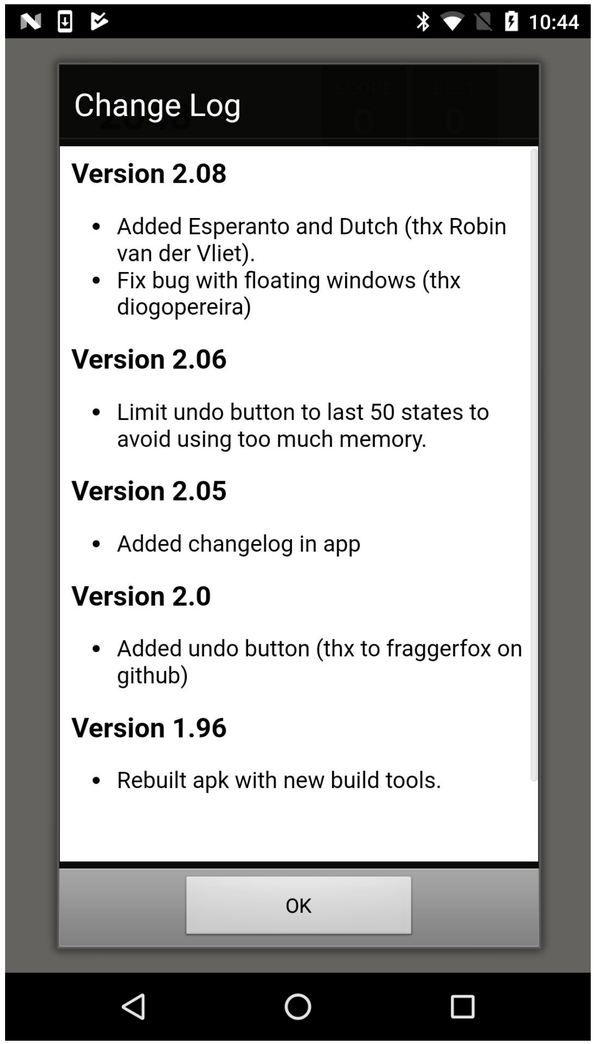
CISPA
HELMHOLTZ-ZENTRUM i. G.

Learning UI Element Interactions

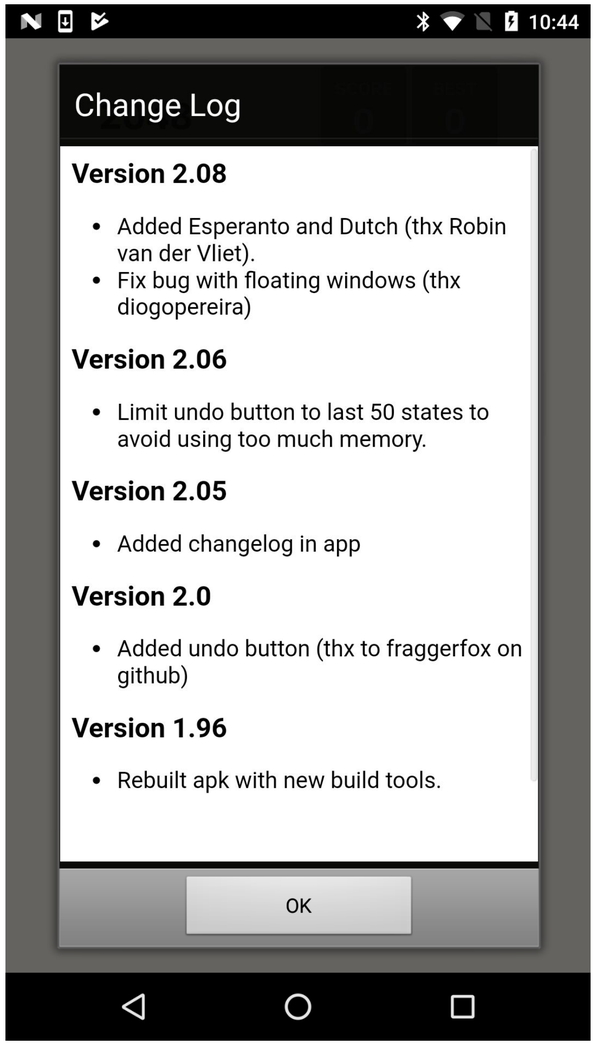
Christian Degott
Nataniel Borges Jr.
Andreas Zeller

UNIVERSITÄT DES SAARLANDES

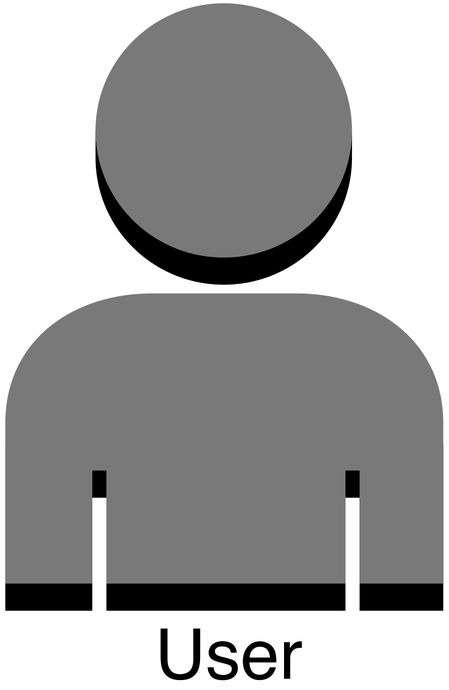
Introduction

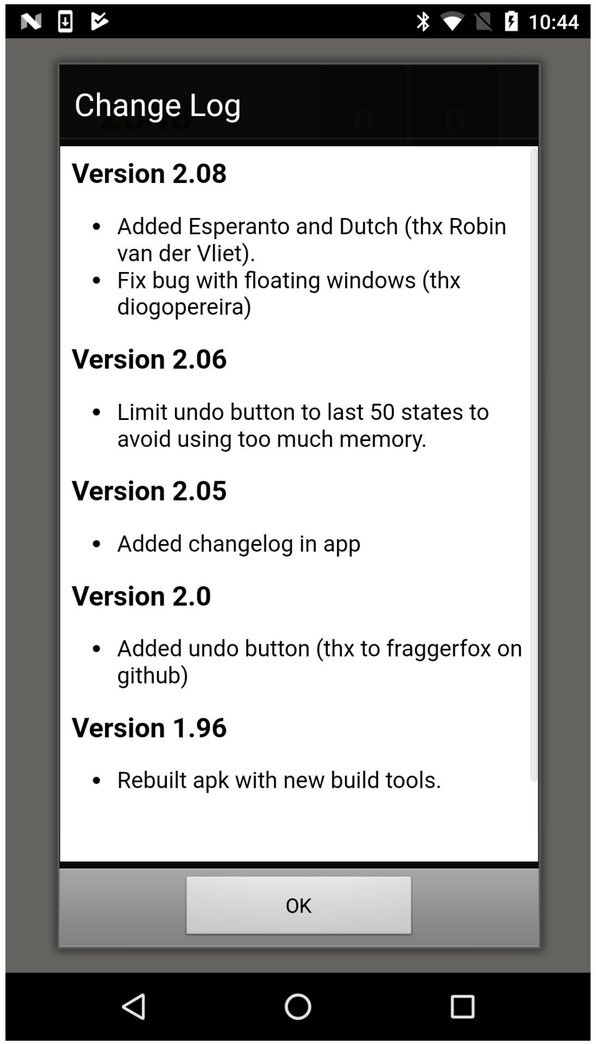


Introduction

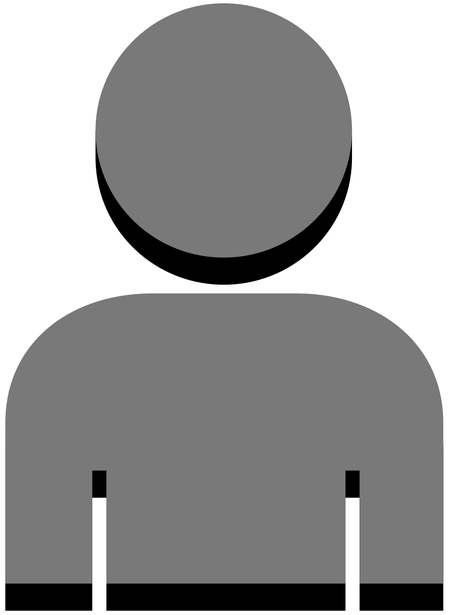


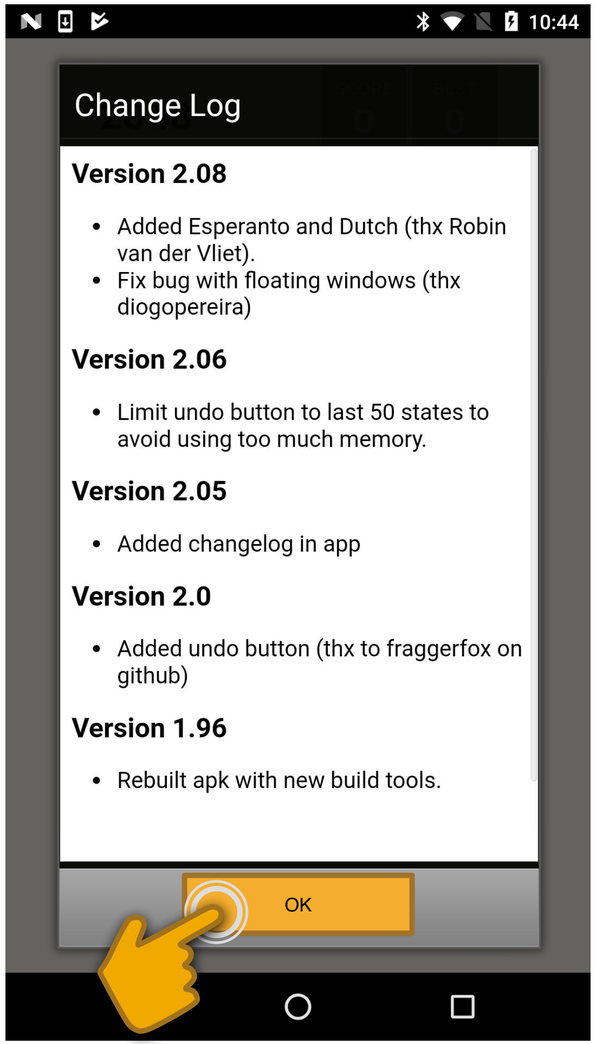
User vs GUI


User


10:44
Change Log
Version 2.08
Added Esperanto and Dutch (thx Robin van der Vliet).
Fix bug with floating windows (thx diogopereira)
Version 2.06
Limit undo button to last 50 states to avoid using too much memory.
Version 2.05
Added changelog in app
Version 2.0
Added undo button (thx to fraggerfox on github)
Version 1.96
Rebuilt apk with new build tools.
OK

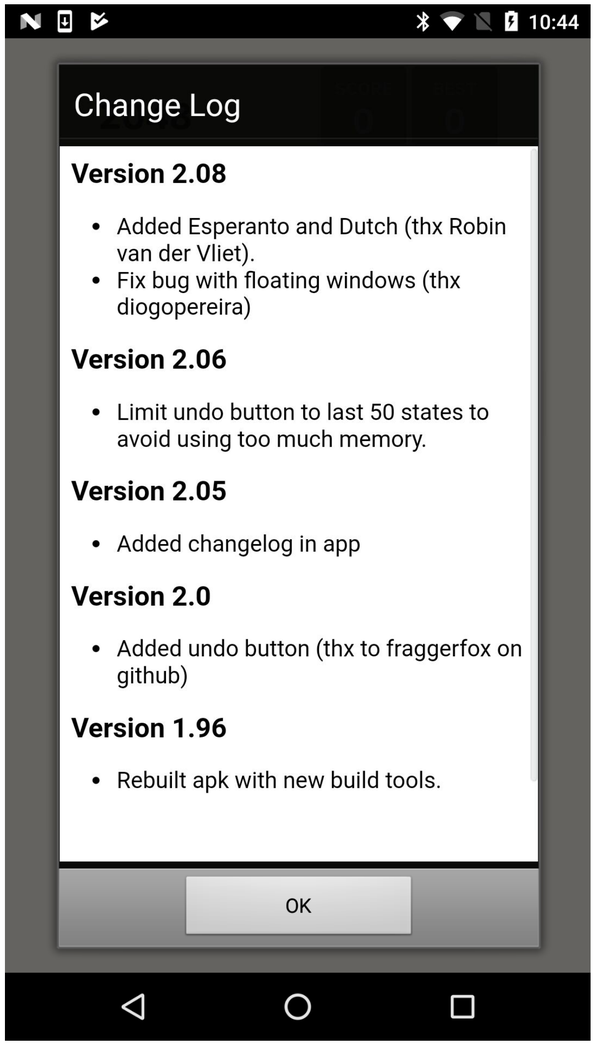
User vs GUI


User


10:44
Change Log
Version 2.08
Added Esperanto and Dutch (thx Robin van der Vliet).
Fix bug with floating windows (thx diogopereira)
Version 2.06
Limit undo button to last 50 states to avoid using too much memory.
Version 2.05
Added changelog in app
Version 2.0
Added undo button (thx to fraggerfox on github)
Version 1.96
Rebuilt apk with new build tools.
OK

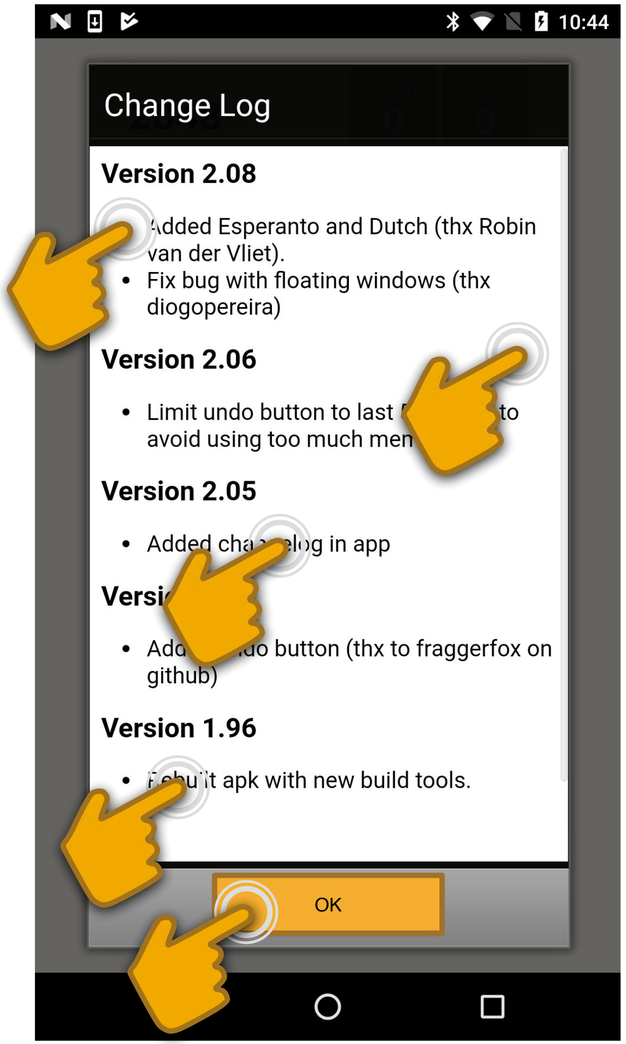
Test Generator vs GUI

Test Generator

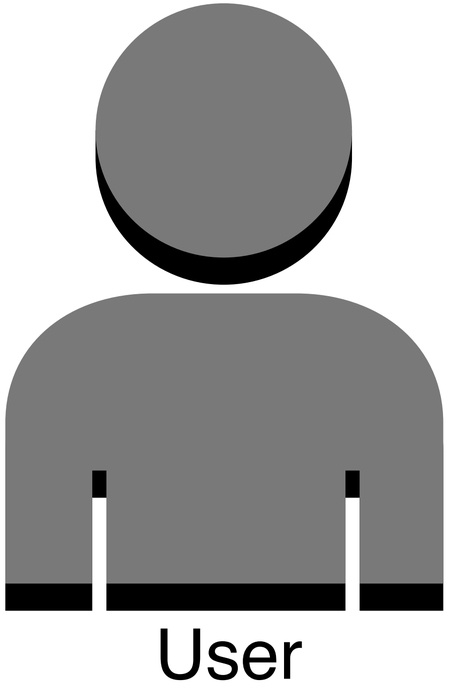


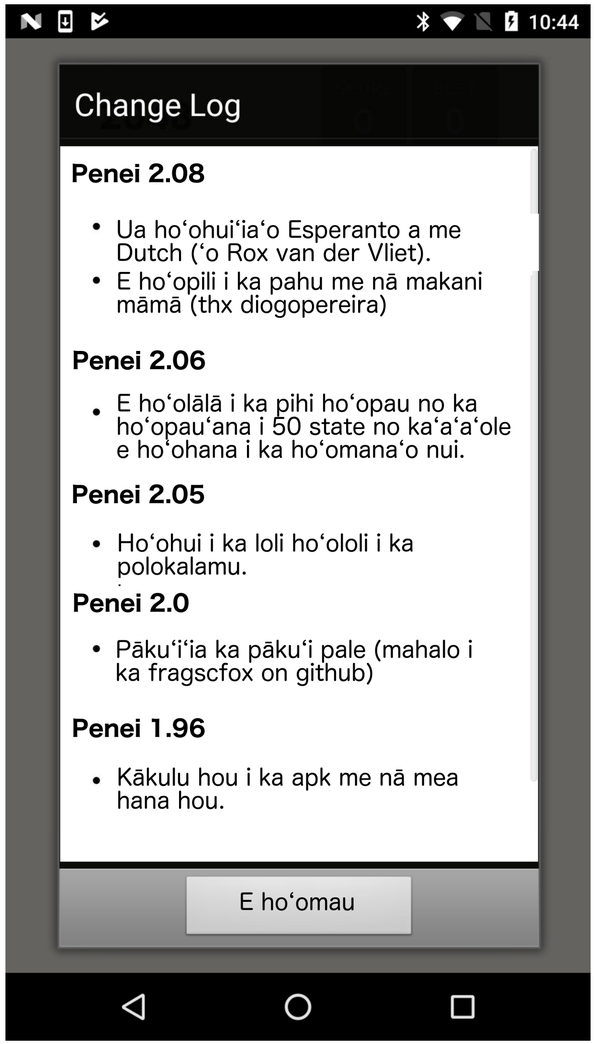
Test Generator vs GUI

Test Generator

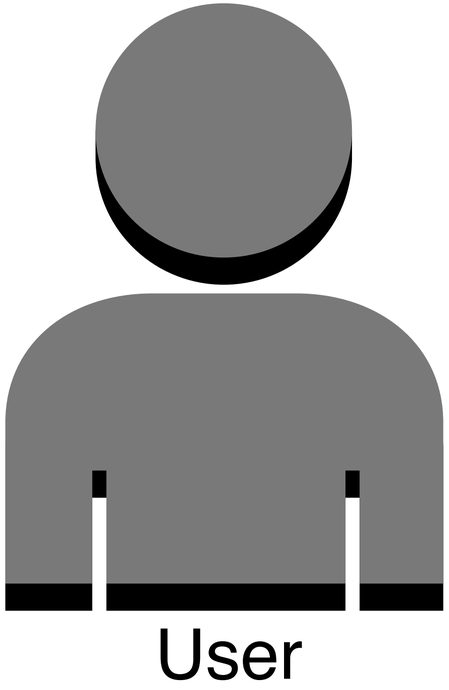


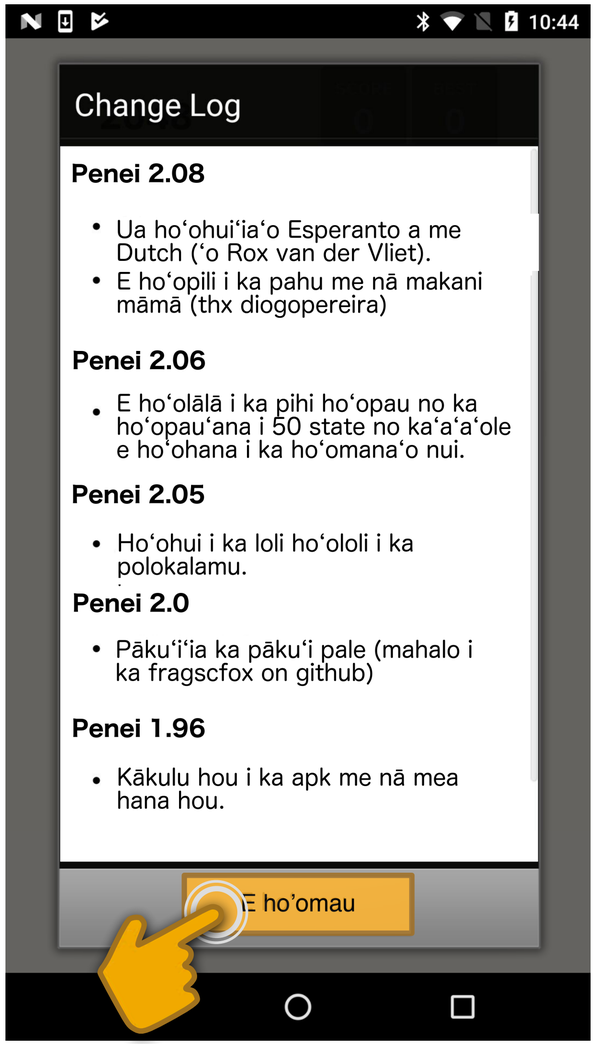
User vs (Unreadable) GUI





User vs (Unreadable) GUI





Motivation — User vs GUI vs Test Generator

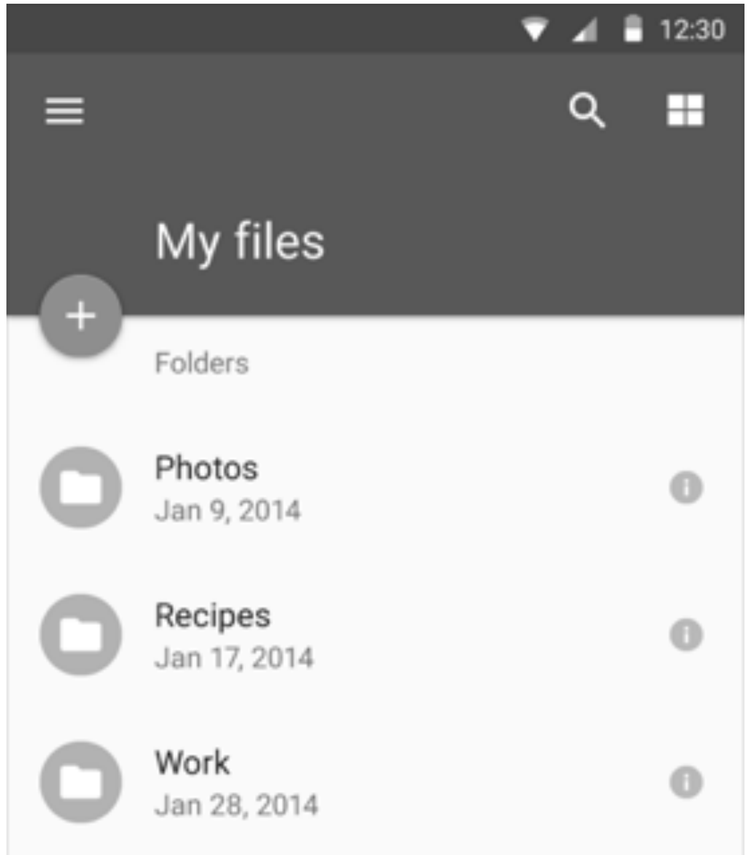
Apps follow **interaction patterns**

Motivation — User vs GUI vs Test Generator

Apps follow **interaction patterns**

Simple | Context-aware | Imperceptible

- *Buttons can be clicked*
- *Hyperlinks can be clicked*

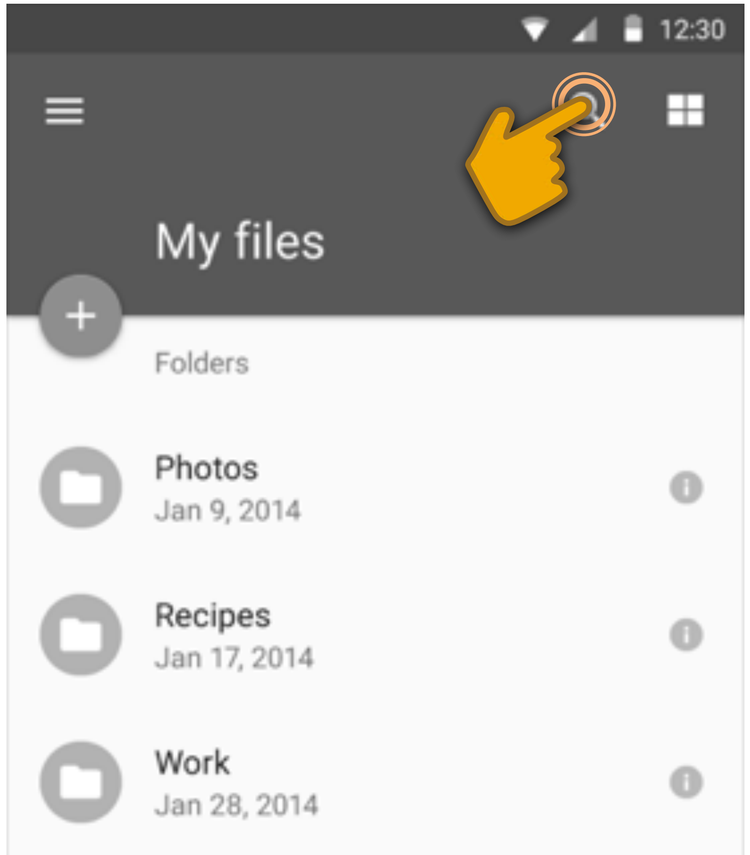


Motivation — User vs GUI vs Test Generator

Apps follow **interaction patterns**

Simple | Context-aware | Imperceptible

- *Buttons can be clicked*
- *Hyperlinks can be clicked*

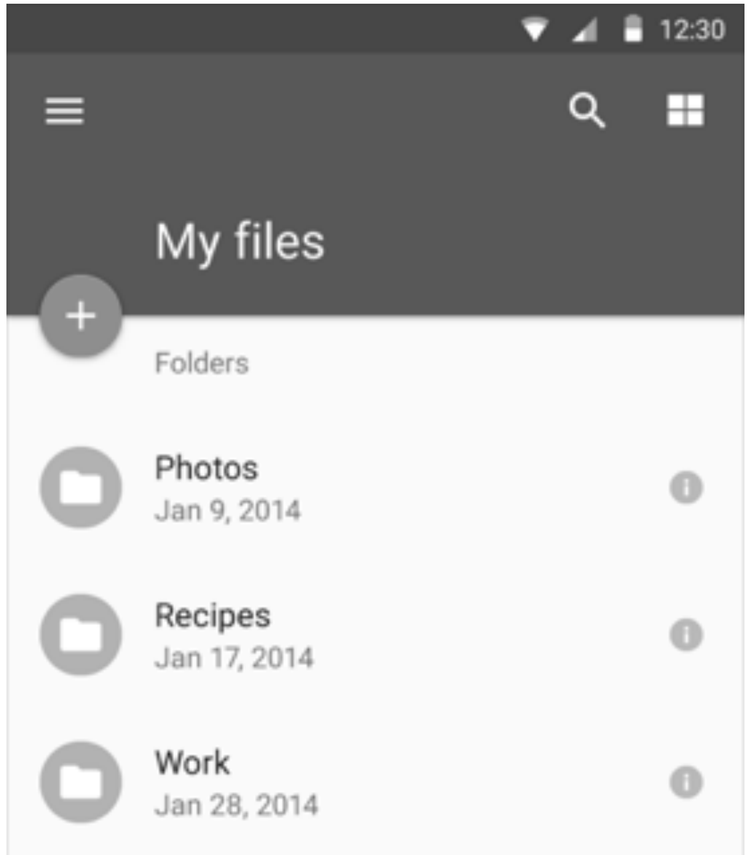


Motivation — User vs GUI vs Test Generator

Apps follow **interaction patterns**

Simple	**Context-aware**	Imperceptible

- *Labels within a layout inside a list view can be clicked*

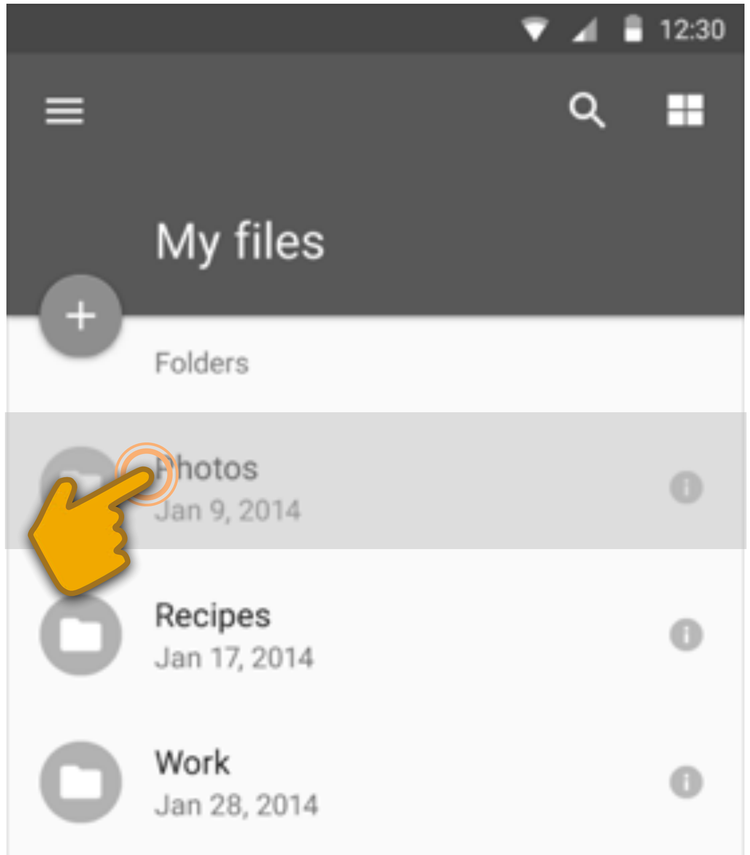


Motivation — User vs GUI vs Test Generator

Apps follow **interaction patterns**

Simple | **Context-aware** | Imperceptible

- *Labels within a layout inside a list view can be clicked*

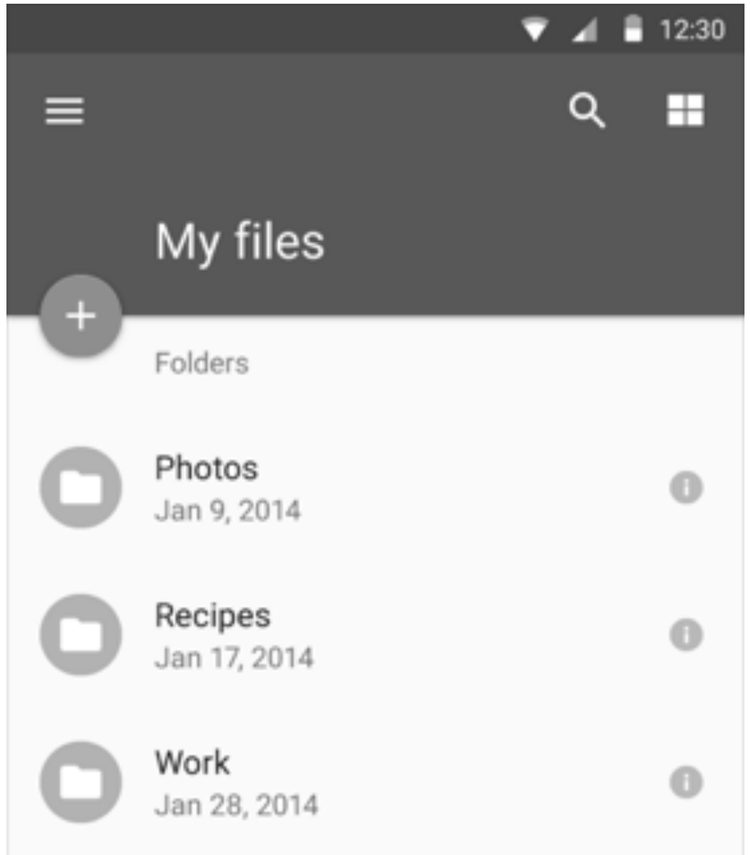


Motivation — User vs GUI vs Test Generator

Apps follow **interaction patterns**

Simple	Context-aware	**Imperceptible**

- *Layouts containing an image button can be clicked*

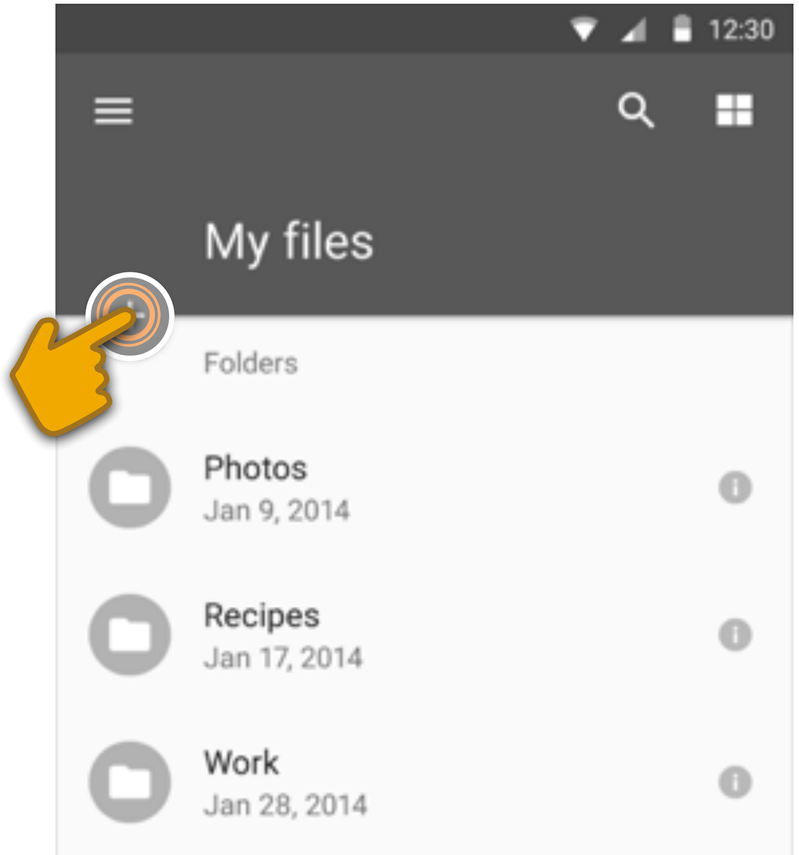


Motivation — User vs GUI vs Test Generator

Apps follow **interaction patterns**

Simple	Context-aware	**Imperceptible**

- *Layouts containing an image button can be clicked*



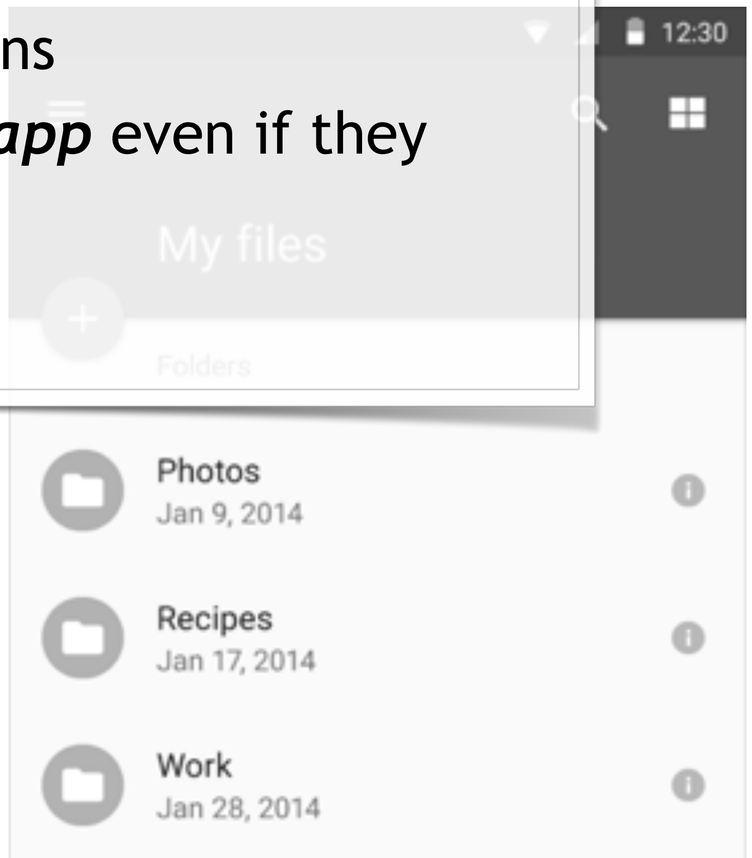
Motivation — User vs GUI vs Test Generator

Apps follow **interaction patterns**

Simple **Context-aware** **Imperceptible**

Users

- **Learn** and **identify** these patterns
- Know ***how to interact with an app*** even if they cannot identify its semantics

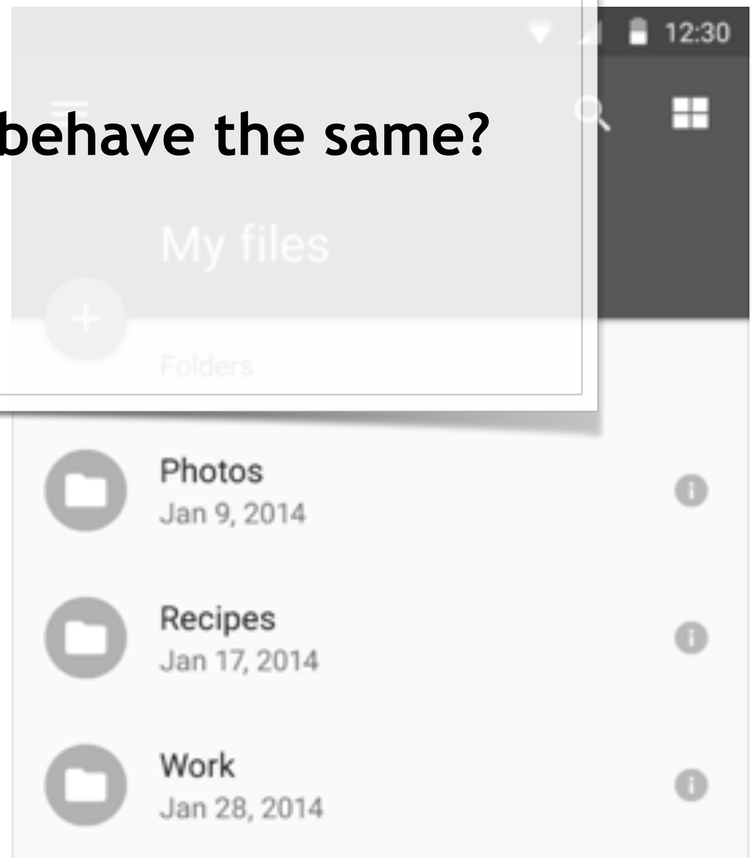


Motivation — User vs GUI vs Test Generator

Apps follow **interaction patterns**

Simple **Context-aware** **Imperceptible**

How to teach a test generator to behave the same?



Testing and the Multi-Armed-Bandit Problem

Given a budget and a set of competing slot machines, a player must decide which one to play.

By interacting with one machine as a time and observing its rewards it can learn, after some time, which slot machines it has to pull to maximize its profits.

Testing and the Multi-Armed-Bandit Problem

Given a budget and a set of competing slot machines, a player must decide which one to play.

By interacting with one machine as a time and observing its rewards it can learn, after some time, which slot machines it has to pull to maximize its profits.

Testing and the Multi-Armed-Bandit Problem

Given a number of actions and a set of competing UI elements, a test generator must decide which one to interact with.

By interacting with one UI element as a time and observing its rewards it can learn, after some time, which UI elements it has to interact with to maximize its reward.

Testing and the Multi-Armed-Bandit Problem

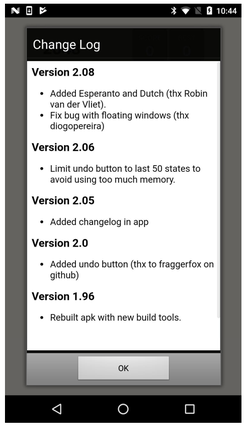
Given a number of actions and a set of competing UI elements, a test generator must decide which one to interact with.

By interacting with one UI element as a time and observing its rewards it can learn, after some time, which UI elements it has to interact with to maximize its reward.

Classic reinforcement learning problem

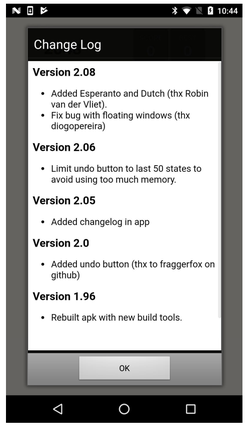
Testing with Reinforcement Learning

Testing with Reinforcement Learning

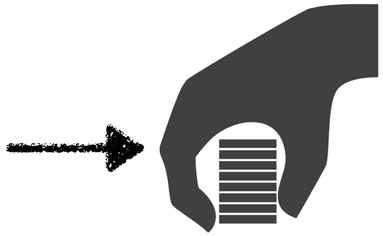


Inspect the UI

Testing with Reinforcement Learning

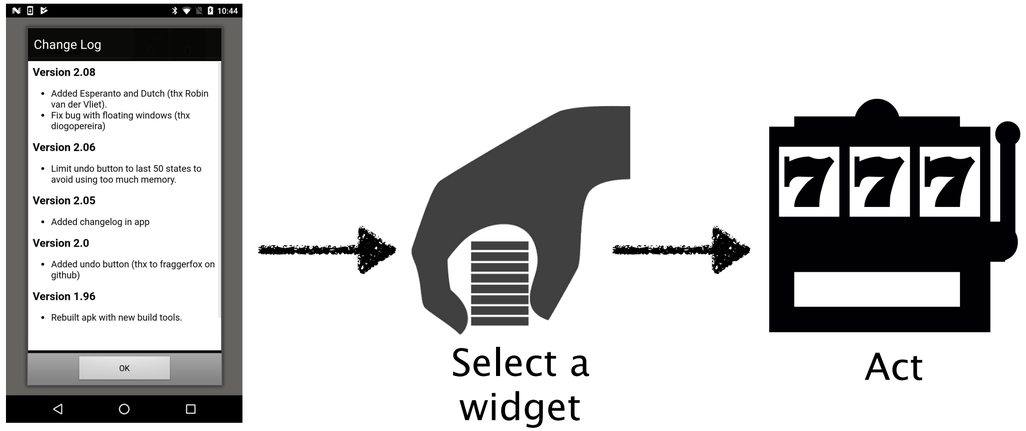


Inspect the UI

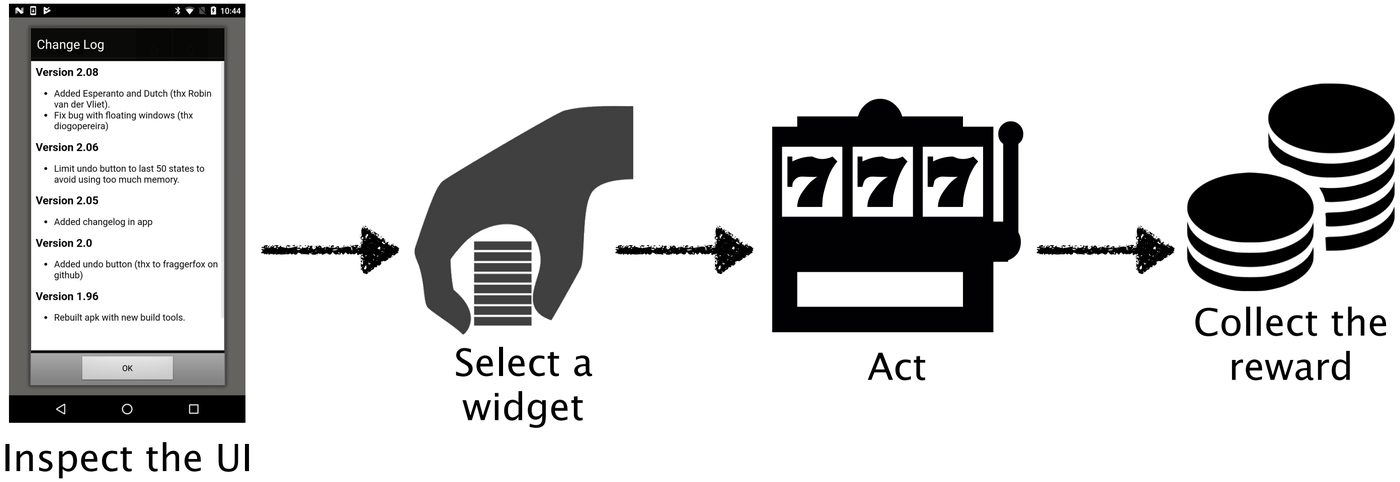


Select a widget

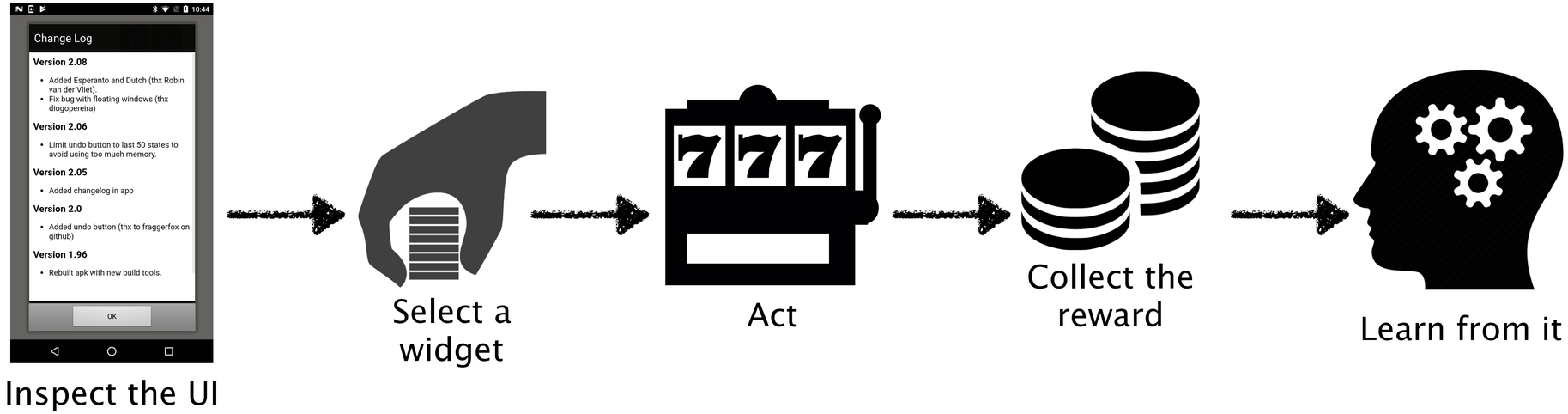
Testing with Reinforcement Learning



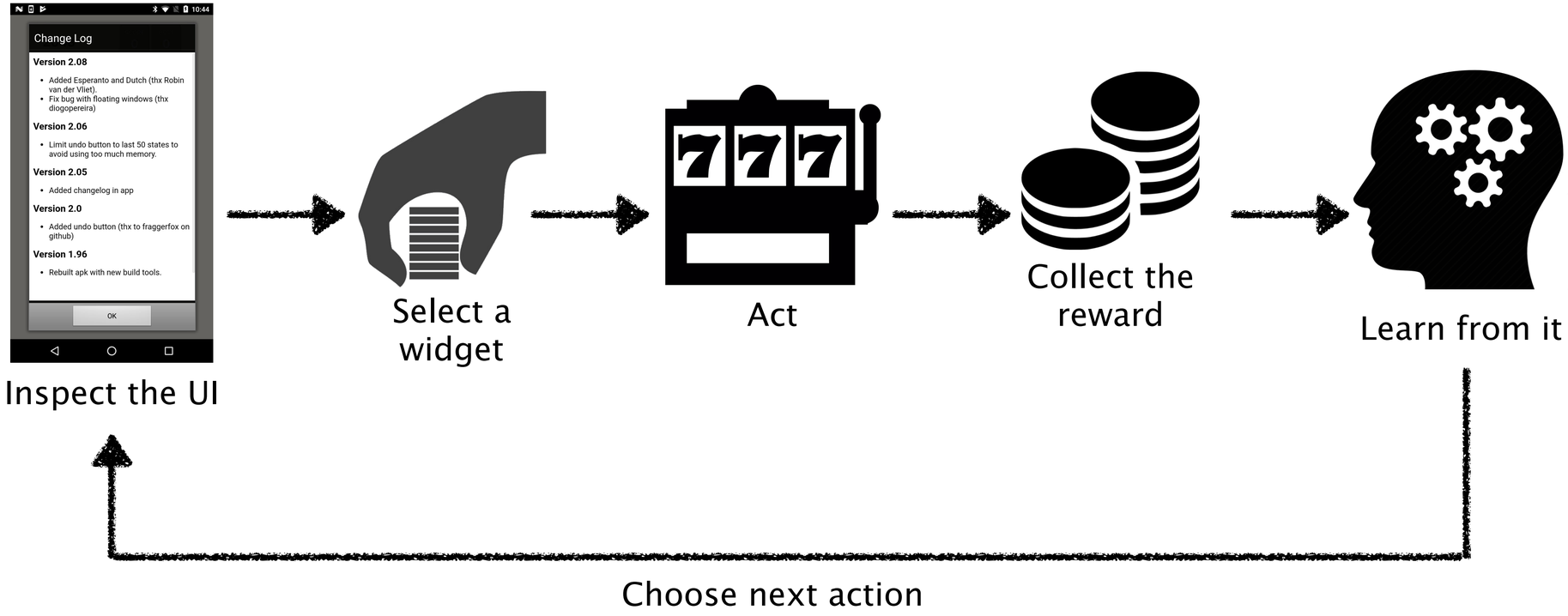
Testing with Reinforcement Learning



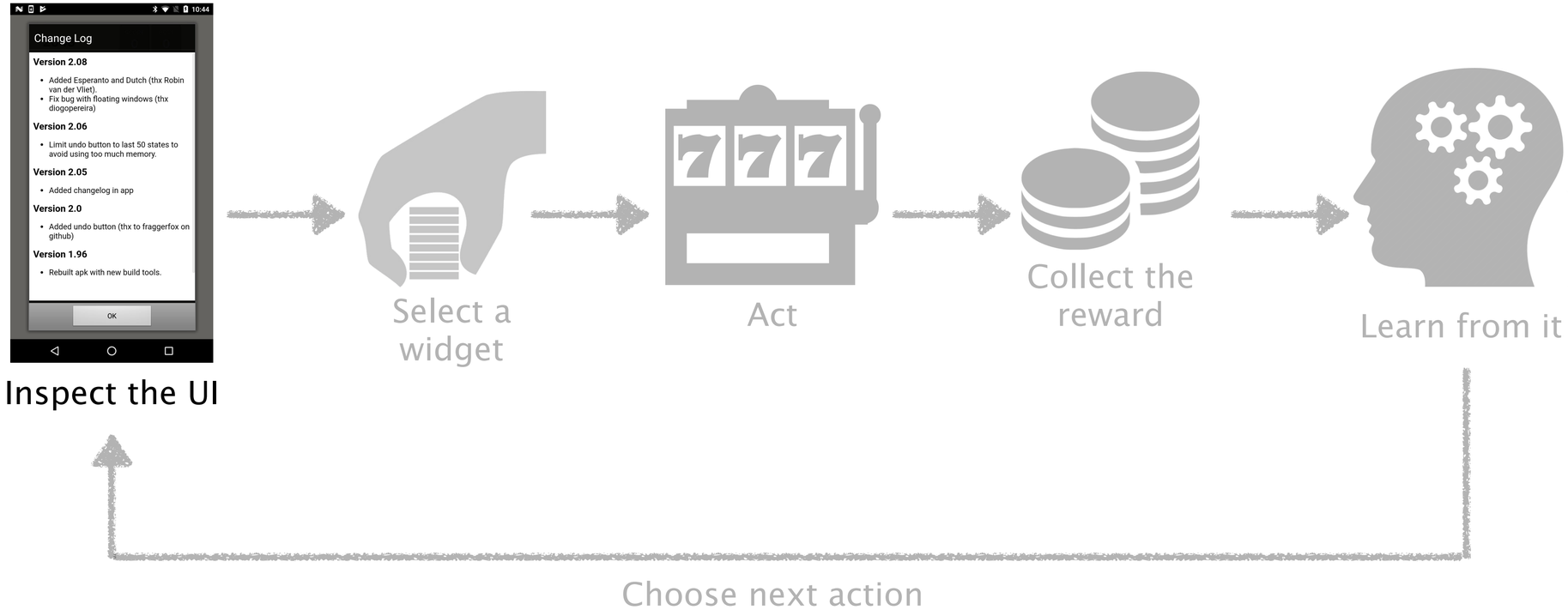
Testing with Reinforcement Learning



Testing with Reinforcement Learning



Testing with Reinforcement Learning



Fetch UI Elements

- All actionable UI elements
  - Extracted from the Accessibility Service

- All possible action types for each element
  - Clickable
  - Long Clickable
  - Scrollable
  - Tick-able
  - Swipe-able

Widget Classes

- Cluster features:
  - class type, parent type and children types

- Probabilities for each:
  - class and action type

Widget Classes

- Cluster features:
  - class type, parent type and children types

- Probabilities for each:
  - class and action type

Class	Parent Type	1st Children	2nd Children
Button	Linear Layout	—	—
(…)			
Label	Linear Layout	—	—

Widget Classes

- Cluster features:
  - class type, parent type and children types

- Probabilities for each:
  - class and action type

Class	Parent Type	1st Children	2nd Children	Action Type
Button	Linear Layout	—	—	**Click**
Button	Linear Layout	—	—	**Long Click**
Button	Linear Layout	—	—	**Swipe**
		(…)		
Label	Linear Layout	—	—	**Click**
Label	Linear Layout	—	—	**Long Click**

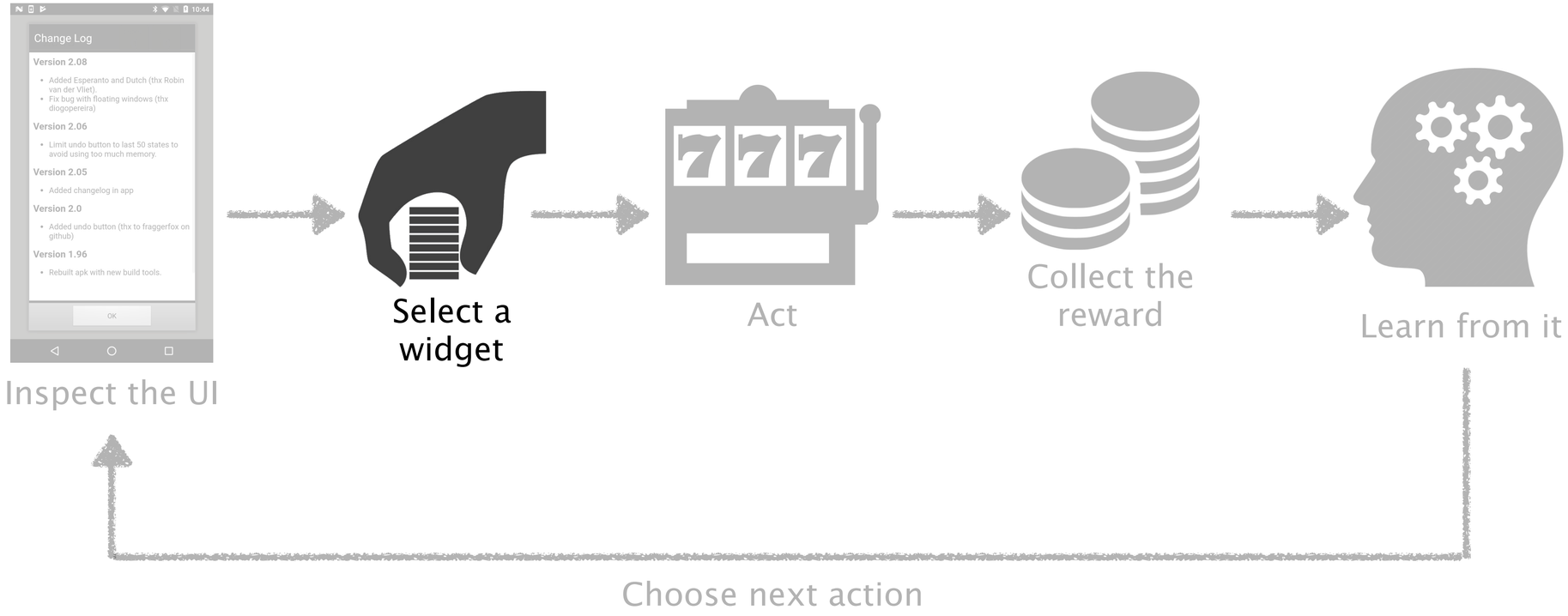
Widget Classes

- Cluster features:
  - class type, parent type and children types

- Probabilities for each:
  - class and action type

Class	Parent Type	1st Children	2nd Children	Action Type	Wins	Trials
Button	Linear Layout	—	—	**Click**	**x**	**X**
Button	Linear Layout	—	—	**Long Click**	**x’**	**X’**
Button	Linear Layout	—	—	**Swipe**	**x”**	**X”**
		(…)				
Label	Linear Layout	—	—	**Click**	**y**	**Y**
Label	Linear Layout	—	—	**Long Click**	**y'**	**Y’**

Testing with Reinforcement Learning

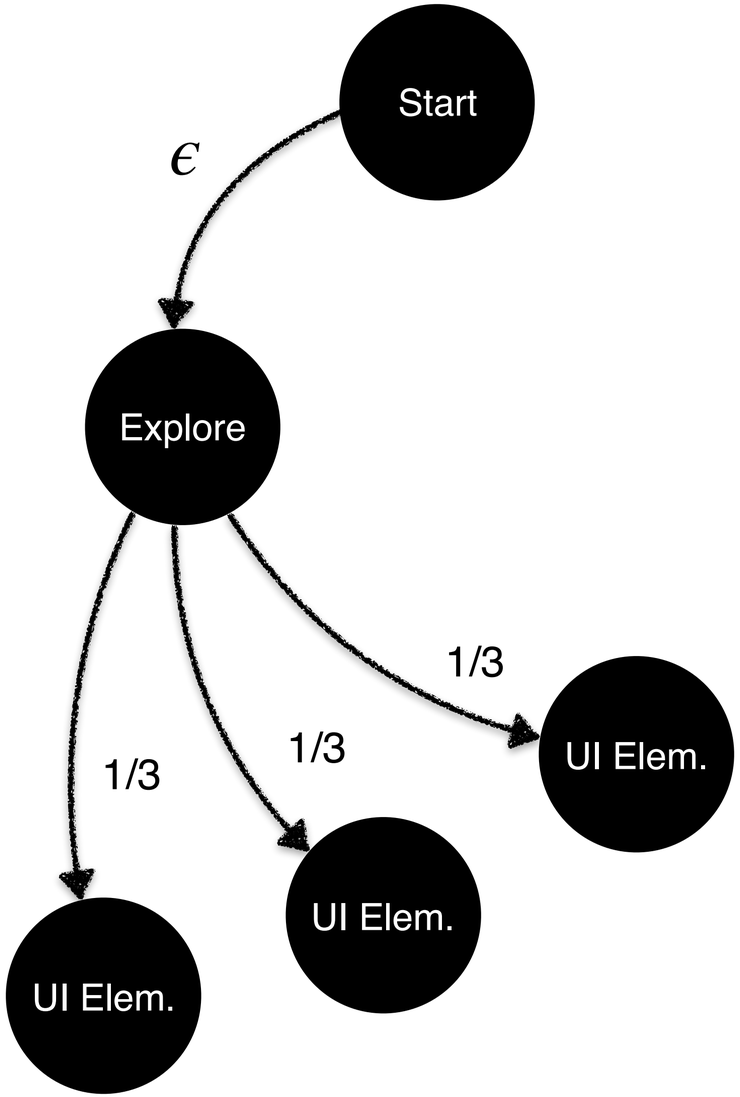


Reinforcement Learning Strategies

- ϵ-Greedy

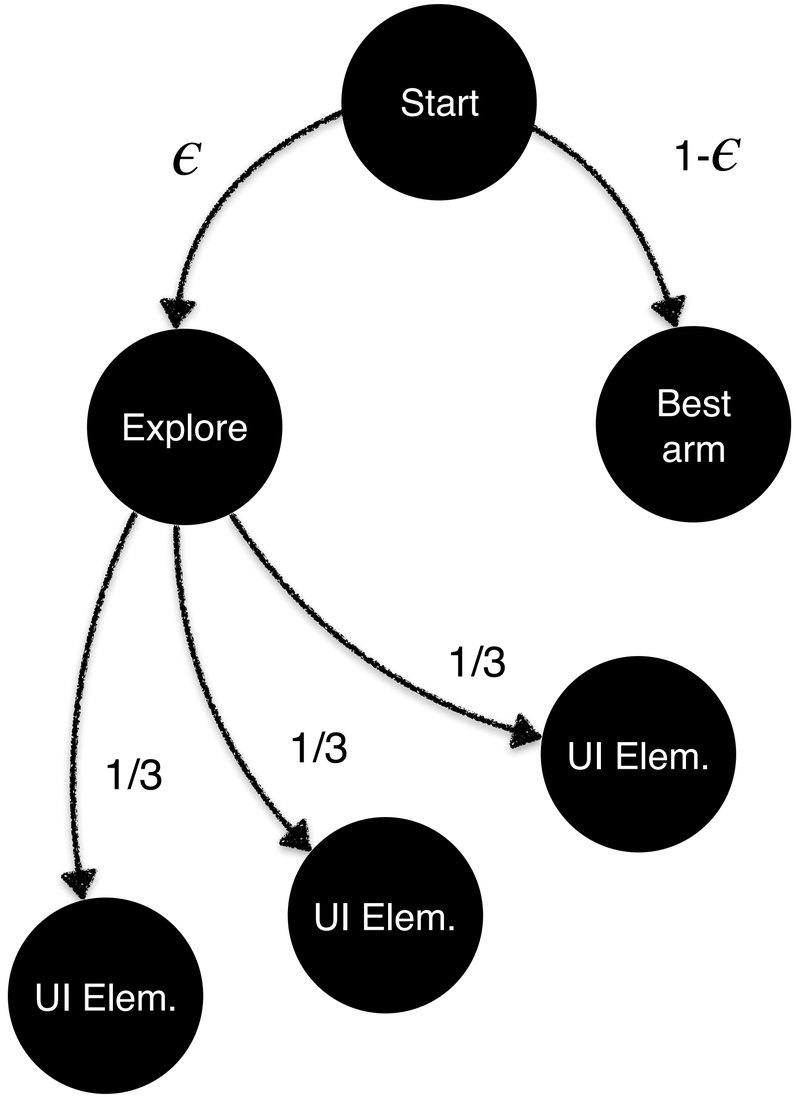
Reinforcement Learning Strategies

- ϵ-Greedy
  - ϵ chance of randomly selecting a UI element



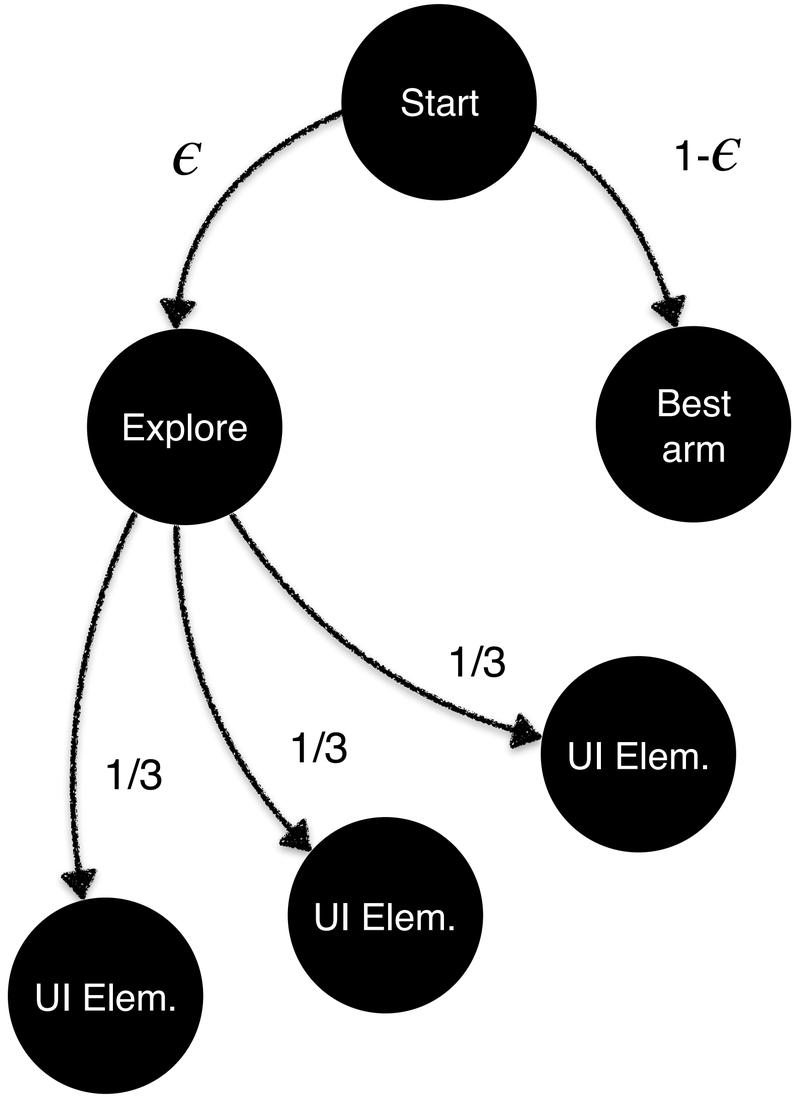
Reinforcement Learning Strategies

- ϵ-Greedy
  - ϵ chance of randomly selecting a UI element
  - 1-ϵ chance of selecting the best UI element



Reinforcement Learning Strategies

- ϵ-Greedy
  - ϵ chance of randomly selecting a UI element
  - 1-ϵ chance of selecting the best UI element
- ϵ balances between *exploration* and *exploitation*

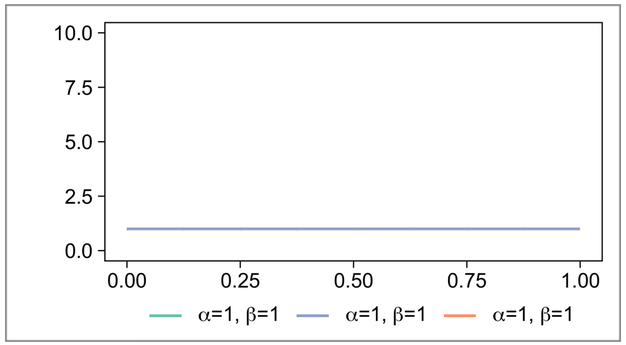


Reinforcement Learning Strategies

- Thompson Sampling

Reinforcement Learning Strategies

- Thompson Sampling
  - Wins and trials
  - β distribution

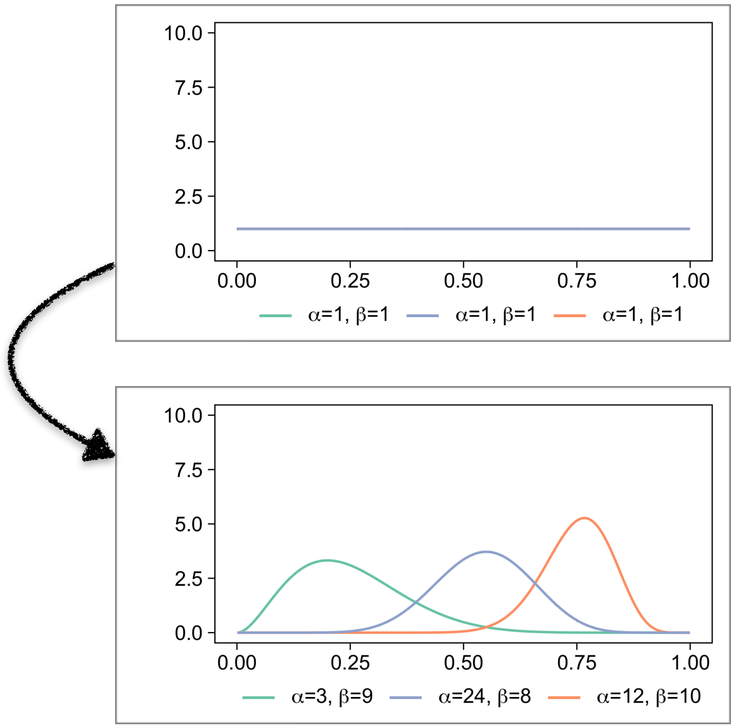


Reinforcement Learning Strategies

- Thompson Sampling
  - Wins and trials
  - β distribution

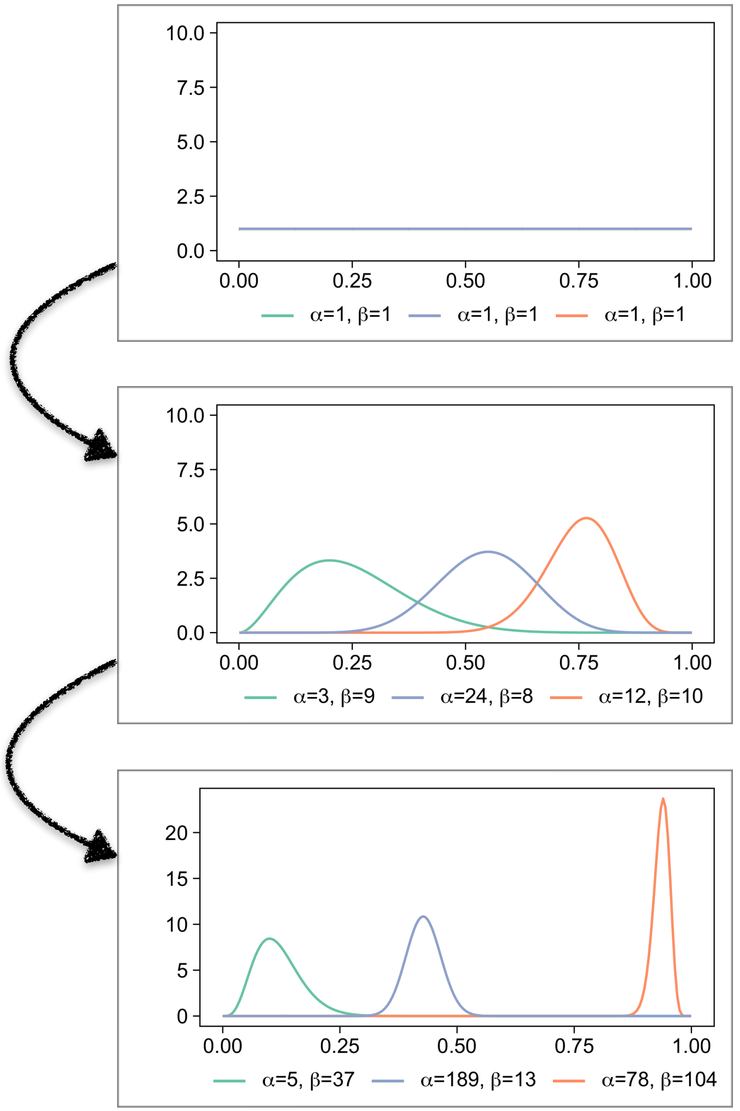
  - Random sample
  - Select class with best sample

  - Increase trials and wins after each action

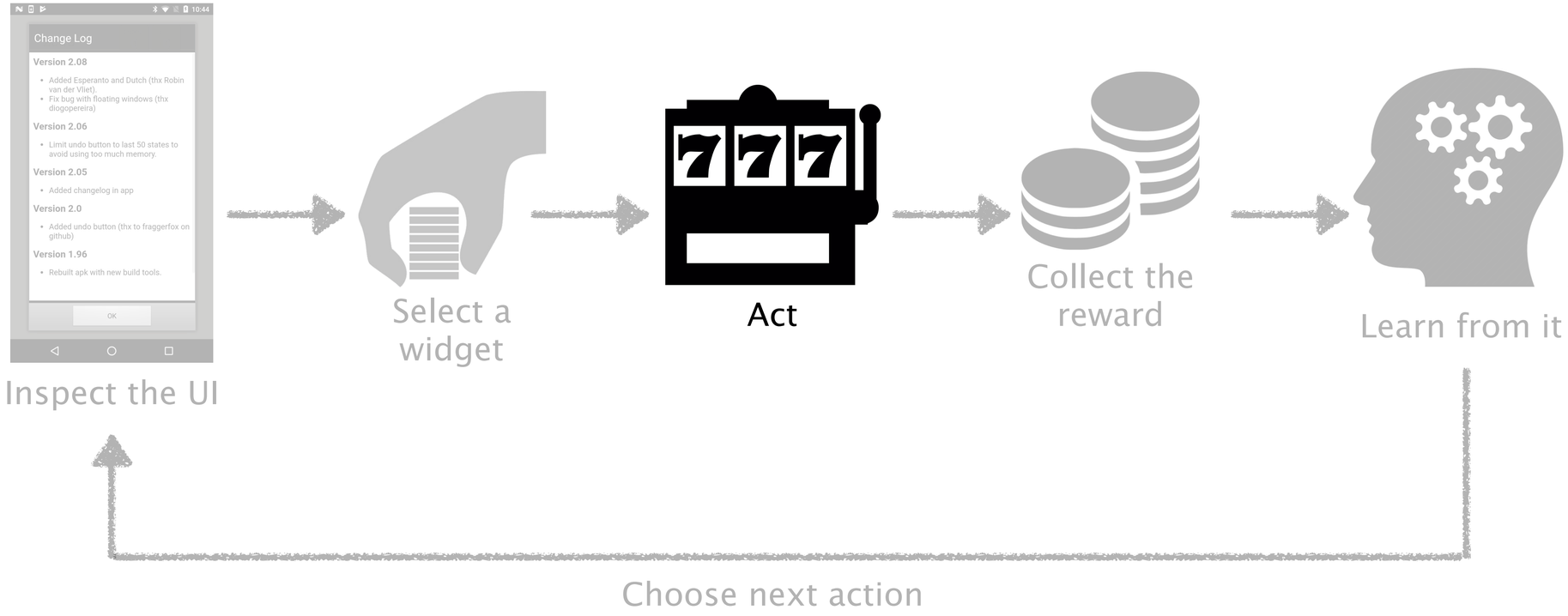


Reinforcement Learning Strategies

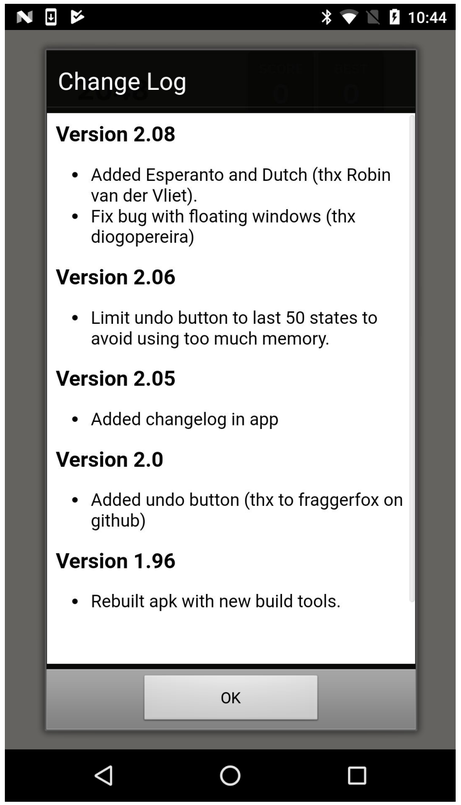
- Thompson Sampling
  - Wins and trials
  - β distribution
  - Random sample
  - Select class with best sample
  - Increase trials and wins after each action
- Distributions adjust over time



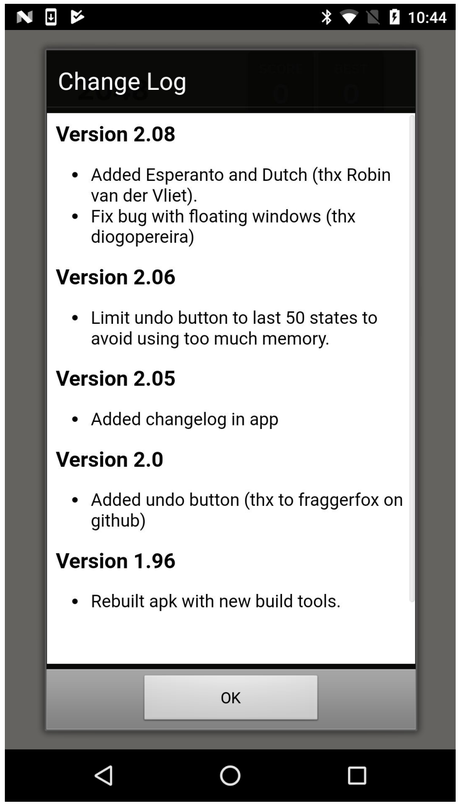
Testing with Reinforcement Learning



Convert Chosen Widget—Class into Action

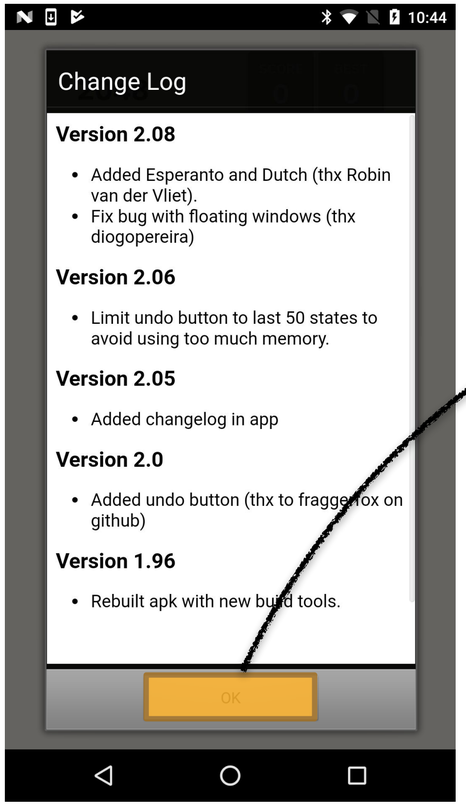


Convert Chosen Widget—Class into Action



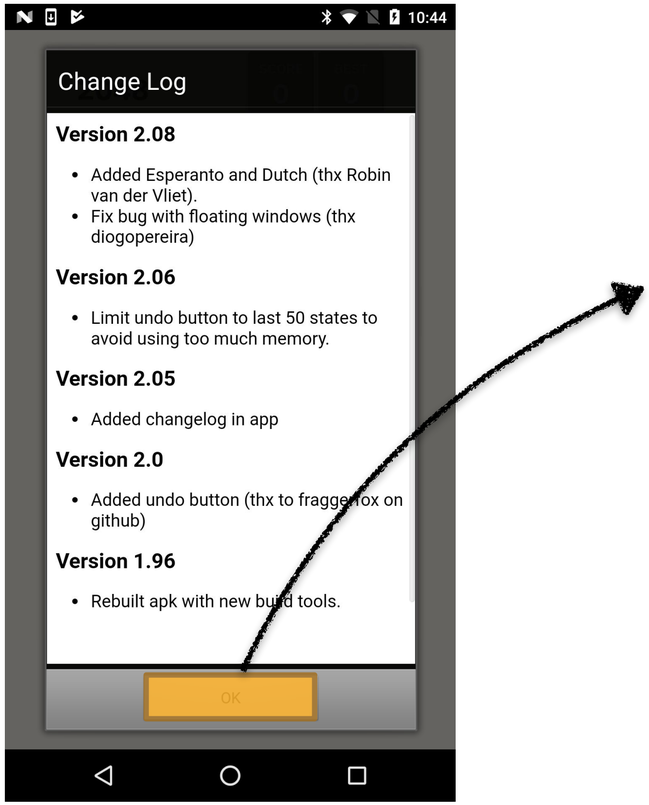
Class	Parent Type	1st Children	2nd Children	Action Type	Wins	Trials
Button	Linear Layout	—	—	**Click**	**x**	**X**
Button	Linear Layout	—	—	**Long Click**	**x'**	**X'**
Button	Linear Layout	—	—	**Swipe**	**x''**	**X''**
		(…)				
Label	Linear Layout	—	—	**Click**	**y**	**Y**
Label	Linear Layout	—	—	**Long Click**	**y'**	**Y'**

Convert Chosen Widget—Class into Action



Class	Parent Type	1st Children	2nd Children	Action Type	Wins	Trials
Button	Linear Layout	—	—	**Click**	**x**	**X**
Button	Linear Layout	—	—	**Long Click**	**x'**	**X'**
Button	Linear Layout	—	—	**Swipe**	**x''**	**X''**
		(…)				
Label	Linear Layout	—	—	**Click**	**y**	**Y**
Label	Linear Layout	—	—	**Long Click**	**y'**	**Y'**

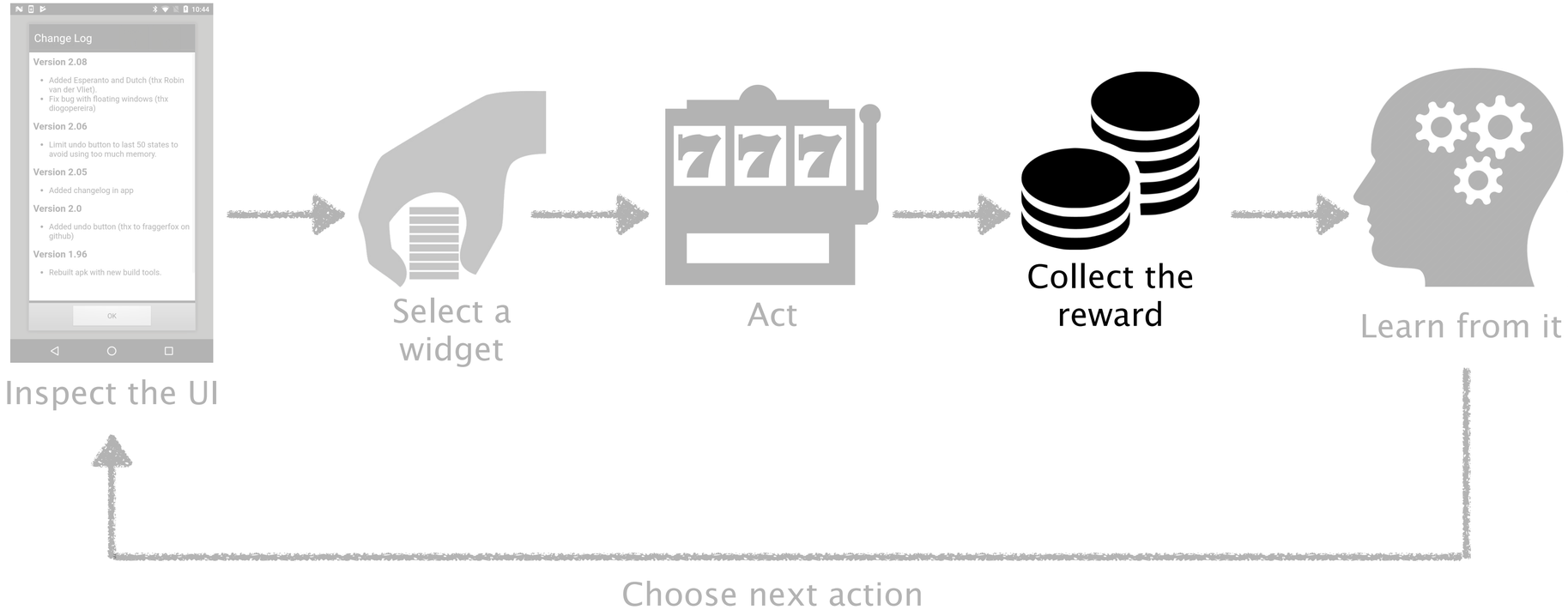
Convert Chosen Widget—Class into Action



Click on “OK" button

Class	Parent Type	1st Children	2nd Children	Action Type	Wins	Trials
Button	Linear Layout	—	—	**Click**	**x**	**X**
Button	Linear Layout	—	—	**Long Click**	**x’**	**X’**
Button	Linear Layout	—	—	**Swipe**	**x”**	**X”**
		(…)				
Label	Linear Layout	—	—	**Click**	**y**	**Y**
Label	Linear Layout	—	—	**Long Click**	**y'**	**Y’**

Testing with Reinforcement Learning



Binary Reward (Bernoulli Multi-Armed Bandit)

Reward	Condition
1	If app state changed
0	Otherwise

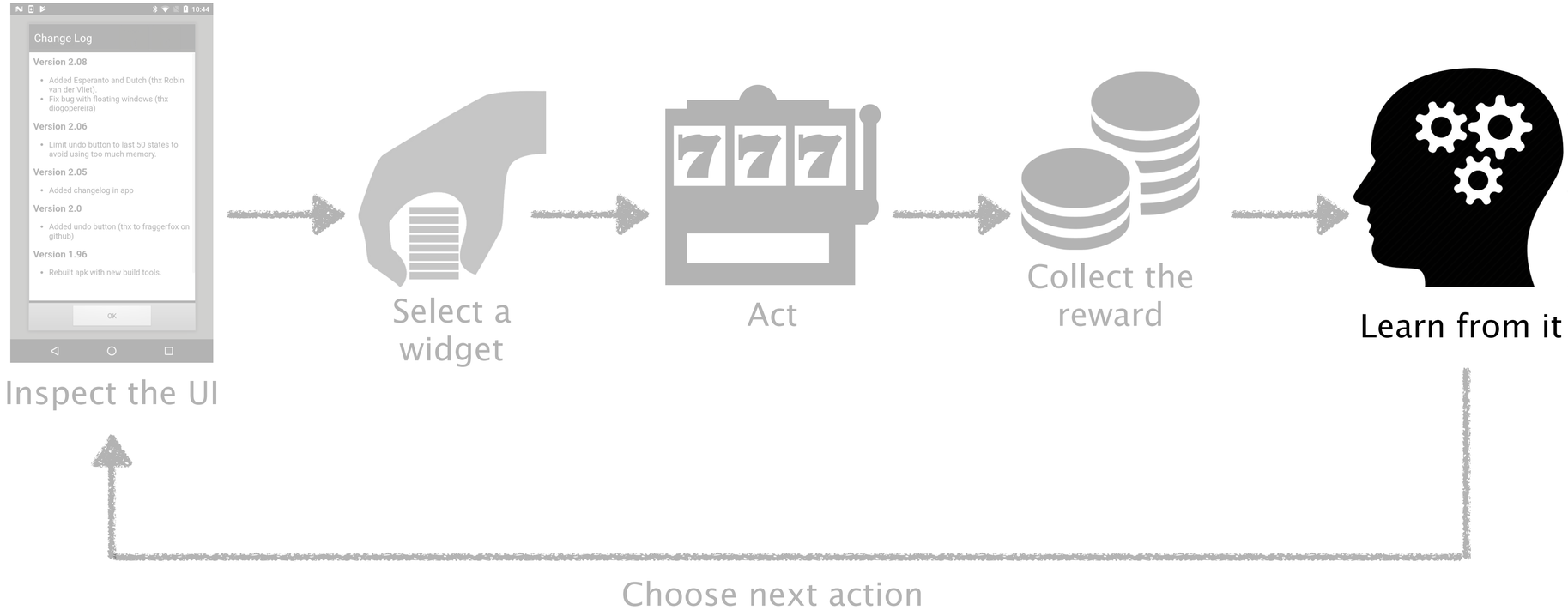
Binary Reward (Bernoulli Multi-Armed Bandit)

Reward	Condition
1	If app state changed
0	Otherwise

A state changes if any UI element is

added, removed, moved, resized, changes text (non-input)

Testing with Reinforcement Learning

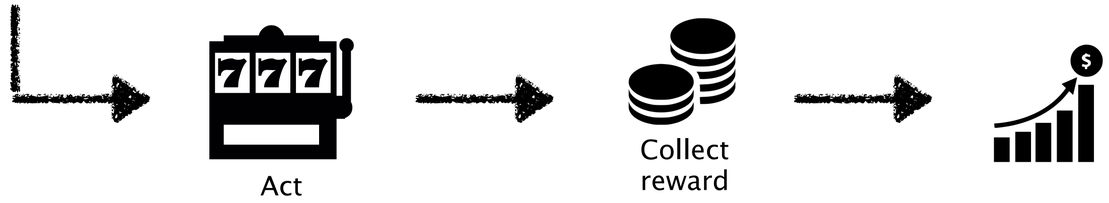


Learn from Rewards

Class	Parent Type	1st Children	2nd Children	Action Type	Wins	Trials
Button	Linear Layout	—	—	Click	1	1

Learn from Rewards

Class	Parent Type	1st Children	2nd Children	Action Type	Wins	Trials
Button	Linear Layout	—	—	Click	1	1



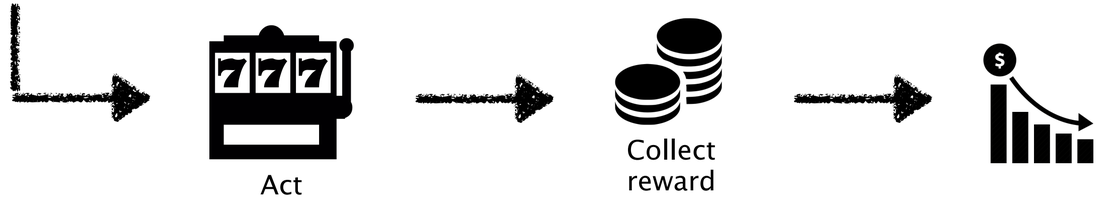
Learn from Rewards

Class	Parent Type	1st Children	2nd Children	Action Type	Wins	Trials
Button	Linear Layout	—	—	Click	1	1
						
Button	Linear Layout	—	—	Click	**2**	**2**

Learn from Rewards

Class	Parent Type	1st Children	2nd Children	Action Type	Wins	Trials
Button	Linear Layout	—	—	Click	1	1

Button	Linear Layout	—	—	Click	2	2



Learn from Rewards

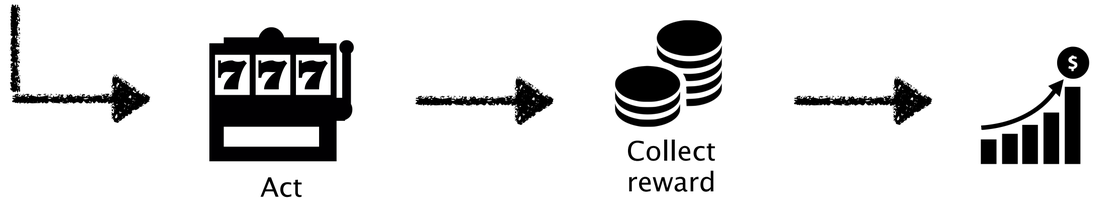
Class	Parent Type	1st Children	2nd Children	Action Type	Wins	Trials
Button	Linear Layout	—	—	Click	1	1
						
Button	Linear Layout	—	—	Click	2	2
						
Button	Linear Layout	—	—	Click	2	3

Learn from Rewards

Class	Parent Type	1st Children	2nd Children	Action Type	Wins	Trials
Button	Linear Layout	—	—	Click	1	1

Button	Linear Layout	—	—	Click	2	2

Button	Linear Layout	—	—	Click	2	3



Learn from Rewards

Class	Parent Type	1st Children	2nd Children	Action Type	Wins	Trials
Button	Linear Layout	—	—	Click	1	1
Button	Linear Layout	—	—	Click	2	2
Button	Linear Layout	—	—	Click	2	3
Button	Linear Layout	—	—	Click	3	4

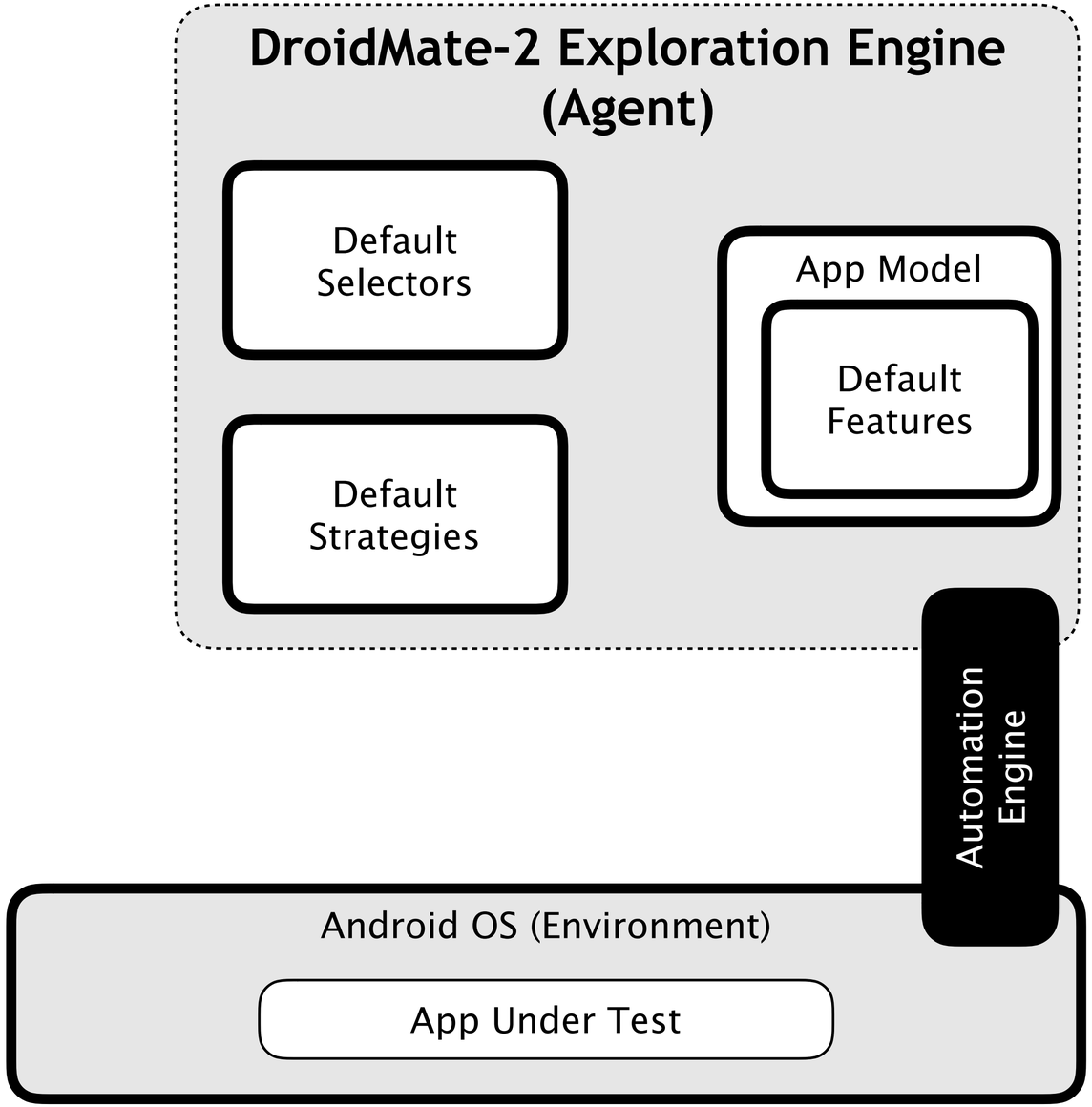
Evaluation

1. Can Multi-Armed-Bandit approaches effectively test apps?

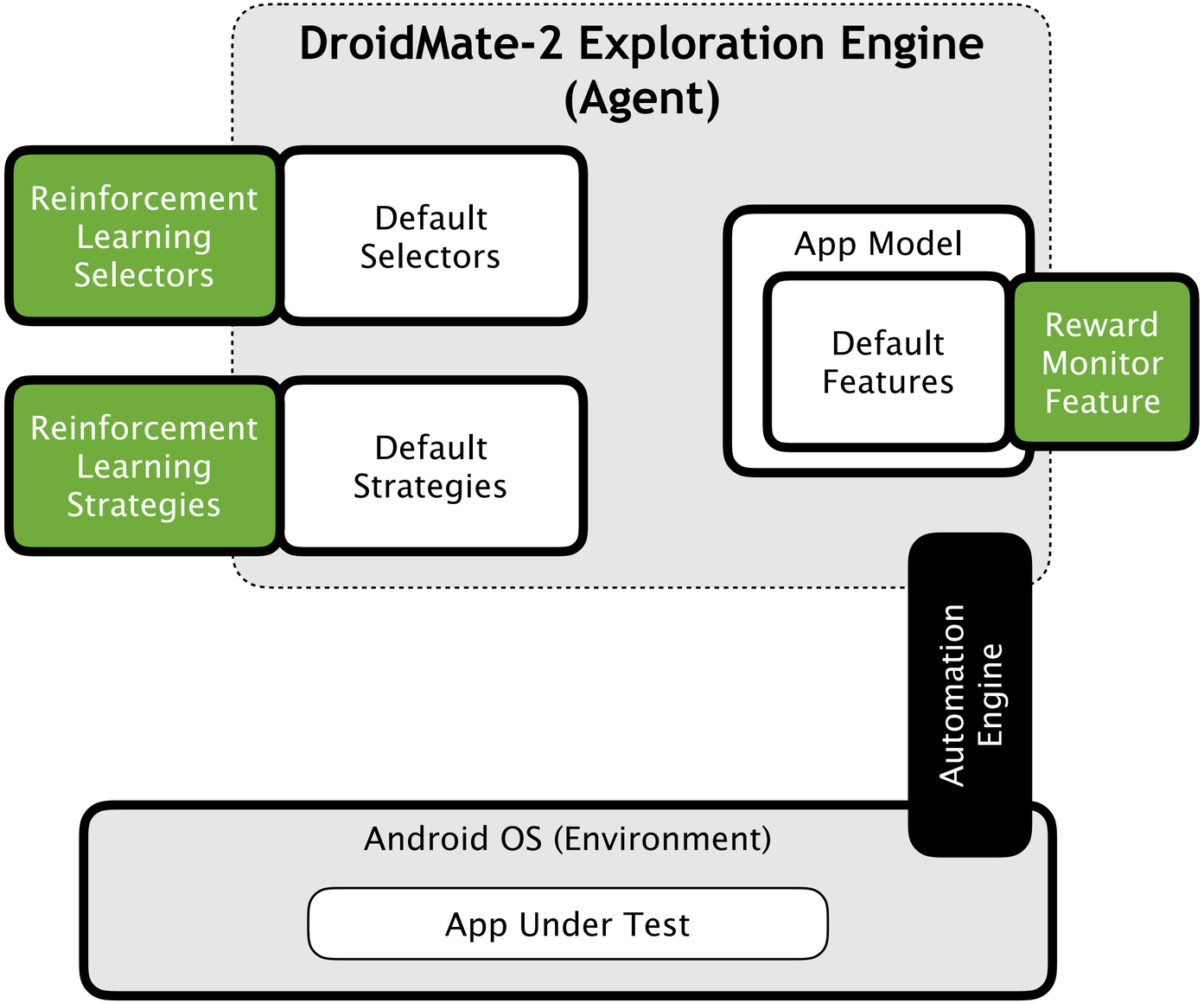
2. Is dynamic knowledge more beneficial to testing than static models from other apps?

3. Can Multi-Armed-Bandit approaches be used to enhance static models?

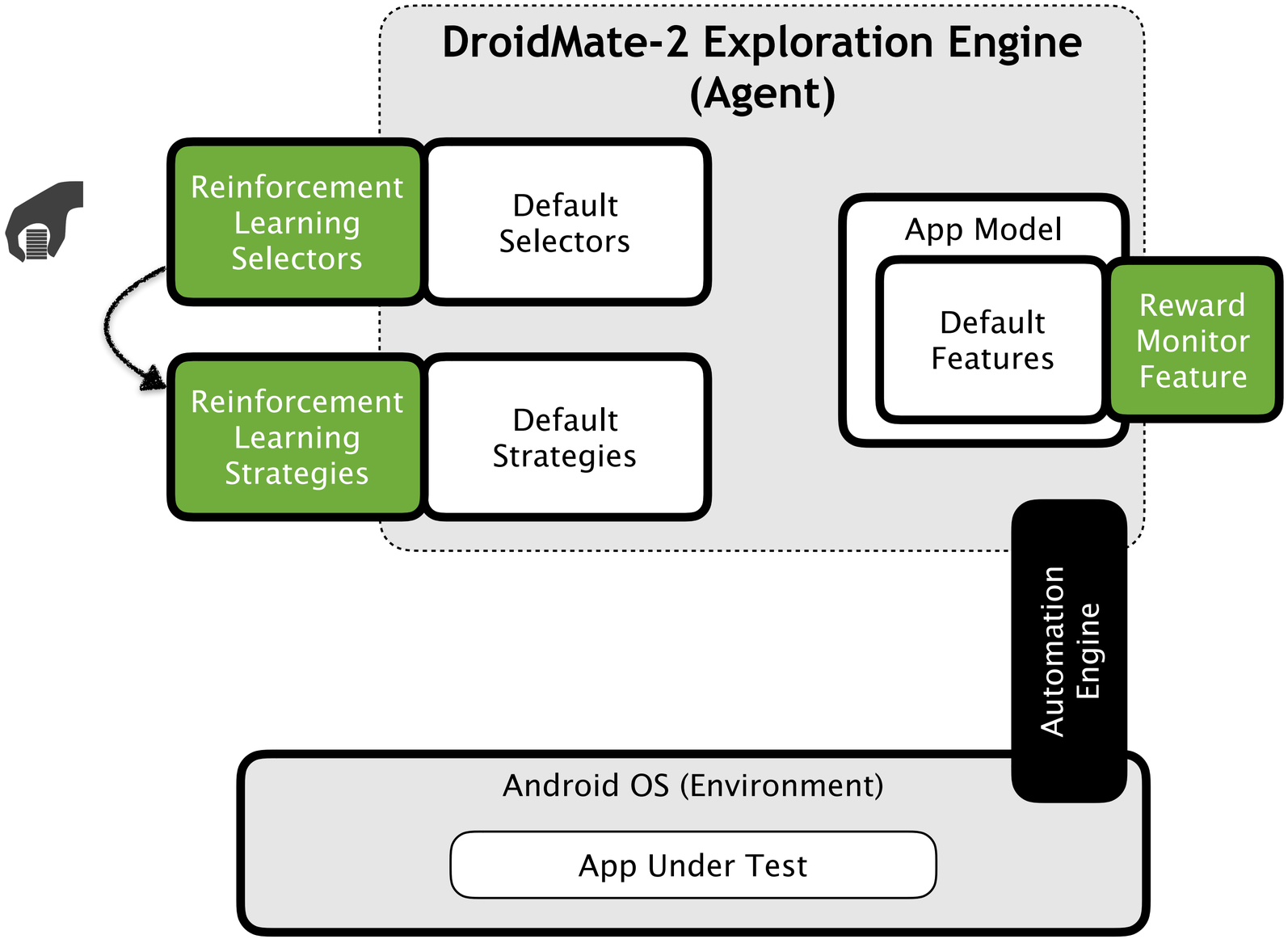
Design: Exploit DroidMate-2 Exploration Engine



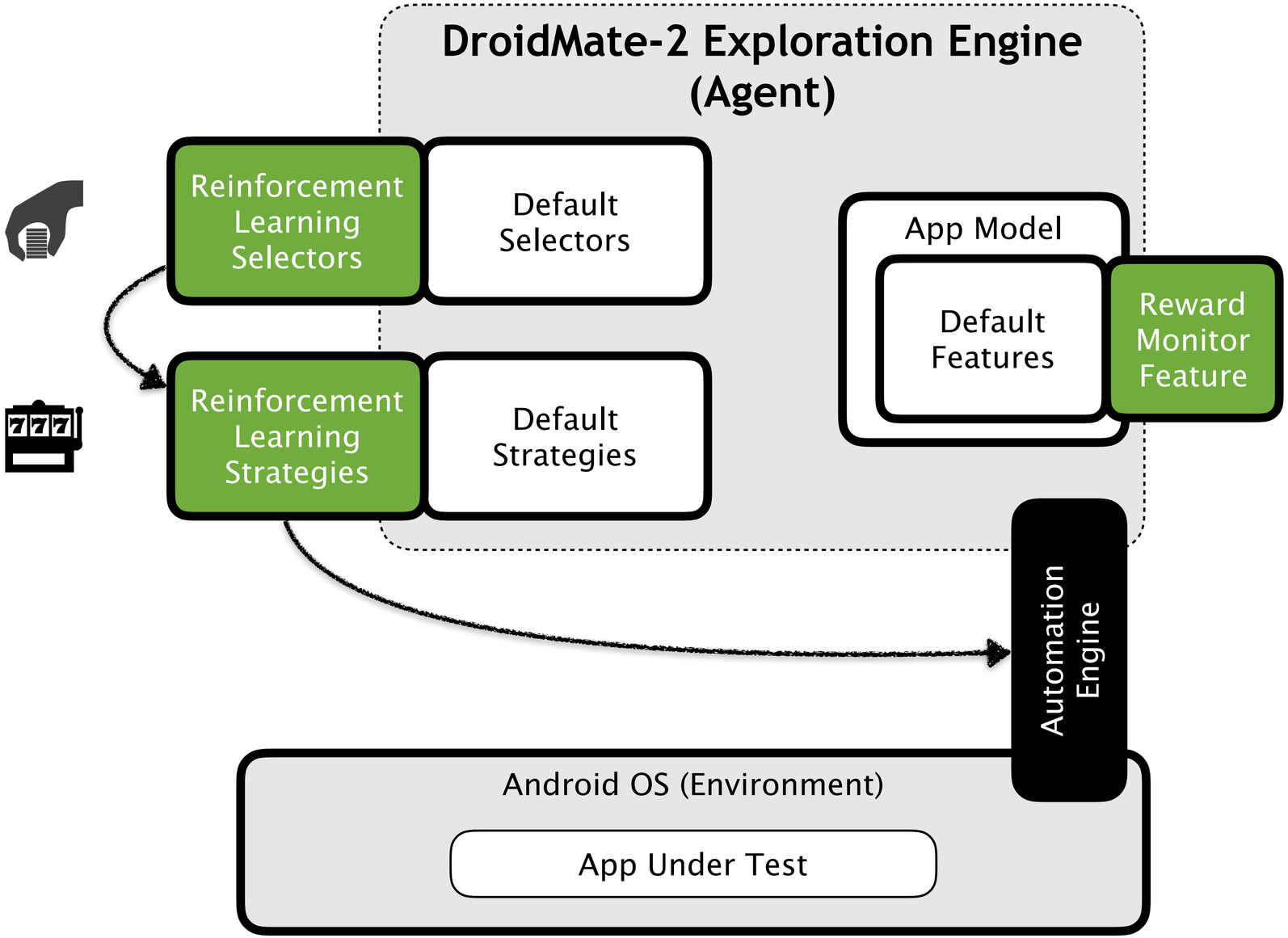
Design: Exploit DroidMate-2 Exploration Engine



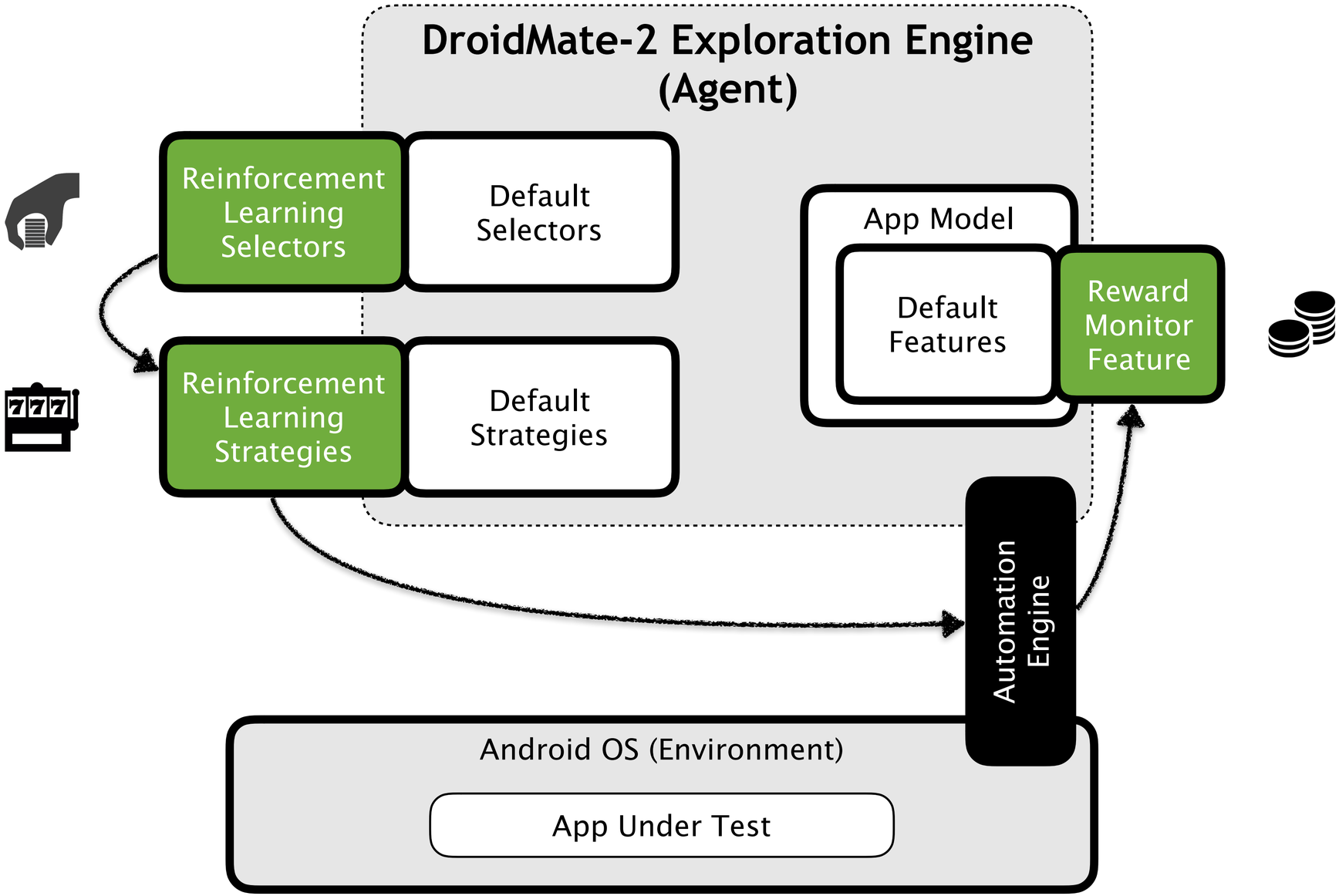
Design: Exploit DroidMate-2 Exploration Engine



Design: Exploit DroidMate-2 Exploration Engine



Design: Exploit DroidMate-2 Exploration Engine

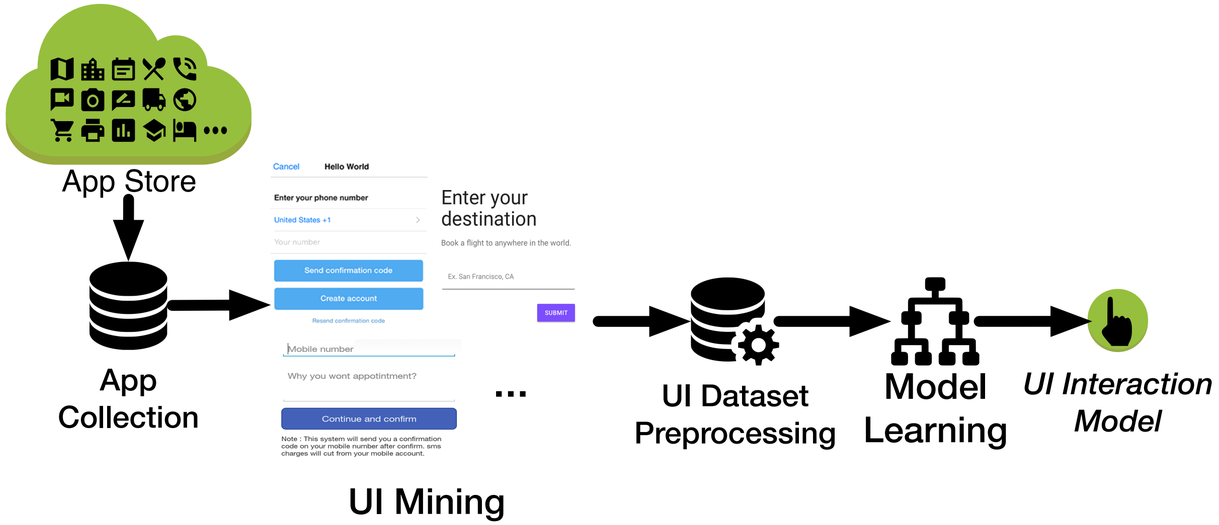


Baseline

- Crowd-Model to Predict Events
  - Static analysis to build a prediction model from multiple apps
  - Reuse the model to test arbitrary apps without any further analysis

- ~19% improvement when compared to random on the same tool

- Reused all possible apps from the original experiment

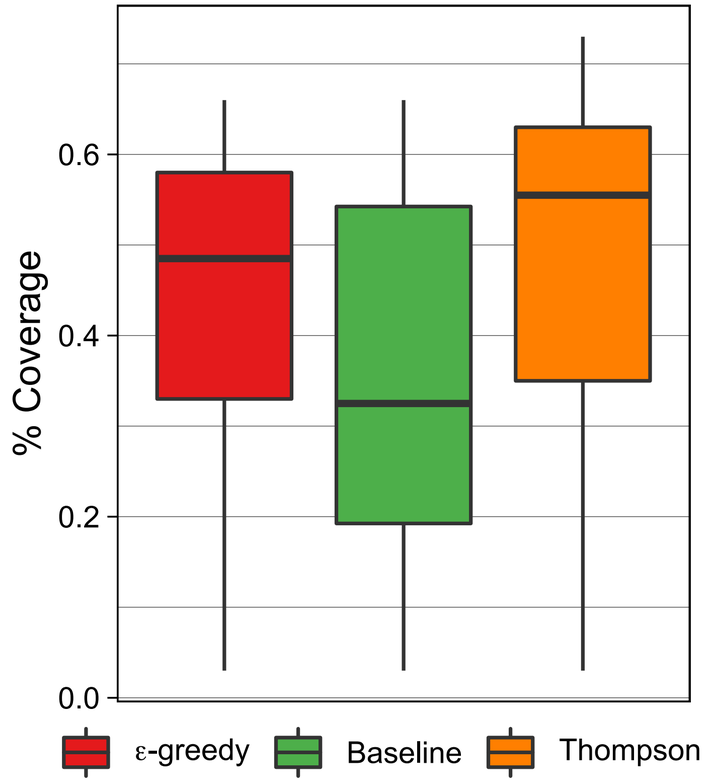


Evaluation — Can Multi-Armed-Bandit approaches effectively test apps?

- **Statement coverage**
  - Baseline vs ϵ-Greedy vs Thompson Sampling
  - ϵ = 0.3
  - 10 runs per app

Evaluation — Can Multi-Armed-Bandit approaches effectively test apps?

- **Statement coverage**
  - Baseline vs ϵ-Greedy vs Thompson Sampling
  - ϵ = 0.3
  - 10 runs per app



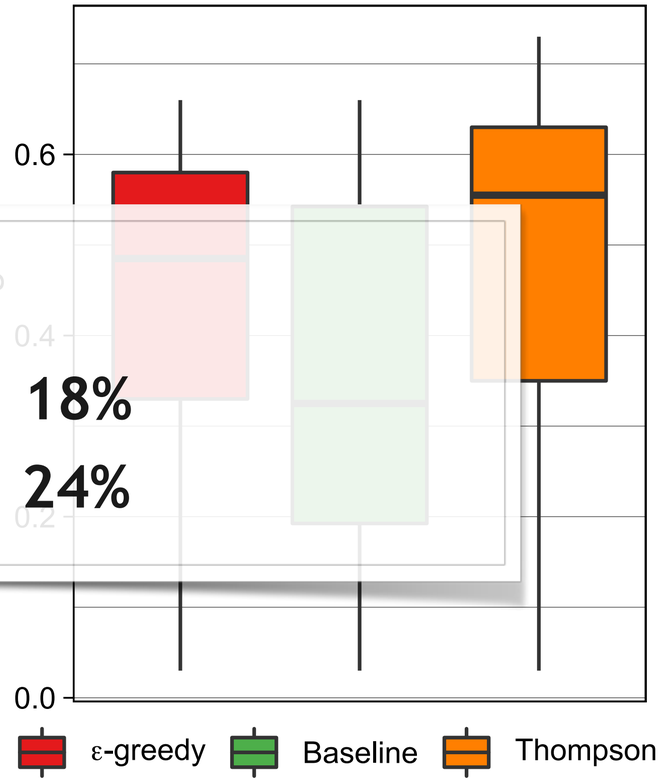
Evaluation — Can Multi-Armed-Bandit approaches effectively test apps?

- **Statement coverage**
  - Baseline vs ϵ-Greedy vs Thompson Sampling
  - ϵ = 0.3
  - 10 runs per app

Average coverage increase:

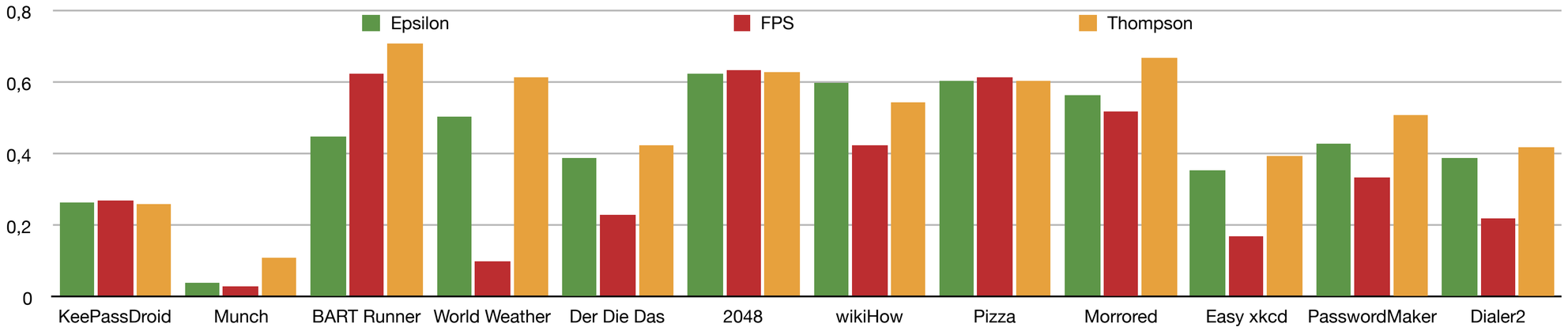
ϵ-Greedy → **18%**

Thompson Sampling → **24%**



Evaluation — Can Multi-Armed-Bandit approaches effectively test apps?

- **Statement coverage**
  - Baseline vs ϵ-Greedy vs Thompson Sampling
  - ϵ = 0.3
  - 10 runs per app

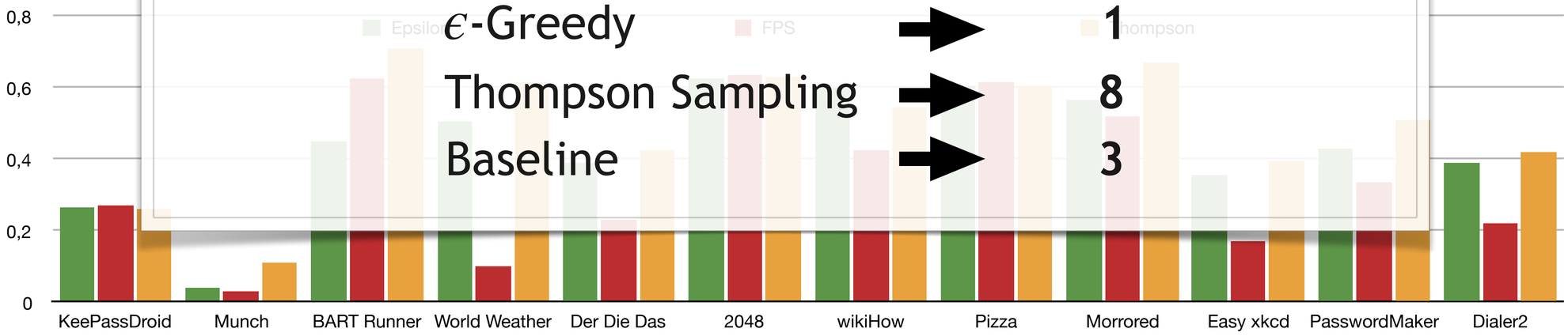


Evaluation — Can Multi-Armed-Bandit approaches effectively test apps?

- **Statement coverage**
  - Baseline vs ϵ-Greedy vs Thompson Sampling
  - ϵ = 0.3
  - 10 runs per app

Most coverage:

ϵ-Greedy	→	1
Thompson Sampling	→	8
Baseline	→	3



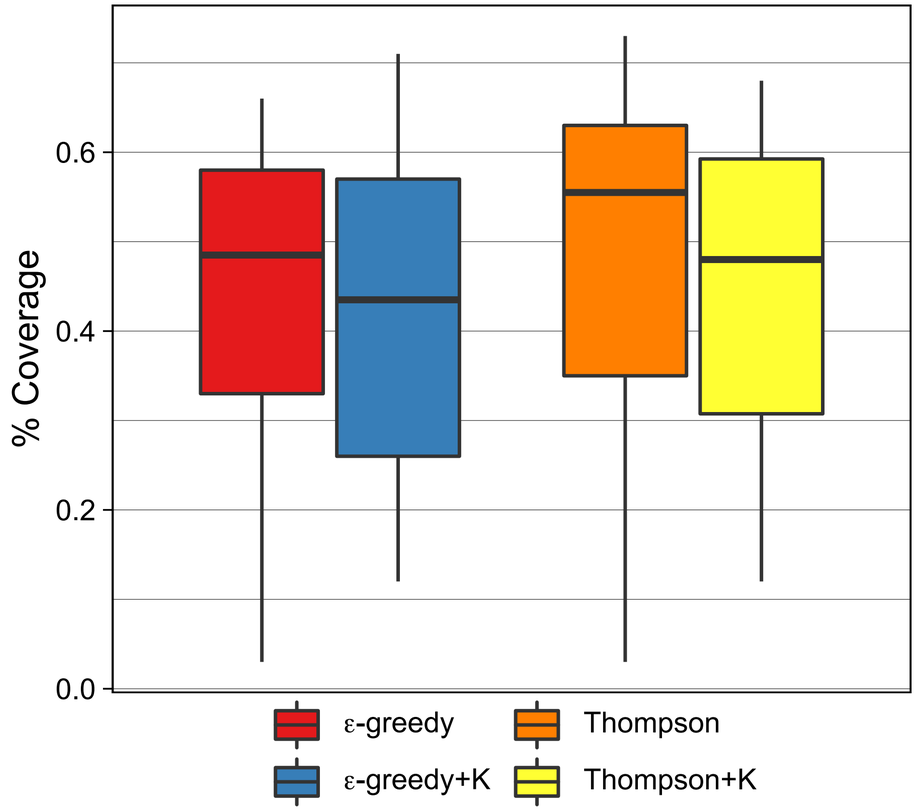
Evaluation — Dynamic vs Static Knowledge?

- **Statement coverage**
  - ϵ-Greedy vs ϵ-Greedy+K
  - Thompson Sampling vs Thompson Sampling+K
  - 10 runs per app

  - Initialized trials and wins based on crowd model probabilities

Evaluation — Dynamic vs Static Knowledge?

- **Statement coverage**
  - ϵ-Greedy vs ϵ-Greedy+K
  - Thompson Sampling vs Thompson Sampling+K
  - 10 runs per app
  - Initialized trials and wins based on crowd model probabilities



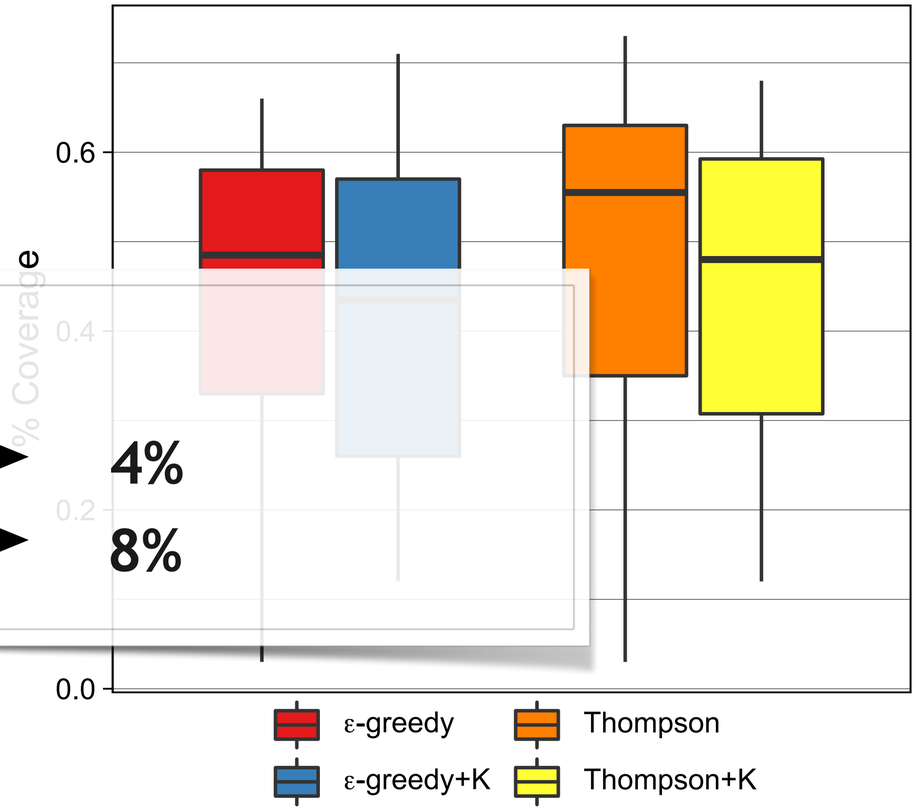
Evaluation — Dynamic vs Static Knowledge?

- **Statement coverage**
  - ϵ-Greedy vs ϵ-Greedy+K
  - Thompson Sampling vs Thompson Sampling+K
  - 10 runs per app
  - Initialized trials and wins based on crowd-model probabilities

Static models decrease coverage by:

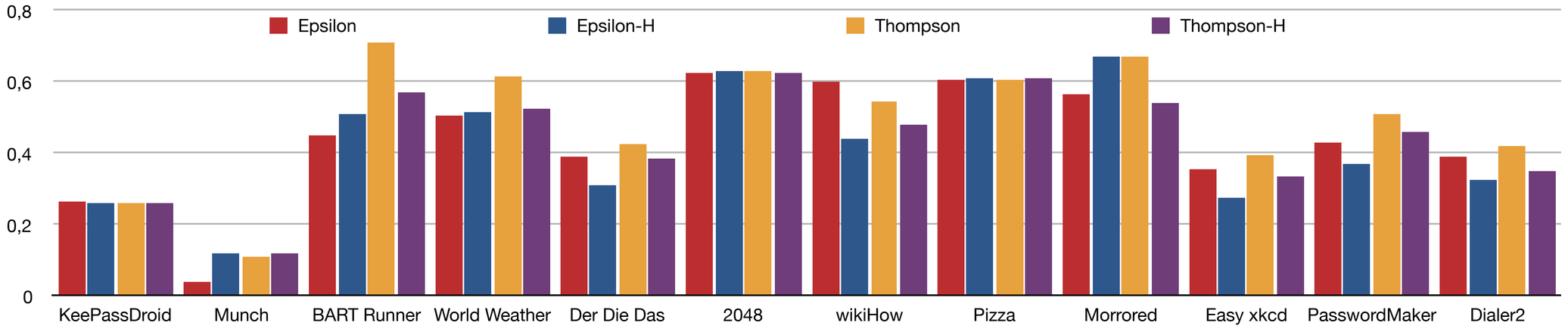
ϵ-Greedy → **4%**

Thompson Sampling → **8%**



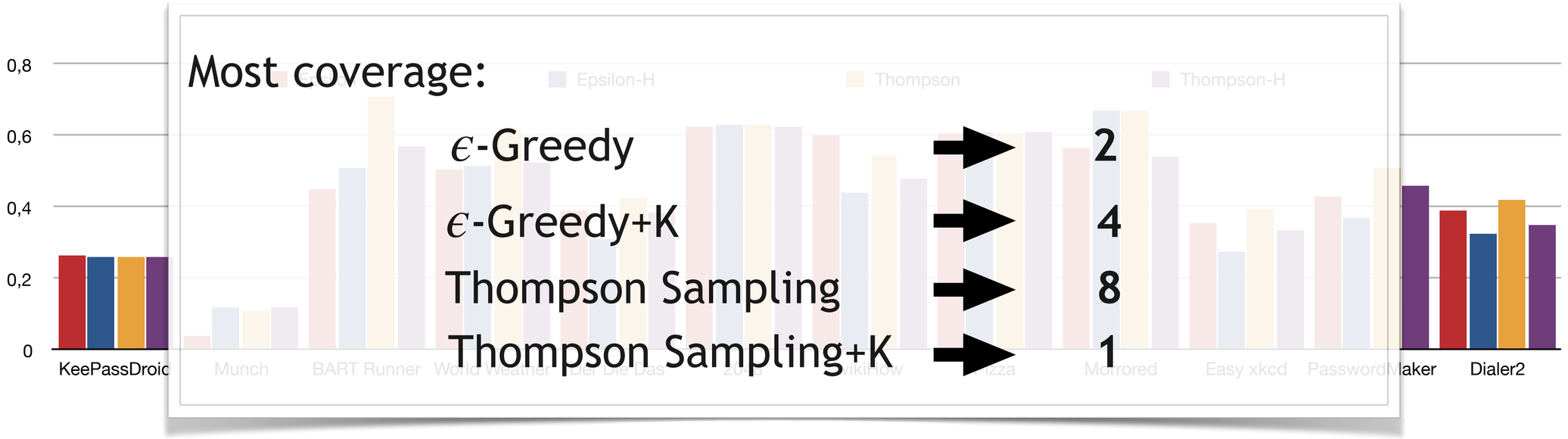
Evaluation — Dynamic vs Static Knowledge?

- **Statement coverage**
  - ϵ-Greedy vs ϵ-Greedy+K vs Thompson Sampling vs Thompson Sampling+K
  - 10 runs per app



Evaluation — Dynamic vs Static Knowledge?

- **Statement coverage**
  - ϵ-Greedy vs ϵ-Greedy+K vs Thompson Sampling vs Thompson Sampling+K
  - 10 runs per app

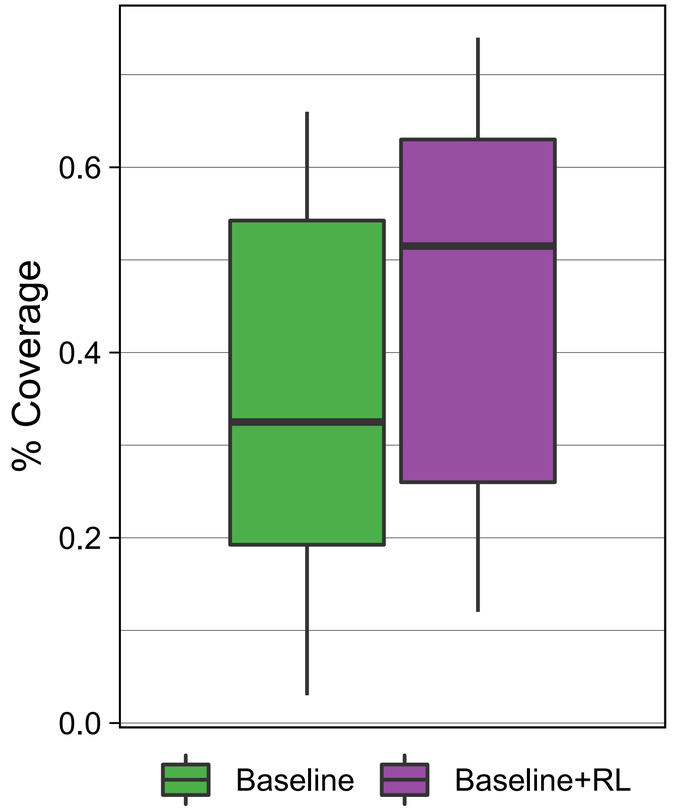


Evaluation — Multi-Armed-Bandit approaches to enhance static models?

- **Statement coverage**
  - Baseline vs Baseline+RL
  - Roulette wheel selection algorithm from Baseline+RL
  - 10 runs per app

Evaluation — Multi-Armed-Bandit approaches to enhance static models?

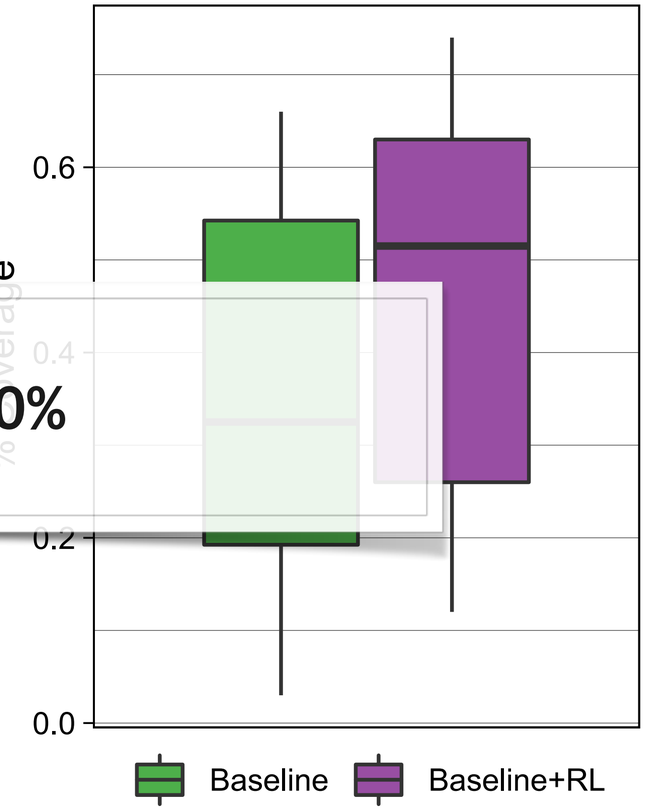
- **Statement coverage**
  - Baseline vs Baseline+RL
  - Roulette wheel selection algorithm from Baseline+RL
  - 10 runs per app



Evaluation — Multi-Armed-Bandit approaches to enhance static models?

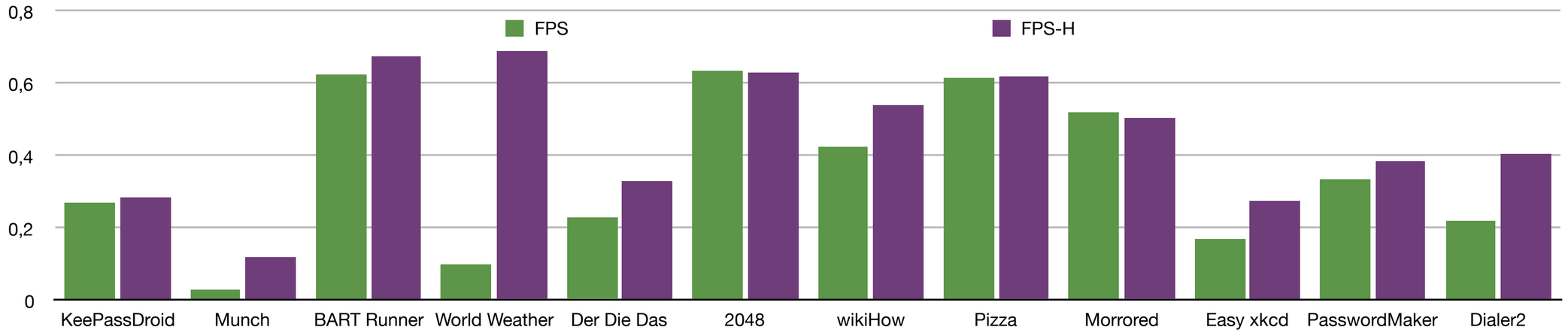
- **Statement coverage**
  - Baseline vs Baseline+RL
  - Roulette wheel selection algorithm from Baseline+RL
  - 10 runs per app

RL improved the baseline by **20%**



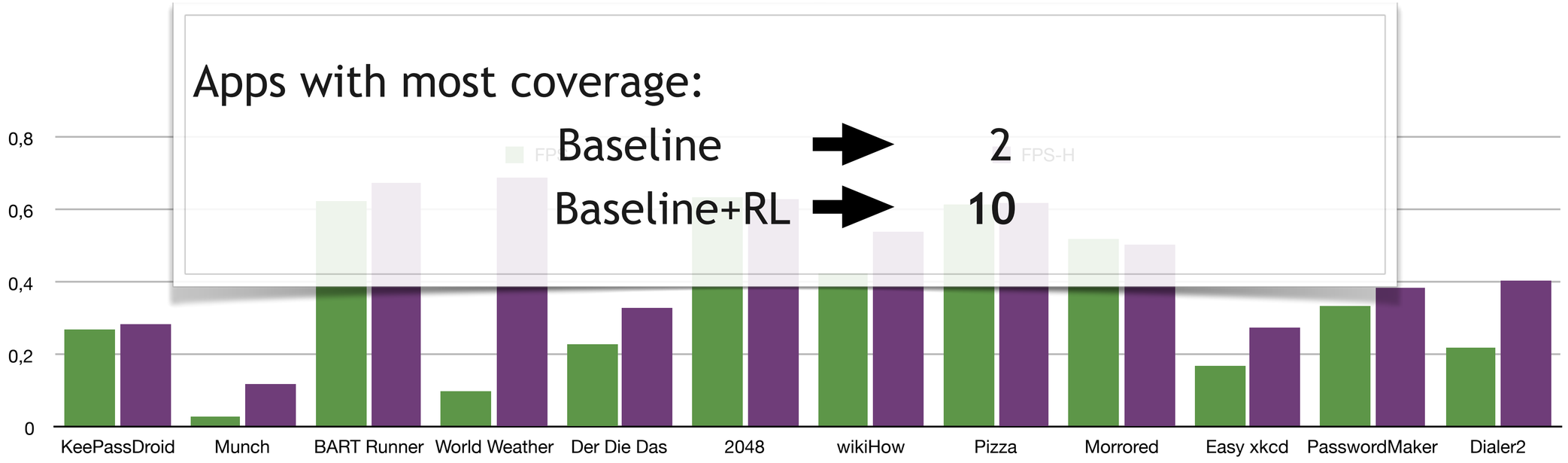
Evaluation — Multi-Armed-Bandit approaches to enhance static models?

- **Statement coverage**
  - Baseline vs Baseline+RL
  - Roulette wheel selection algorithm from Baseline+RL
  - 10 runs per app



Evaluation — Multi-Armed-Bandit approaches to enhance static models?

- **Statement coverage**
  - Baseline vs Baseline+RL
  - Roulette wheel selection algorithm from Baseline+RL
  - 10 runs per app



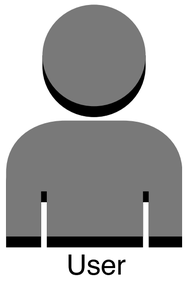
Summary

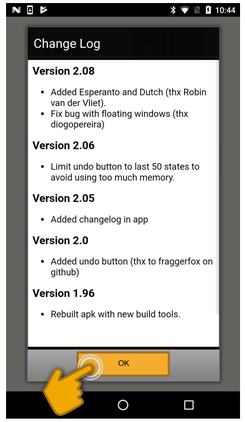
CISPA
HELMHOLTZ-ZENTRUM i. G.

UNIVERSITÄT
DES
SAARLANDES

Summary

User vs GUI

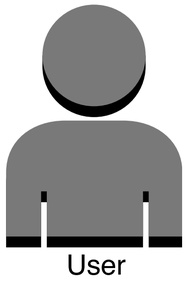


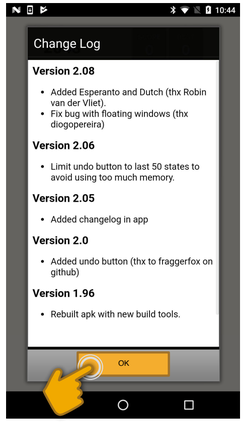


X

Summary

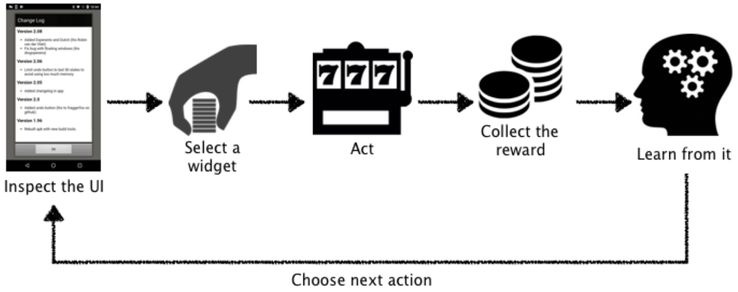
User vs GUI





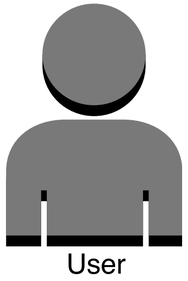
X

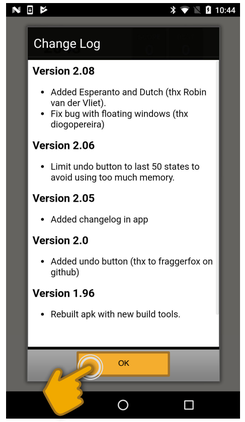
Testing with Reinforcement Learning



Summary

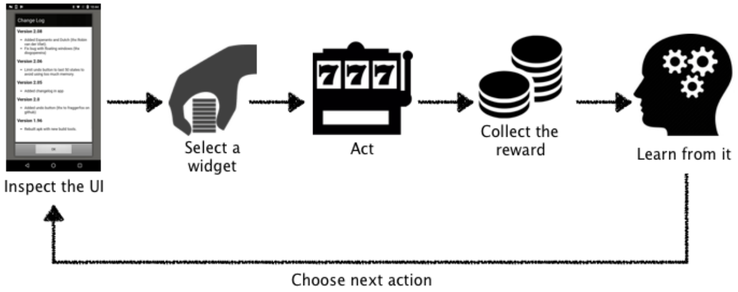
User vs GUI





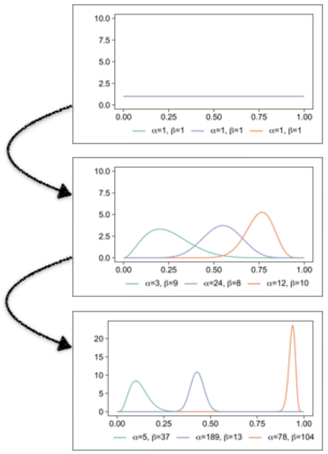
X

Testing with Reinforcement Learning



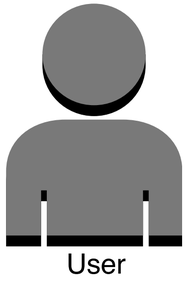
Reinforcement Learning Strategies

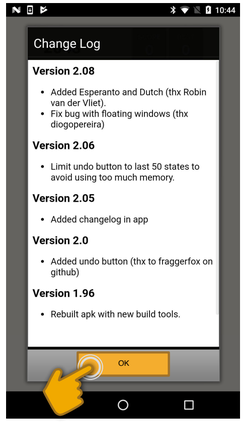
- Thompson Sampling
  - Wins and trials
  - β distribution
  - Random sample
  - Select class with best sample
  - Increase trials and wins after each action
- Distributions adjust over time



Summary

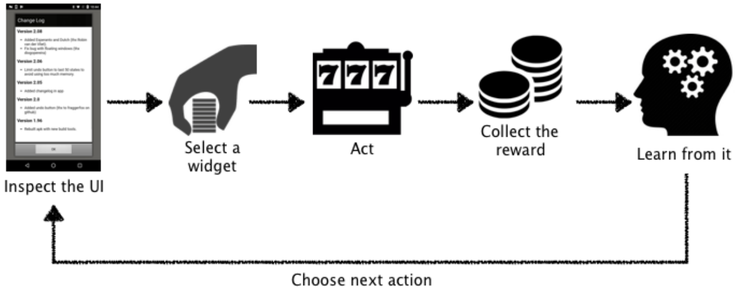
User vs GUI





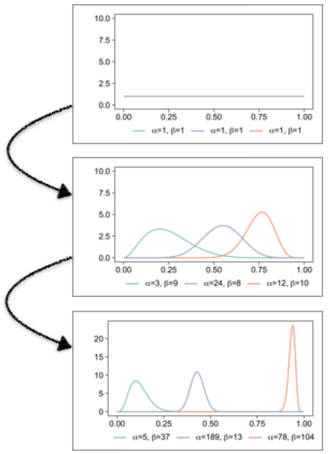
X

Testing with Reinforcement Learning



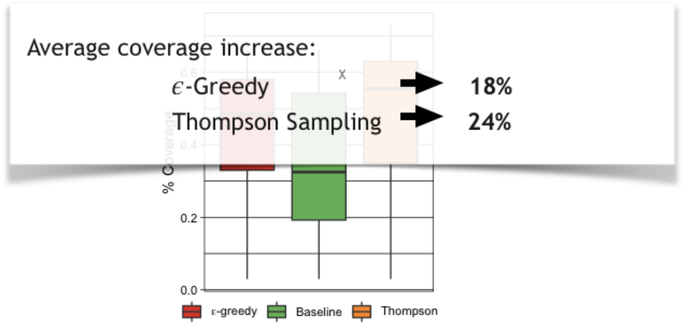
Reinforcement Learning Strategies

- Thompson Sampling
  - Wins and trials
  - β distribution
  - Random sample
  - Select class with best sample
  - Increase trials and wins after each action
- Distributions adjust over time



Evaluation — Can RL effectively test apps?

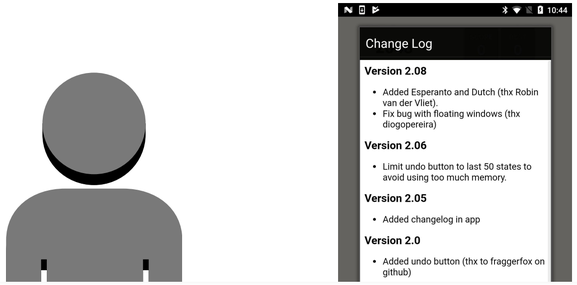
- **Statement coverage**
  - Baseline vs ϵ-Greedy vs Thompson Sampling
  - 10 runs per app



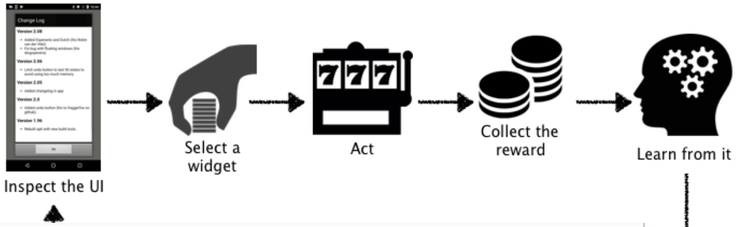
UNIVERSITÄT DES SAARLANDES

Summary

User vs GUI



Testing with Reinforcement Learning



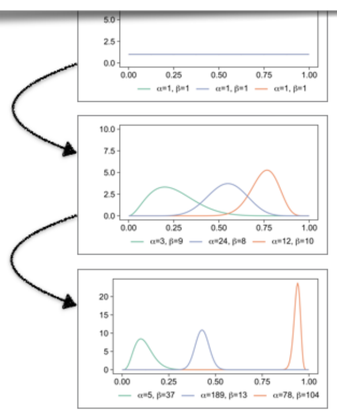
Available on GitHub

https://github.com/uds-se/droidmate-bandits

build passing JitPack 1.0

Reinfo

- Thom
  - Wins and trials
  - β distribution
  - Random sample
  - Select class with best sample
  - Increase trials and wins after each action
- Distributions adjust over time



- Baseline vs ϵ-Greedy vs Thompson Sampling
- 10 runs per app

Average coverage increase:

ϵ-Greedy	18%
Thompson Sampling	24%