phase.

The second phase shows all of the teams that did not fail in the regular season. You see some half circles there -- teams were allowed to submit either one or two sets of players. Half a circle means that the team submitted two entries and one has dropped out by this point.

The third phase shows teams that beat the judges cops in the regular season and the fourth is teams that did not fail in the playoffs.

Next, we can see the same four slides for the twist: initial submission, those teams that did not fail in the regular season, those that beat the judges cops, and finally those that did not fail in the twist.

Twist: no failure in playoffs



The dots here correspond to teams. Each team gets its own dot. The color of the dot indicates which programming language the team used. We will see a series of these slides where the dots will turn grey when the corresponding team drops out of the tournament, according to the tournament phases: the regular season and the playoffs.

The right-hand column shows which colors match which teams. I cut off languages with 5 and a fewer entries and combined them into the last color. The numbers are the absolute counts of teams and will remain the same for each slide. the percentages show the number remaining at each stage of the tournament and will change during the progression.

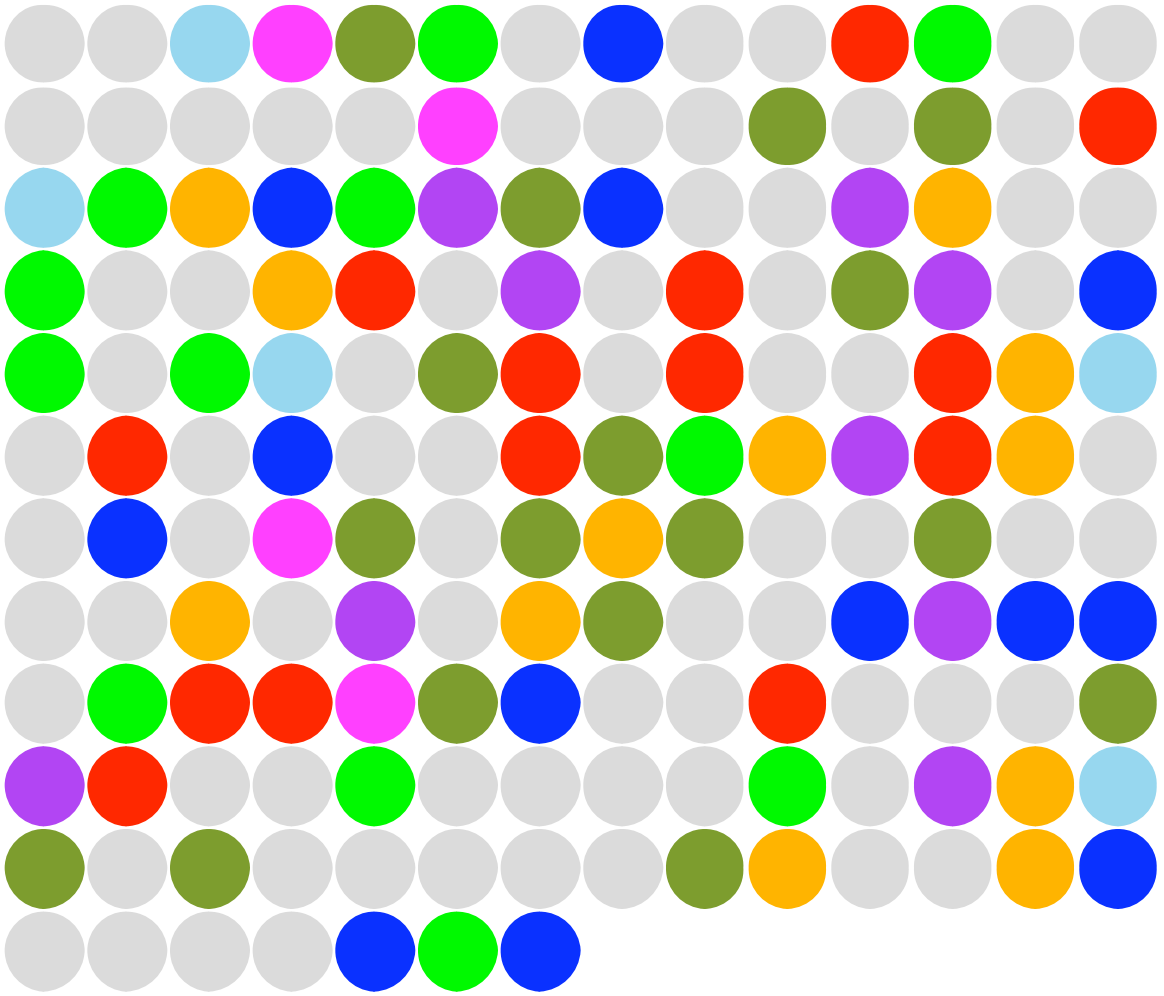
Already on the first slide there are three teams that have dropped out here, at the bottom. those are teams that only submitted to the twist. I allowed this, since it didn't make sense to disallow it -- teams can (and did) submit entries that were bogus, just to reserve a slot for the second phase.

The second phase shows all of the teams that did not fail in the regular season. You see some half circles there -- teams were allowed to submit either one or two sets of players. Half a circle means that the team submitted two entries and one has dropped out by this point.

The third phase shows teams that beat the judges cops in the regular season and the fourth is teams that did not fail in the playoffs.

Next, we can see the same four slides for the twist: initial submission, those teams that did not fail in the regular season, those that beat the judges cops, and finally those that did not fail in the twist.

Twist: initial submission



C++	34	41%
OCaml	21	61%
Python	20	60%
Java	19	47%
Haskell	16	75%
C	10	50%
Perl	9	44%
the rest	32	53%

The main trend you see here is that there is quickly lots of grey and, perhaps as we expect, many teams submitted buggy code.

The slide I think really stands out is the twist initial submission (repeated here). There were nearly half of the teams that did not even submit something to the twist. I had feared this would happen, but teams knew that the twist is where the contest would be decided and, as we have seen, assuming you have working pre-twist entries it is not difficult to get working twist entries.

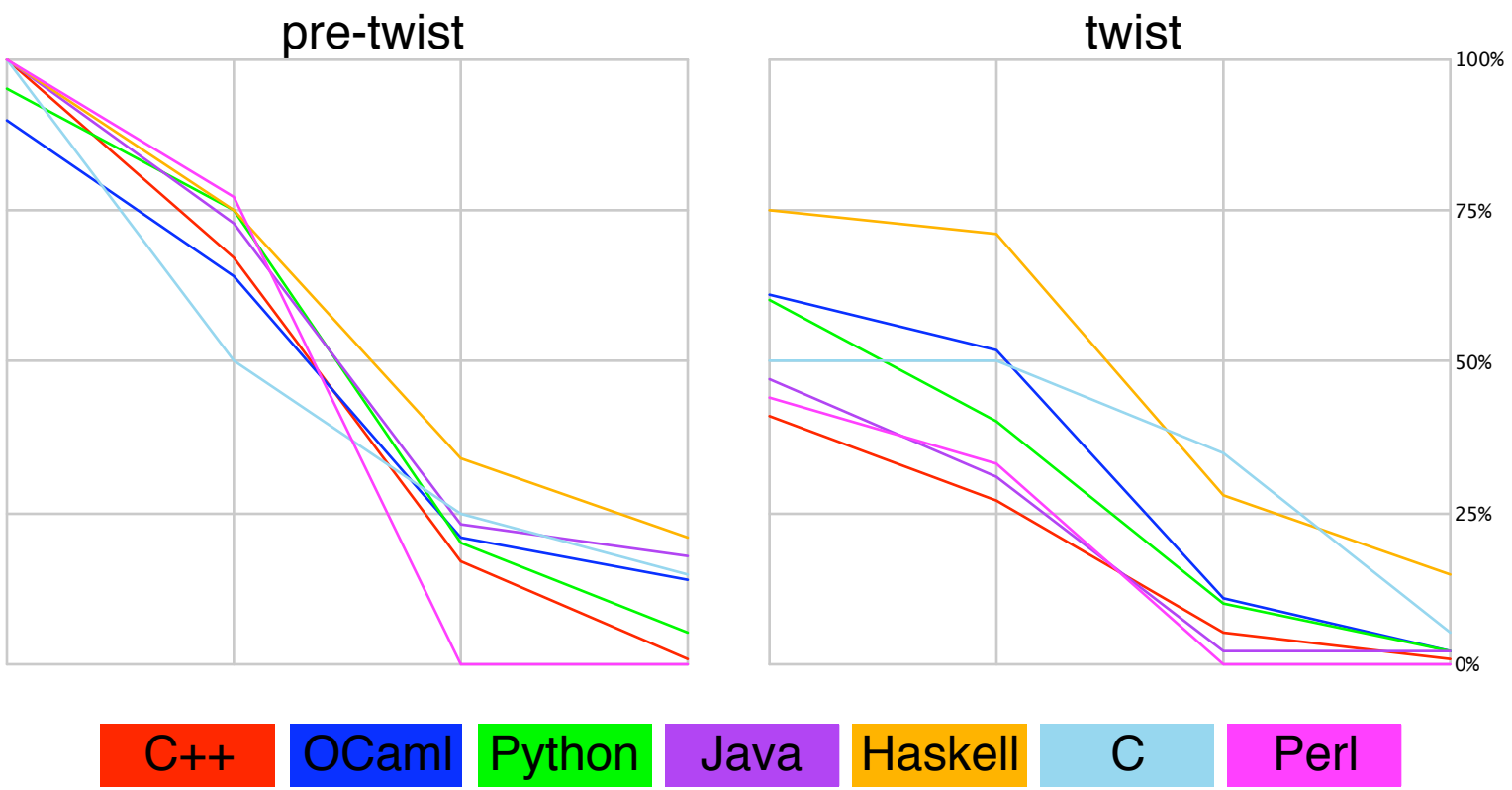
This slide should really speak to us as PL researchers and educators. I think it is fair to assume that most of the people who did not submit a solution *wanted* to submit something. And yet, they were unable.

I would go further and say that the problem here is the 3-day hack fest mentality that permeates programming culture. I certainly see this in the students I teach, and I certainly imagine, based on what I see in the software I use that this is not unique to just my students :).

But the truth is that we, the PL research community, have an opportunity and, dare I say, an obligation to help with this problem. We know something about how to structure programs to get good re-use and, in particular, to help make the structure of programs match the structure of the problems they solve.

We should be teaching that to our students (early!), we should be develping that knowledge and we are to some extent, but we are failing to bring this knowledge out to the world effectively.

Performance in stages



As I studied these numbers, there were two other things I saw, and I think they show up particularly well here. These graph summarize the previous slides. From left to right, we see the initial submissions, then failures, losing, and more failures. and top to bottom are percentages of remaining teams, colored by programming languages.

The clear fact that stands out here is that Haskell is the language of choice for the programming contest. Haskell stands out a little bit in the first round, but it clearly stands out in the twist. So, kudos to the Haskell community for both producing a language that lets people build re-usable code and instilling this as a value in their community!

One other interesting fact here -- look at how C behaves in the twist (you can also see that a little bit in the pre-twist). Its curve is not really the same as the other keys. It stays higher longer and then suddenly drops. What this means is that there are more failures late. Although there were only 5 C teams in the twist so it it seems hard to generalize, but this matches my experience with C -- bugs are generally harder to find and so come out late.

Desiderata, i (one last time)

Twist playoffs

- 13 entries participating (3 disqualified late)
- 349 randomly selected pods
- Average pod place determines overall winner

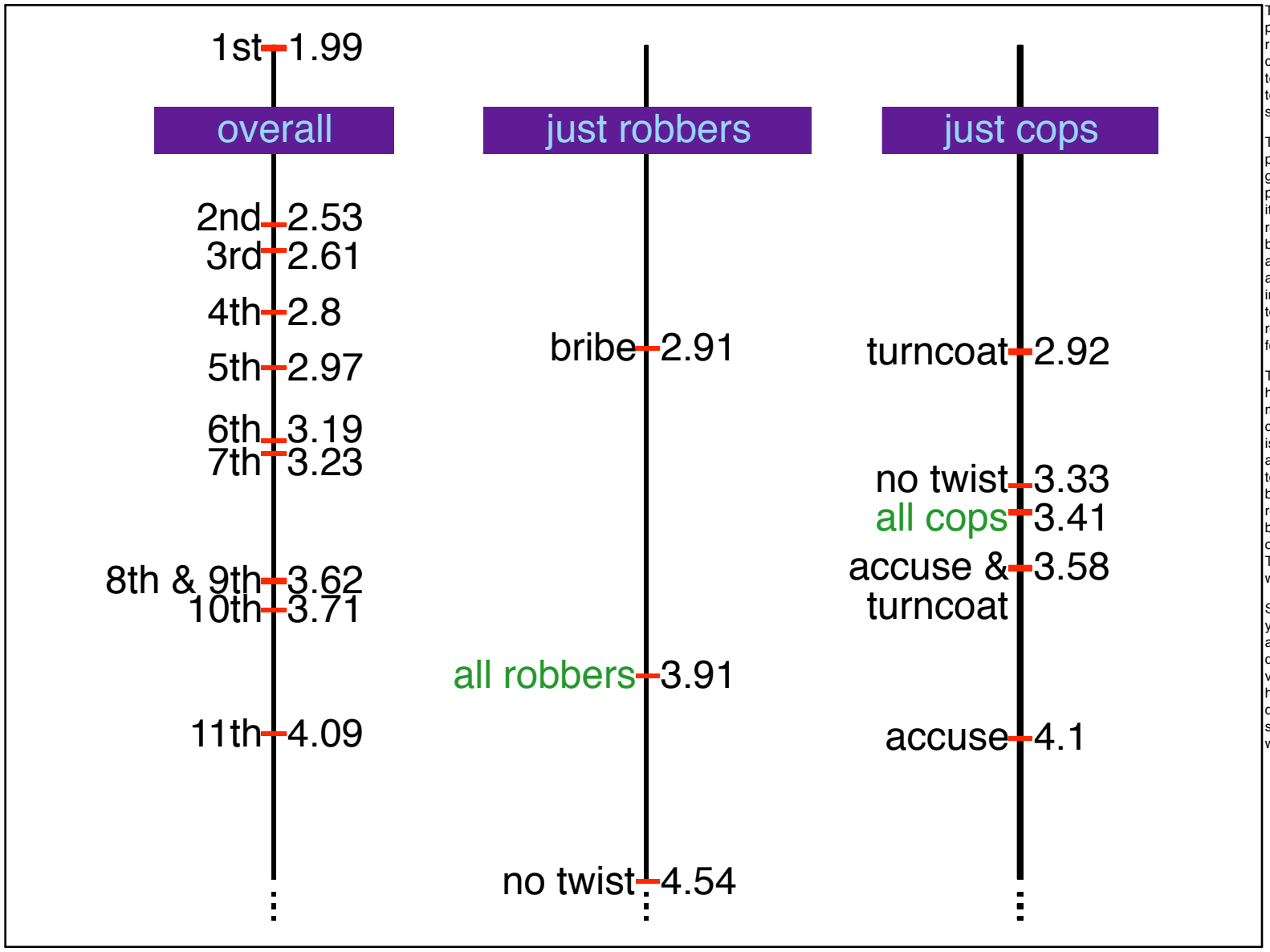
Of the 12564 times a player played,

- 2640 tried to use the twist,
- 9924 didn't

Lets return to the first desiderata, in particular the hope that taking advantage of the twist actually had an impact on performance in the second round.

By the time we got to the final playoffs, there were only 13 entries left. We ran 350 pods, randomly selected, but lost one logfile. And recall, the winner was decided by the average place in the pod.

Overall, there were few people who used the twist. And by using the twist I don't mean a clever use at all -- I merely scanned the output logs for the robber message that indicates it is looking for dirty cops and for the cop message that indicates the cop is willing to become dirty or make an accustion.



These scales show the average pod placements for teams for the teams, running from top to bottom. The left column shows the placements for the teams from one thru 11 (the last two teams are much lower than those shown here).

The second column shows the average pod placements for different robbers. In green, you see the average pod placement for all of the robbers. Below it, you see the average place for robbers that did not exhibit twist behavior, ie, did not attempt to bribe any cops. Above it, you see the average place for robbers that did implement the twist. Clearly, attempting to bribe cops was a big win for the robber, so our desiderata was achieved for robbers.

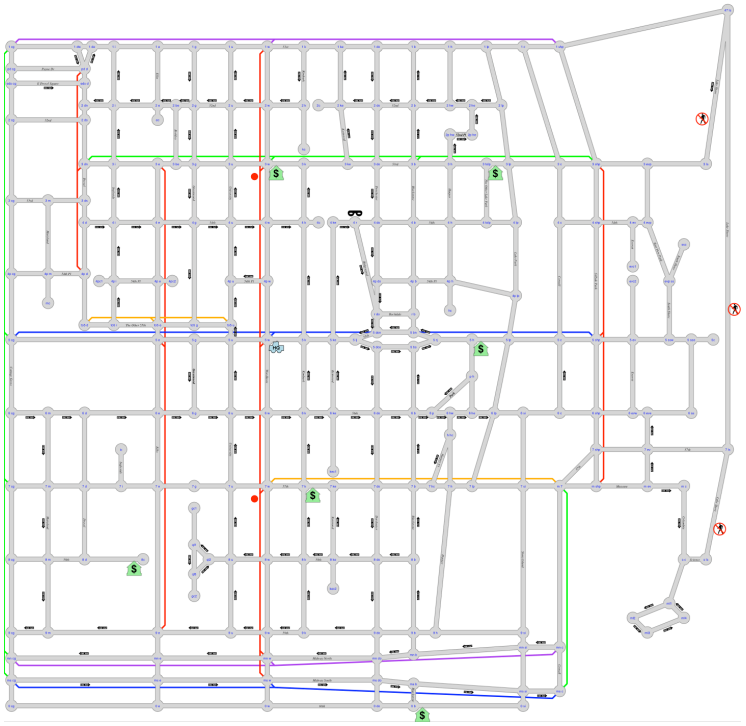
There is more interesting things happening here in the cops. As we might have guessed, becoming a dirty cop is a good idea, on average. What is surprising is that making accusations appears to be a bad idea. But, this has to do with the way the judge's bots behaved in the regular season. In the regular season, two of the judge cops became dirty, one right in the beginning of the game and one in the middle. These dirty cops then immediately went and hid in a culdesac.

So, in order to pass the regular season, you had to either make accusations and gain control of those cops, become dirty, or be able to catch the robber very quickly. In the tournament, however, it was much harder to detect dirty cops, so players that had that strategy did not fare as well as those with the other two strategies.

The winners

Board game

How good a liar are you? How far can you trust your buddies?



After the contest finished, I was sitting around chatting with some of the organizers and we decided that the game really works better as a board game than a computer game. All of the coordination works much better and trying to figure who is lying can really be a lot of fun.

So, Matthew Flatt designed a full size board and Hsing-Huei Huang actually built them and made cards with pictures of the organizers actual noses in order to have private smell information while playing and we've put it all together into a box along with a people-friendly set of rules as extra prizes for the winners that made it to Estonia.

Without further ado ...

Judges' Prize

The judges' prize goes to Dylan Hackers

The Dylan Hackers are an extremely cool bunch of re-hackers.

Andreas Bogk and Hannes Mehnert are here to accept the award on behalf of Dylan Hackers

Third Prize

The third prize goes to Combat-Tanteidan

Haskell is not too shabby.

Sadly, Takayuki Muranushi and Hideyuki Tanaka could not make it

Second Prize

The second prize goes to Dylan Hackers

Dylan is a fine programming tool for many applications.

Andreas Bogk and Hannes Mehnert are here to accept the prize on behalf of Dylan Hackers

First Prize

The first prize goes to KiebererAndXiaoTou

Haskell is also the programming tool of choice for discriminating hackers.

Wolfgang Thaller is here to accept the prize
on behalf of the KiebererAndXiaoTou

Thanks

The contestants

Eli Barzilay, Matthias Blume, Jay McCarthy, Maurice Codik, Matthias Felleisen, Matthew Flatt, Jacob Matthews, Scott Owens, David Press, Mike Rainey, John Reppy, John Riehl, Jono Spiro, Dave Tucker, Adam Wick, and 黃馨慧