Whisker: Automated Testing of Scratch Programs

Marvin Kreis

Chair of Software Engineering II
University of Passau

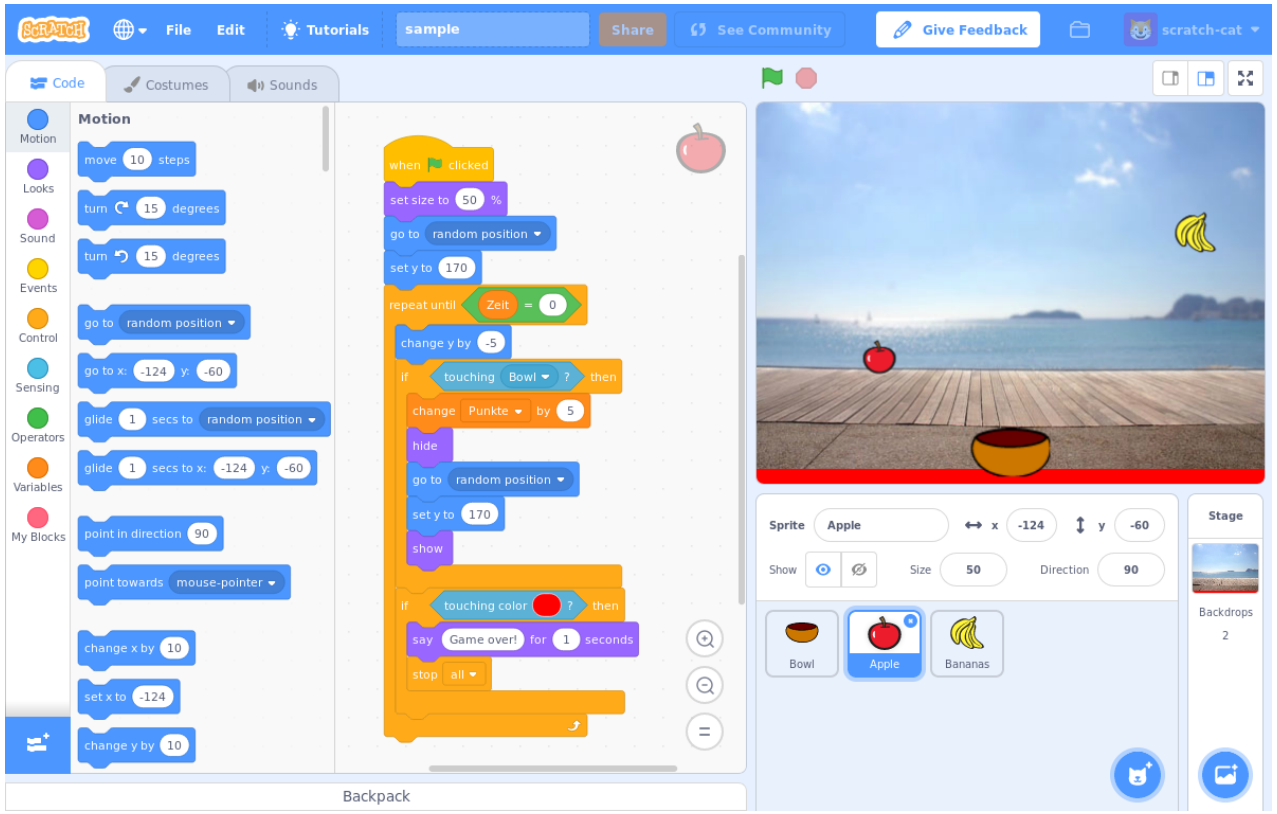
2019-04-02

1. What is Scratch?
2. Why automated testing for Scratch?
3. How to test Scratch programs?
4. Testing Framework: Whisker
5. Evaluation + Results
6. Future Work

What is Scratch?

What is Scratch?

- Block-based programming language
- Programs create interactive animations on a stage
- Code is separated into scripts that are triggered by events
- 🚩 event is the entry point of the program



Why automated testing for Scratch?

Why automated testing for Scratch?

Many schools and universities deploy Scratch as a gentle introduction to programming.

Grading Scratch assignments is very time consuming

- every project has to be opened individually
- programs require large amounts of user interaction

Some courses are attended by a large number of students

- manual testing for grading infeasible
- example University of Utah: > 200 [2]

$\rightarrow$ Automated functional testing

Why automated testing for Scratch?

Many schools and universities deploy Scratch as a gentle introduction to programming.

Grading Scratch assignments is very time consuming

- every project has to be opened individually
- programs require large amounts of user interaction

Some courses are attended by a large number of students

- manual testing for grading infeasible
- example University of Utah: > 200 [2]

$\rightarrow$ Automated functional testing

Why automated testing for Scratch?

Many schools and universities deploy Scratch as a gentle introduction to programming.

Grading Scratch assignments is very time consuming

- every project has to be opened individually
- programs require large amounts of user interaction

Some courses are attended by a large number of students

- manual testing for grading infeasible
- example University of Utah: > 200 [2]

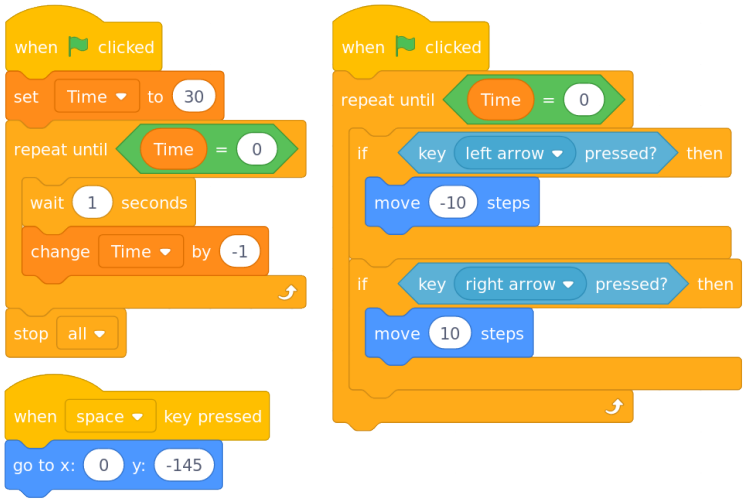
$\rightarrow$ Automated functional testing

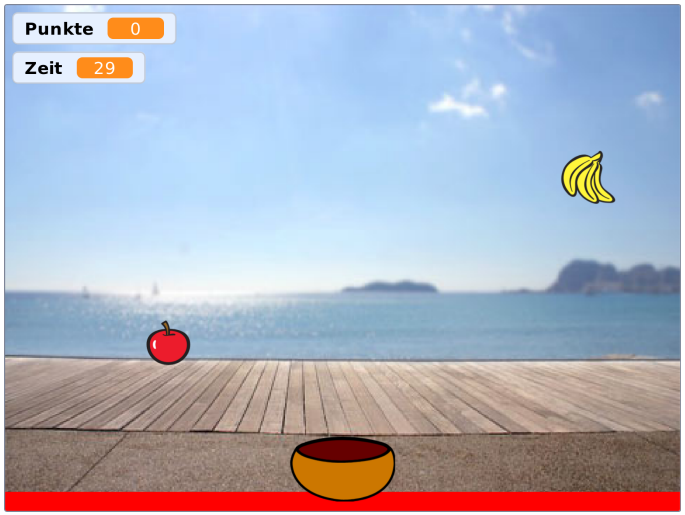
Why is automated testing for Scratch difficult?

Why is automated testing for Scratch difficult?

Scratch is difficult to interact with in an automated way:

- no functions, that take parameters and return a value
- no textual IO, keyboard / mouse input and graphical output



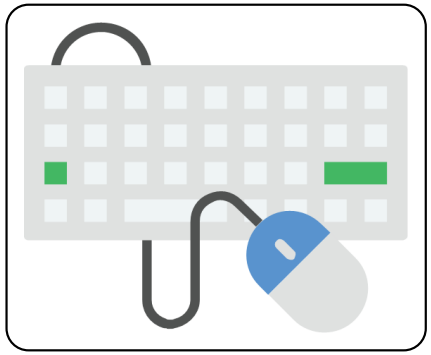


How to test Scratch programs?

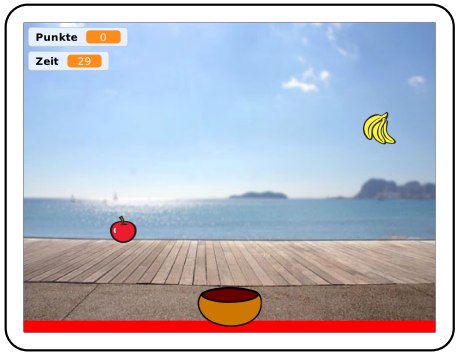
How to test Scratch programs? Automating IO

Approach: Test on a system level by automating Scratch's IO

Input



Output



Input

```
t.inputImmediate({
    device: 'mouse',
    isDown: true,
    x: 50,
    y: 100
});
```

Output

```
sprite.x
sprite.y
sprite.rotation
sprite.sayText
sprite.costume
variable.value
```

Figure: Comparison of Scratch's IO and Whisker's IO

How to test Scratch programs? Automating IO

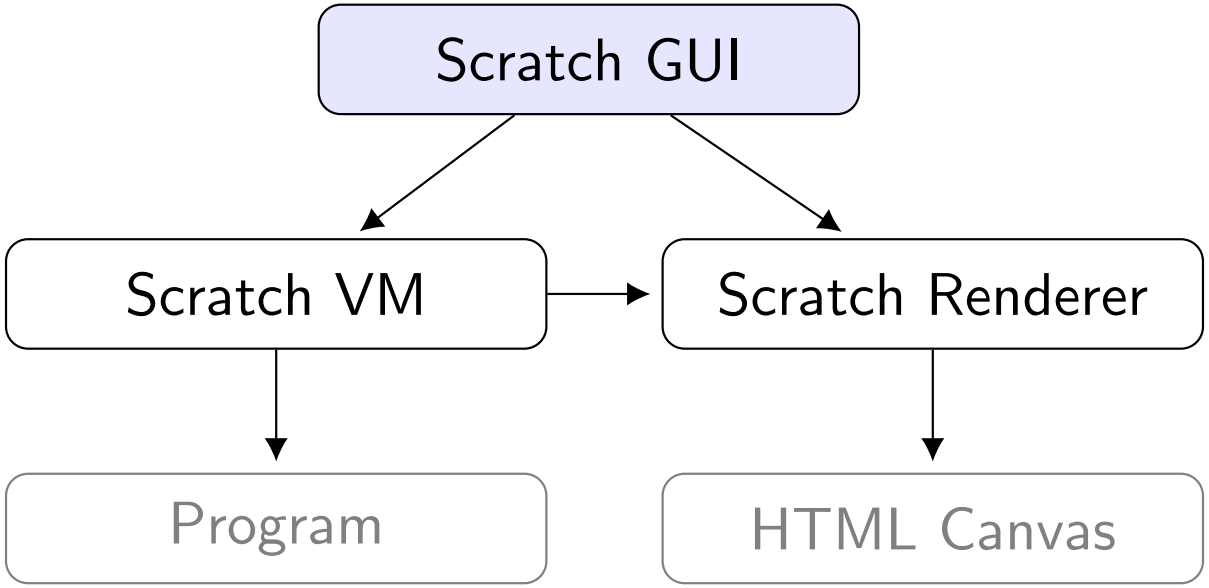


Figure: General architecture of Scratch

How to test Scratch programs? Automating IO

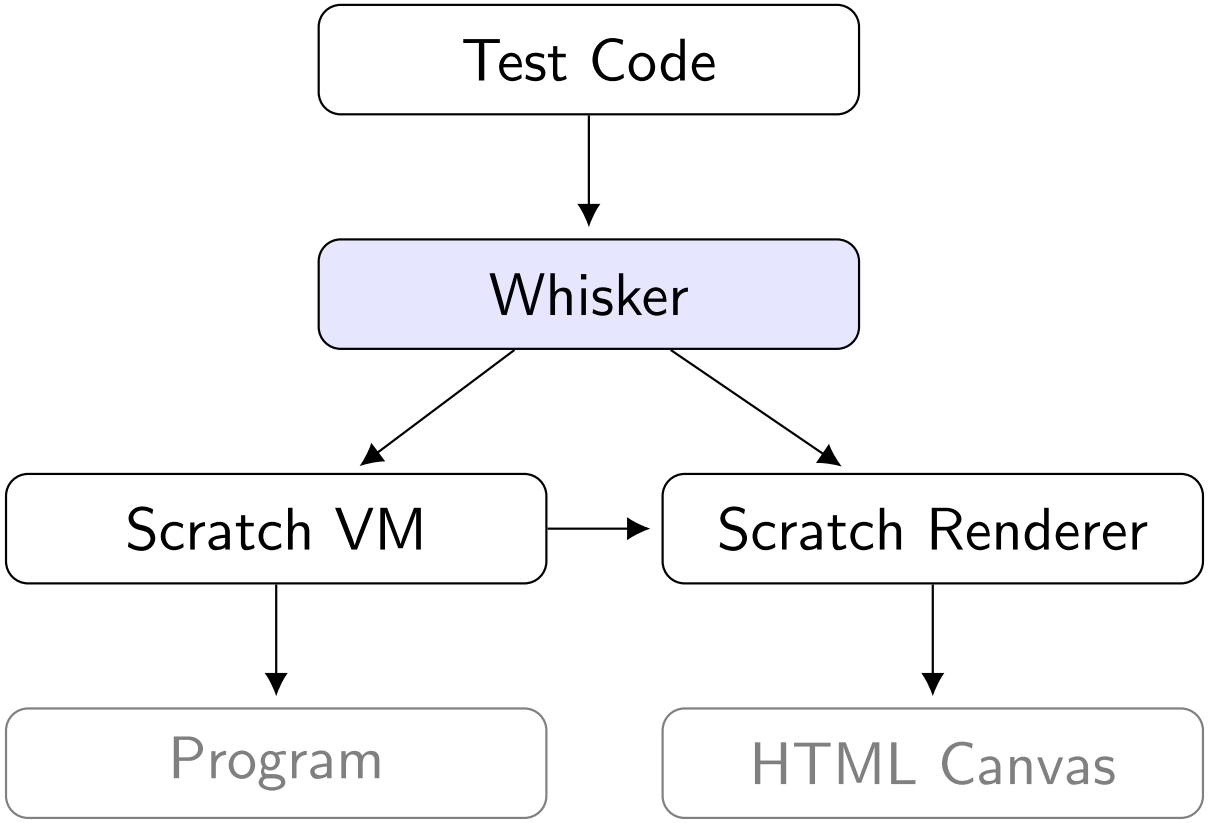
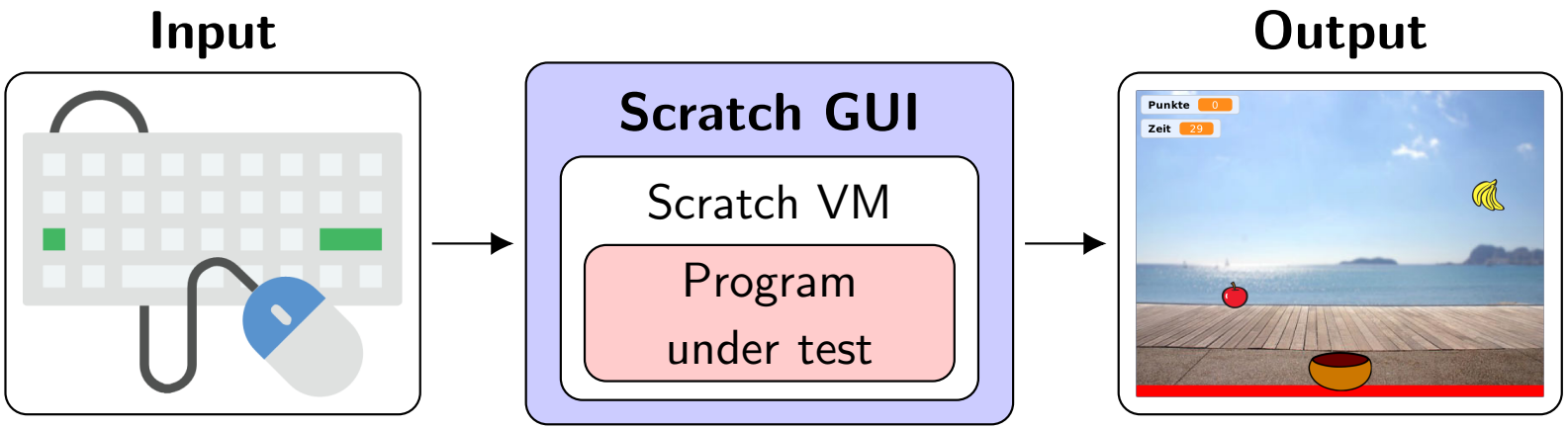


Figure: General architecture of Whisker

How to test Scratch programs? Automating IO



Input

```
t.inputImmediate({
    device: 'mouse',
    isDown: true,
    x: 50,
    y: 100
});
```

Whisker

Scratch VM

Program under test

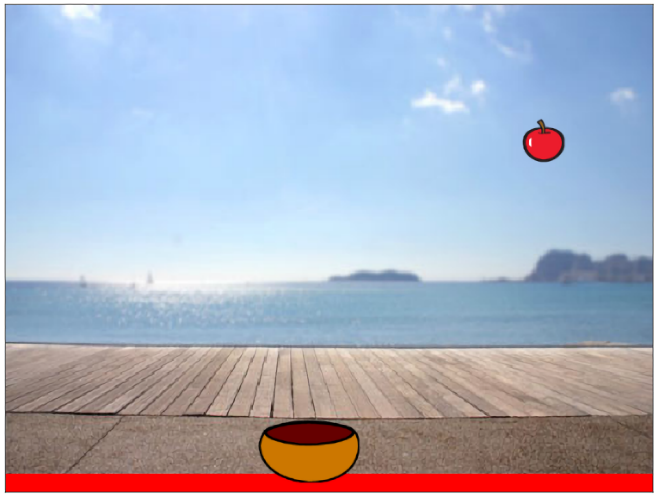
Output

```
sprite.x
sprite.y
sprite.rotation
sprite.sayText
sprite.costume
variable.value
```

Figure: Comparison of Scratch's IO and Whisker's IO

Whisker

Whisker, Whisker's GUI



Scratch Project

sample.sb3 Browse

Tests

tests.js Browse

Scratch Controls

Reset Run All Tests

Enable Input

Search:

	#	Name	Categories	
+	1	Bowl Initialization Test	initialization, bowl	▶
+	2	Bowl Movement Test	bowl movement, bowl, input	▶
+	3	Bowl Movement Details Test	bowl movement, bowl, input	▶

TAP13

```
# project: sample.sb3
TAP version 13
1..3
ok 1 - Bowl Initialization Test
ok 2 - Bowl Movement Test
not ok 3 - Bowl Movement Details Test
  ---
  severity: fail
  error:
    name: AssertionError
    actual: false
    expected: true
    operator: ok
    message: Bowl must move in steps of size 10.
```

Figure: Whisker's GUI

Whisker, Example Test

```
const test = async function (t) {
    await t.runForTime(100);

    const sprite = t.getSprite('Sprite1');
    let oldX = sprite.x;

    await t.runForTime(250);

    t.assert.ok(oldX === sprite.x);

    t.inputImmediate({
        device: 'keyboard',
        key: 'right arrow',
        isDown: true
    });

    await t.runForTime(250);

    t.assert.ok(sprite.x > oldX);
}
```

Whisker, Example Test

```
const test = async function (t) {
    await t.runForTime(100);

    const sprite = t.getSprite('Sprite1');
    let oldX = sprite.x;

    await t.runForTime(

    t.assert.ok(oldX ==

    t.inputImmediate({
        device: 'keyboa
        key: 'right arr
        isDown: true
    });

    await t.runForTime(250);

    t.assert.ok(sprite.x > oldX);
}
```

```
/* Running the program */

await t.runForTime(500);
await t.runUntil(() => a > b, 1000);

t.cancelRun();

t.greenFlag();
```

Whisker, Example Test

```
const test = async function (t) {
    await t.runForTime(100);

    const sprite = t.getSprite('Sprite1');
    let oldX = sprite.x;

    await t.runForTime(250);

    t.assert.ok(oldX === sprite.x);

    t.inputImmediate({
        device: 'keyboard',
        key: 'right arrow',
        isDown: true
    });

    await t.runForTime(250);

    t.assert.ok(sprite.x > oldX);
}
```

Whisker, Example Test

```
const test = async function (t) {
    await t.runForTime(100);

    const sprite = t.get
    let oldX = sprite.x;

    await t.runForTime(2

    t.assert.ok(oldX ===

    t.inputImmediate({
        device: 'keyboar
        key: 'right arro
        isDown: true
    });

    await t.runForTime(2

    t.assert.ok(sprite.x > oldX);
}
```

```
/* Accessing sprites and variables */

t.getSprite('Sprite1');
t.getSprites(sprite => sprite.x > 100);
t.getStage();

sprite.getVariable('my variable');

sprite.x;
sprite.rotation;
sprite.visible;
sprite.sayText;
variable.value;

sprite.isTouchingEdge();
```

Whisker, Example Test

```
const test = async function (t) {
    await t.runForTime(100);

    const sprite = t.getSprite('Sprite1');
    let oldX = sprite.x;

    await t.runForTime(250);

    t.assert.ok(oldX === sprite.x);

    t.inputImmediate({
        device: 'keyboard',
        key: 'right arrow',
        isDown: true
    });

    await t.runForTime(250);

    t.assert.ok(sprite.x > oldX);
}
```

Whisker, Example Test

```
const test = async function (t) {
    await t.runForTime(10

    const sprite = t.getS
    let oldX = sprite.x;

    await t.runForTime(25

    t.assert.ok(oldX ===

    t.inputImmediate({
        device: 'keyboard
        key: 'right arrow
        isDown: true
    });

    await t.runForTime(25

    t.assert.ok(sprite.x
}
```

```
/* Simulating Inputs */

t.inputImmediate({
    device: 'keyboard',
    key: 'space',
    isDown: true,
    duration: 100
});
t.addInput(1000, {
    device: 'mouse',
    x: 100,
    y: 200,
    isDown: true
});

t.getMousePos();
t.isKeyDown('space');
```

Whisker, Example Test

```
const test = async function (t) {
    await t.runForTime(100);

    const sprite = t.getSprite('Sprite1');
    let oldX = sprite.x;

    await t.runForTime(250);

    t.assert.ok(oldX === sprite.x);

    t.inputImmediate({
        device: 'keyboard',
        key: 'right arrow',
        isDown: true
    });

    await t.runForTime(250);

    t.assert.ok(sprite.x > oldX);
}
```

Whisker, Callbacks

- Callbacks get executed every time a frame is rendered
- Make it possible to observe what the user sees

```
t.addCallback(() => someList.push(sprite.x));

const callback = t.addCallback(() => {
    if (sprite.visible) {
        t.inputImmediate({ device: 'mouse', isDown: true });
    } else {
        t.inputImmediate({ device: 'mouse', isDown: false });
    }
});

callback.disable();
callback.enable();
callback.isActive();
```

Whisker, Constraints

- Constraints define conditions that must hold for the program
- Like callbacks, constraints are checked (executed) every time a frame is rendered

```
const constraint = t.addConstraint(() => {
    t.assert.ok(sprite.visible, 'Sprite must be visible.');
});

constraint.disable();
constraint.enable();
constraint.isActive();

t.onConstraintFailure('fail');
t.onConstraintFailure('nothing');
```

Whisker, Automated Input Generation

- At a constant frequency, performs a random input from a pool
- Whisker can detect what inputs the program can react to

```
t.setRandomInputInterval(150);

t.registerRandomInputs([
    { device: 'keyboard', key: 'left arrow', duration: [50, 100] },
    { device: 'keyboard', key: 'right arrow', duration: [50, 100] },
    { device: 'mouse', x: [-50, 50], y: [-100, 100], weight: 0.5 }
]);

t.detectRandomInputs({ duration: [50, 100] });
```

Property-based Testing with Whisker

Property-based Testing with QuickCheck

Example: QuickCheck

- Define properties
- Feed program with (random) input
- Check if the program complies to the defined properties

```
prop_RevApp xs ys =
    reverse (xs++ys) == reverse xs++reverse ys
```

```
Main> quickCheck prop_RevApp
OK: passed 100 tests.
```

```
Main> quickCheck prop_RevApp
Falsifiable, after 1 tests:
[2]
[-2,1]
```

Figure: QuickCheck Usage Example (from [1])

Property-based Testing with QuickCheck

Example: QuickCheck

- Define properties
- Feed program with (random) input
- Check if the program complies to the defined properties

```
prop_RevApp xs ys =
    reverse (xs++ys) == reverse xs++reverse ys
```

```
Main> quickCheck prop_RevApp
OK: passed 100 tests.
```

```
Main> quickCheck prop_RevApp
Falsifiable, after 1 tests:
[2]
[-2,1]
```

Figure: QuickCheck Usage Example (from [1])

Property-based Testing with Whisker

Property-based testing can be performed on Scratch programs:

- Callbacks: Observe the program's output
- Constraints: Define properties, checked in the background
- (Automated Input Generation: Control the program)

Property-based Testing with Whisker

```
await t.runForTime(100)
const sprite = t.getSprite('Sprite1');

let oldX = sprite.x;

t.addCallback(() => {
    oldX = sprite.x;
});

t.addConstraint(() => {
    if (t.isKeyDown('right arrow')) {
        t.assert.ok(sprite.x > oldX);
    } else {
        t.assert.ok(sprite.x === oldX);
    }
});

t.detectRandomInputs();
await t.runForTime(2000);
```

Property-based Testing with Whisker

Advantages over normal tests:

- Many constraints can be checked in few program executions
- Constraints require little code

Evaluation

RQ: Can test results match results of manual grading?

Evaluation, Test Results

Test subjects:

- 37 student implementations of a simple catching game
- from a 6th and 7th grade Scratch workshop [3]
- Graded manually, on a scale from 0 to 30

Two test suites:

- "normal" test suite with 28 test cases
  - each test case executes the program once, independently
- "constraint" test suite with 26 constraints
  - only one test case
  - uses generated input
  - runs the program for 10s with 30 resets $\rightarrow$ 300s

Measured item:

- Correlation between manual scores and the number of test or constraint passes

Evaluation, Test Results

Test subjects:

- 37 student implementations of a simple catching game
- from a 6th and 7th grade Scratch workshop [3]
- Graded manually, on a scale from 0 to 30

Two test suites:

- "normal" test suite with 28 test cases
  - each test case executes the program once, independently
- "constraint" test suite with 26 constraints
  - only one test case
  - uses generated input
  - runs the program for 10s with 30 resets $\rightarrow$ 300s

Measured item:

- Correlation between manual scores and the number of test or constraint passes

Evaluation, Test Results

Test subjects:

- 37 student implementations of a simple catching game
- from a 6th and 7th grade Scratch workshop [3]
- Graded manually, on a scale from 0 to 30

Two test suites:

- "normal" test suite with 28 test cases
  - each test case executes the program once, independently
- "constraint" test suite with 26 constraints
  - only one test case
  - uses generated input
  - runs the program for 10s with 30 resets $\rightarrow$ 300s

Measured item:

- Correlation between manual scores and the number of test or constraint passes

Evaluation, Test Results

Excluded Projects:

- 6 of the 37 projects were excluded from the calculation
- Most because they don't start with the event, but with other key presses
  - Tests try to start the program with the button
  - Not an issue for manual grading

Evaluation, Test Results, Normal Tests

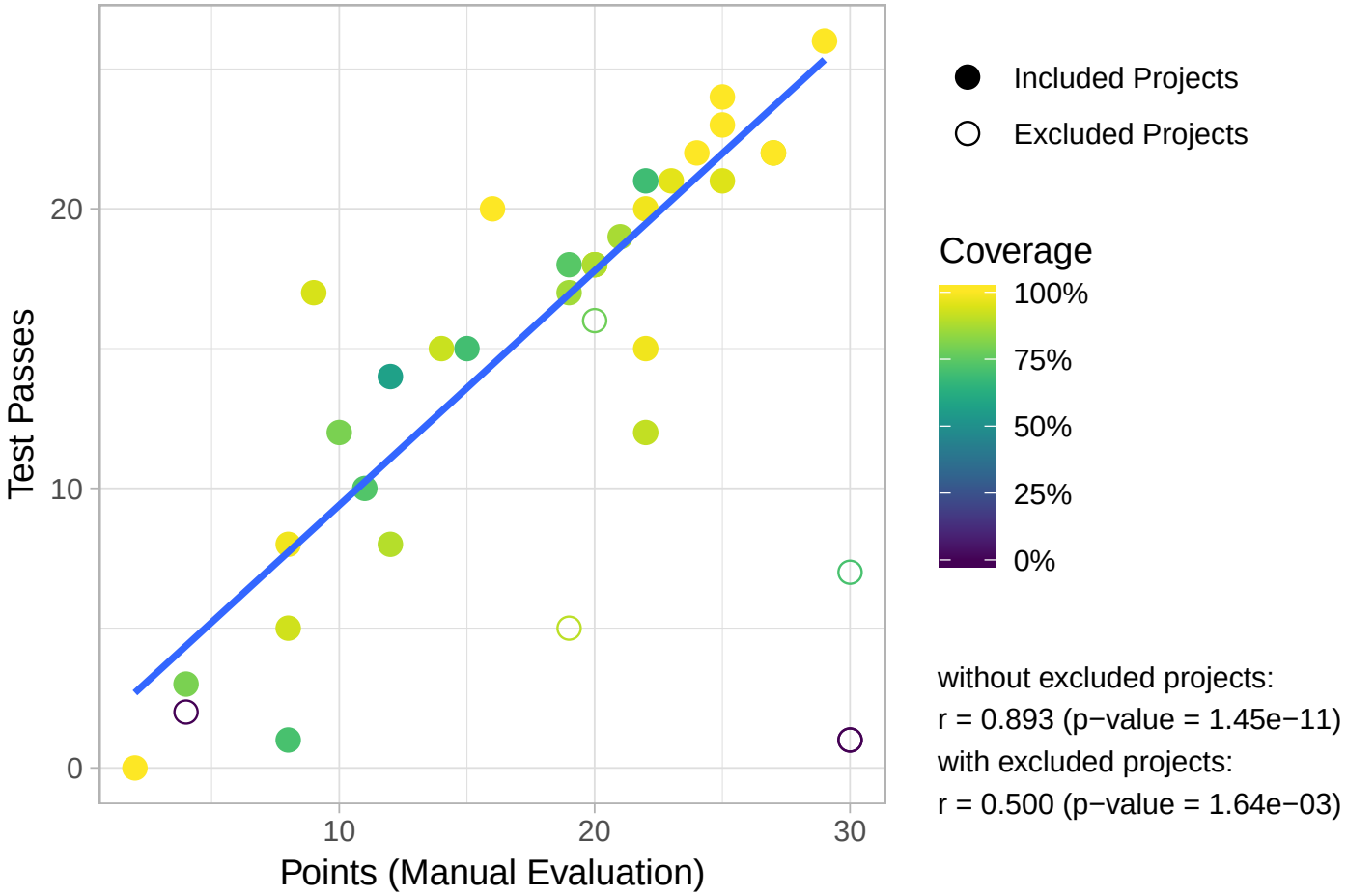


Figure: Comparison between results of normal tests and manual scores, 1st run

Evaluation, Test Results, Normal Tests

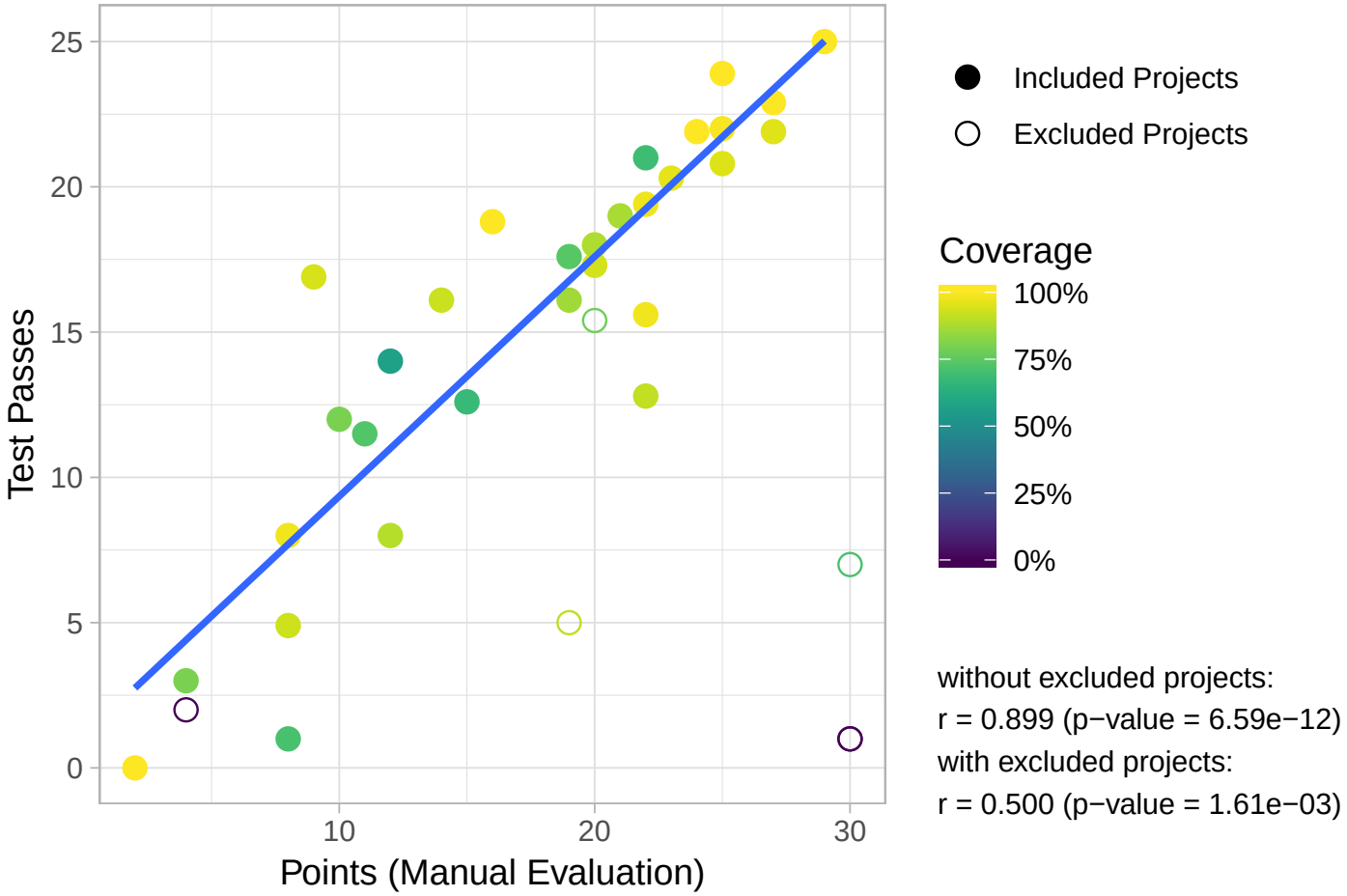


Figure: Comparison between results of normal tests and manual scores, average over 10 runs

Evaluation, Test Results, Constraint Tests

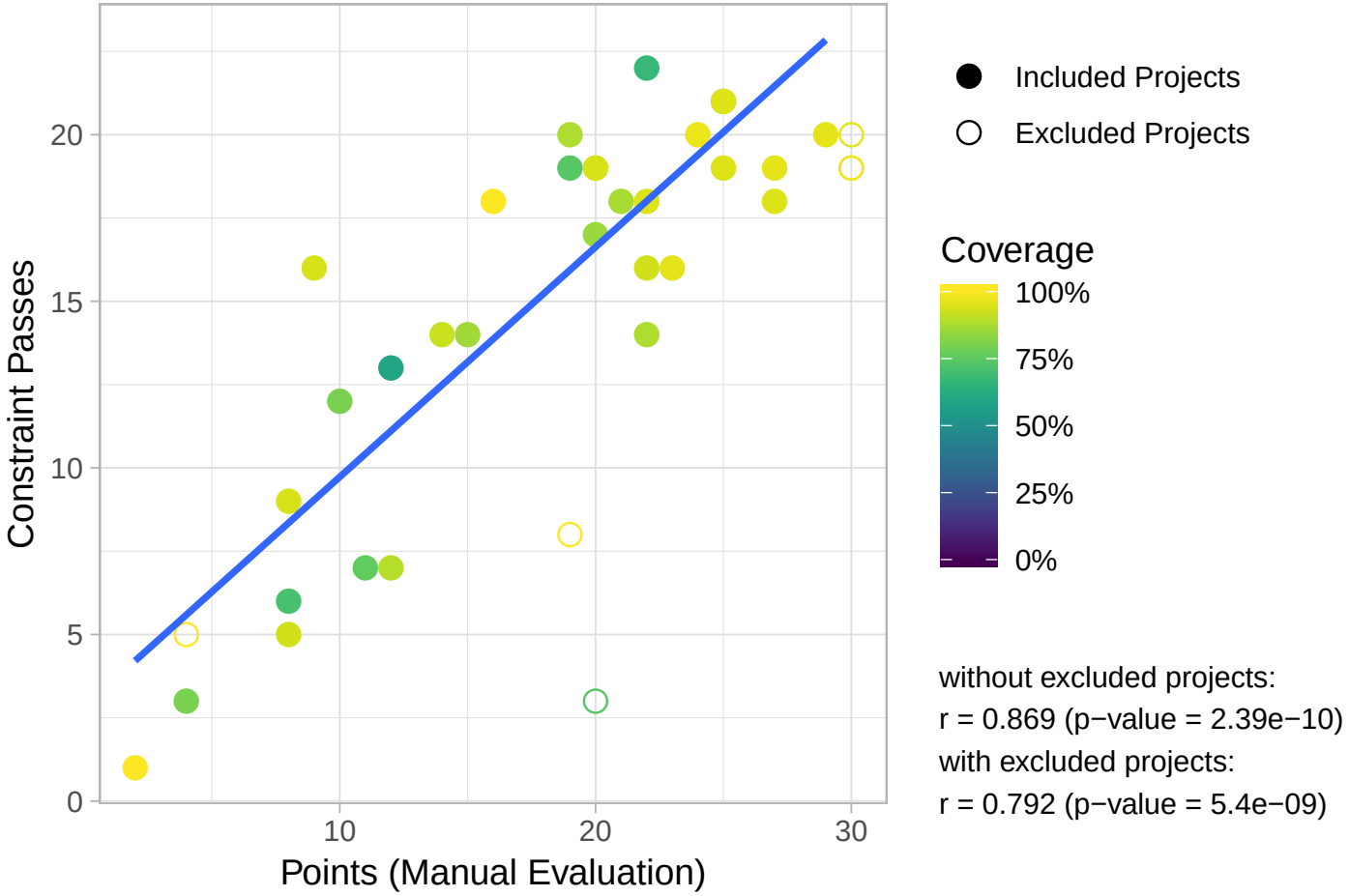


Figure: Comparison between results of constraint tests and manual scores, 1st run

Evaluation, Test Results, Constraint Tests

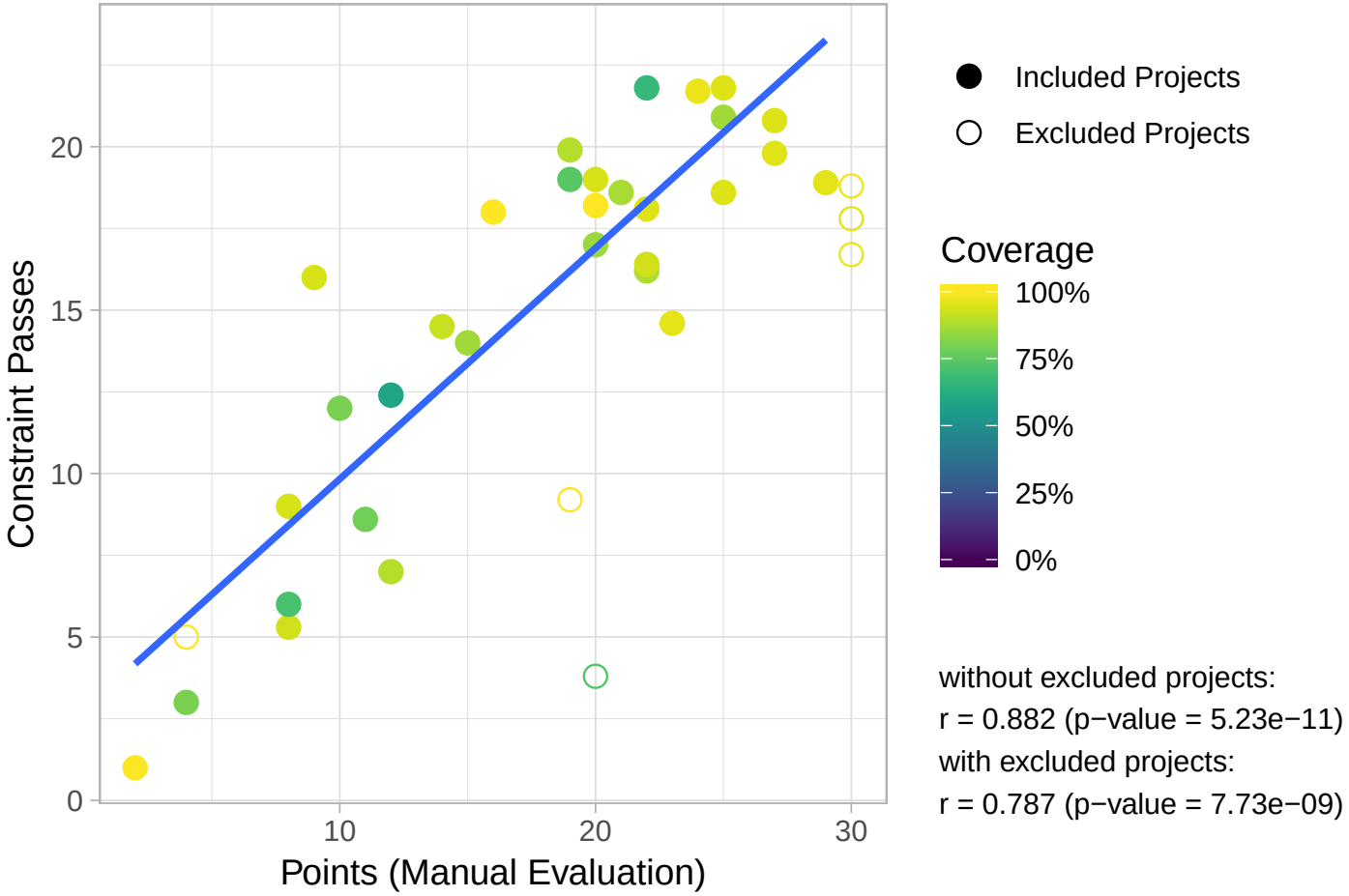


Figure: Comparison between results of constraint tests and manual scores, average over 10 runs

RQ: What coverage can be achieved with automated input?

Evaluation, Coverage of Automated Input

Test subjects:

- 24 sample solutions to Code Club's[1] online Scratch courses
- Run with generated input for 10 minutes

Measured item:

- Mean coverage of the projects after 10 minutes
- Coverage measured every second

[1] `https://codeclubprojects.org/`

Evaluation, Coverage of Automated Input

Test subjects:

- 24 sample solutions to Code Club's[1] online Scratch courses
- Run with generated input for 10 minutes

Measured item:

- Mean coverage of the projects after 10 minutes
- Coverage measured every second

[1] `https://codeclubprojects.org/`

Evaluation, Coverage of Automated Input

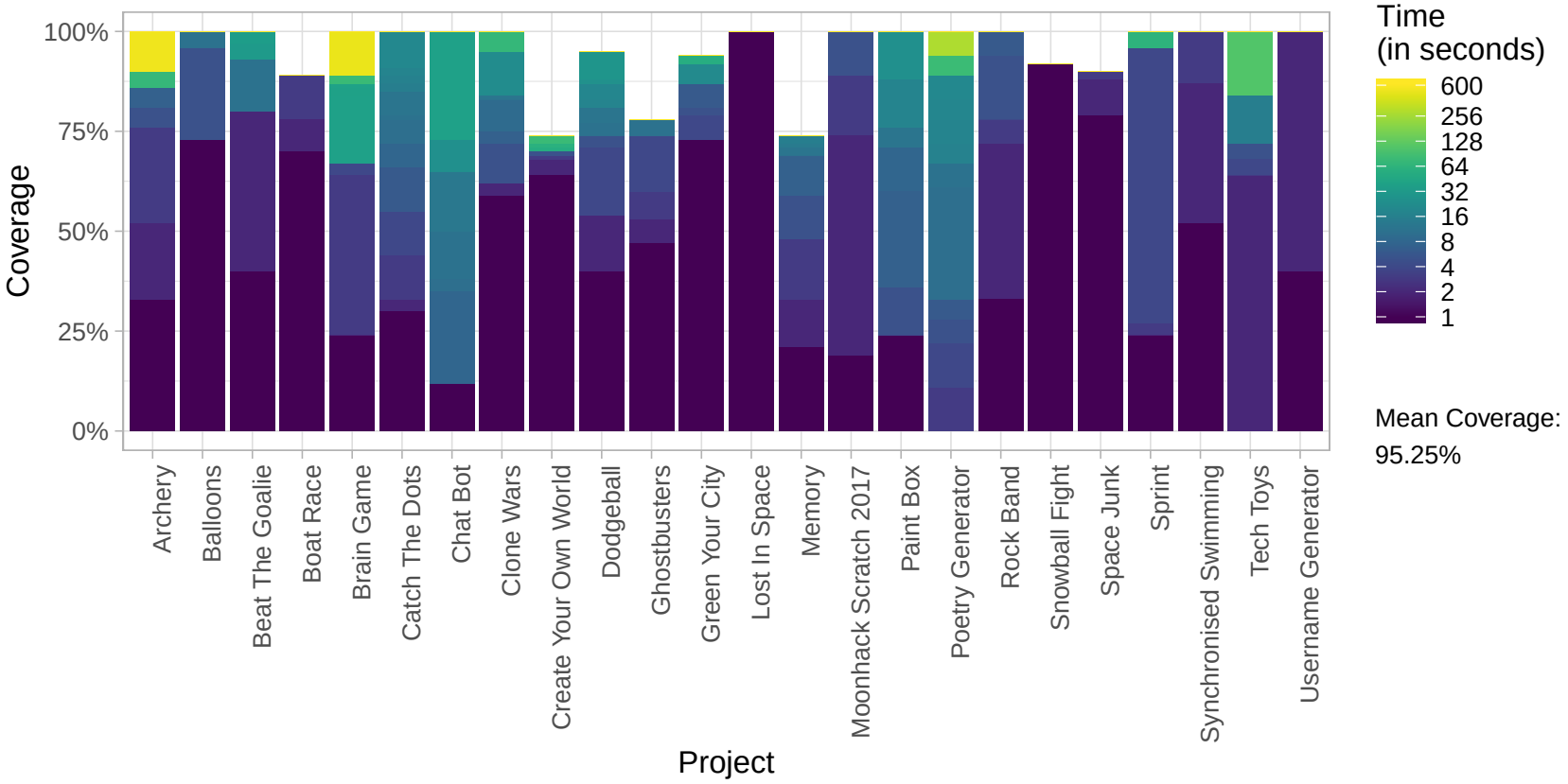


Figure: Coverage per project, 1st run

Evaluation, Coverage of Automated Input

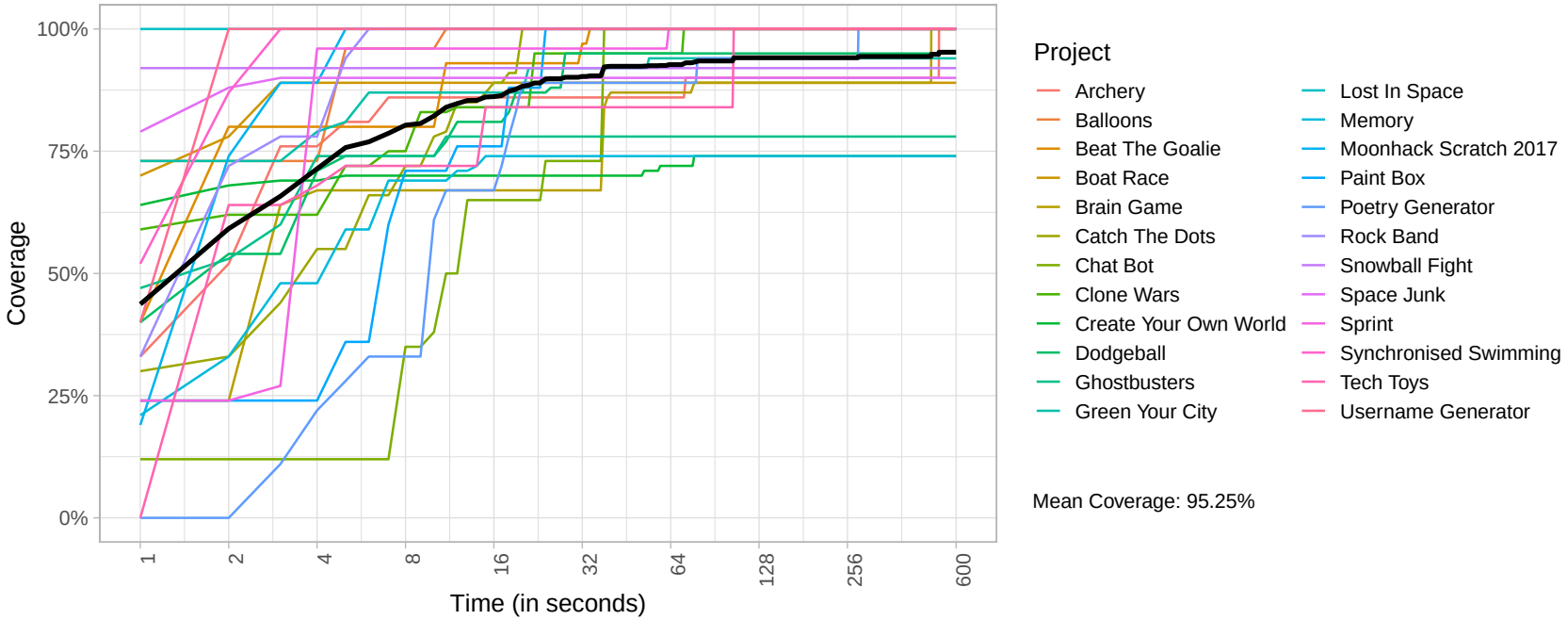


Figure: Coverage over time, 1st run

Evaluation, Coverage of Automated Input

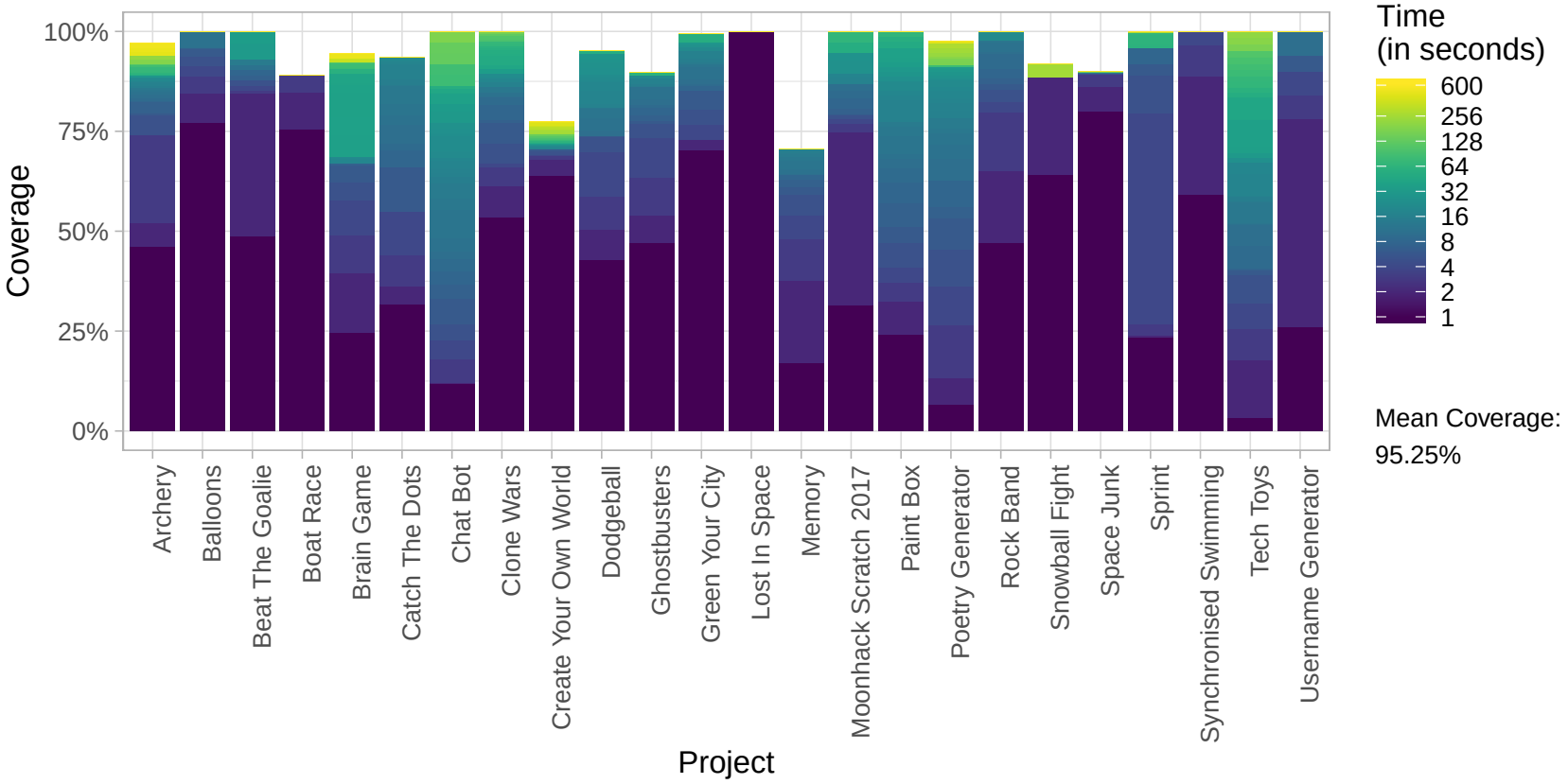


Figure: Coverage per project, average over 10 runs

Evaluation, Coverage of Automated Input

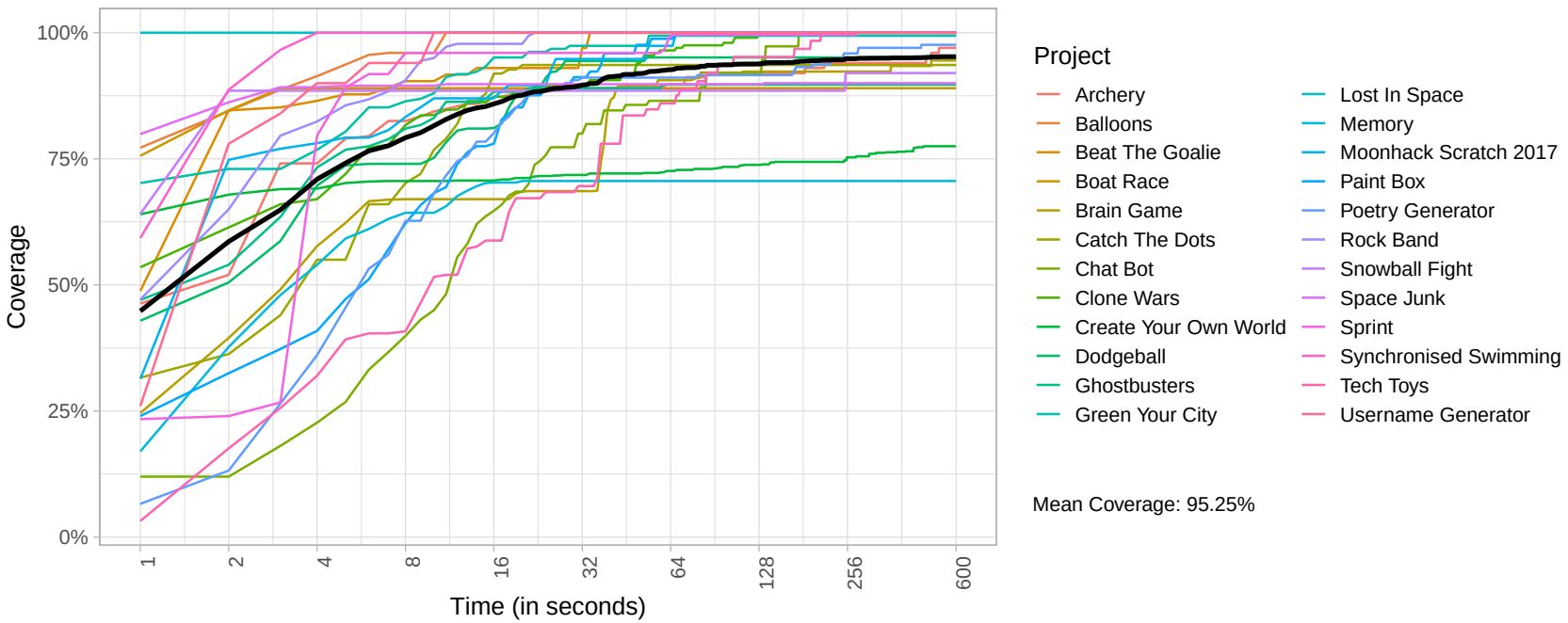


Figure: Coverage over time, average over 10 runs

Evaluation, Coverage of Automated Input

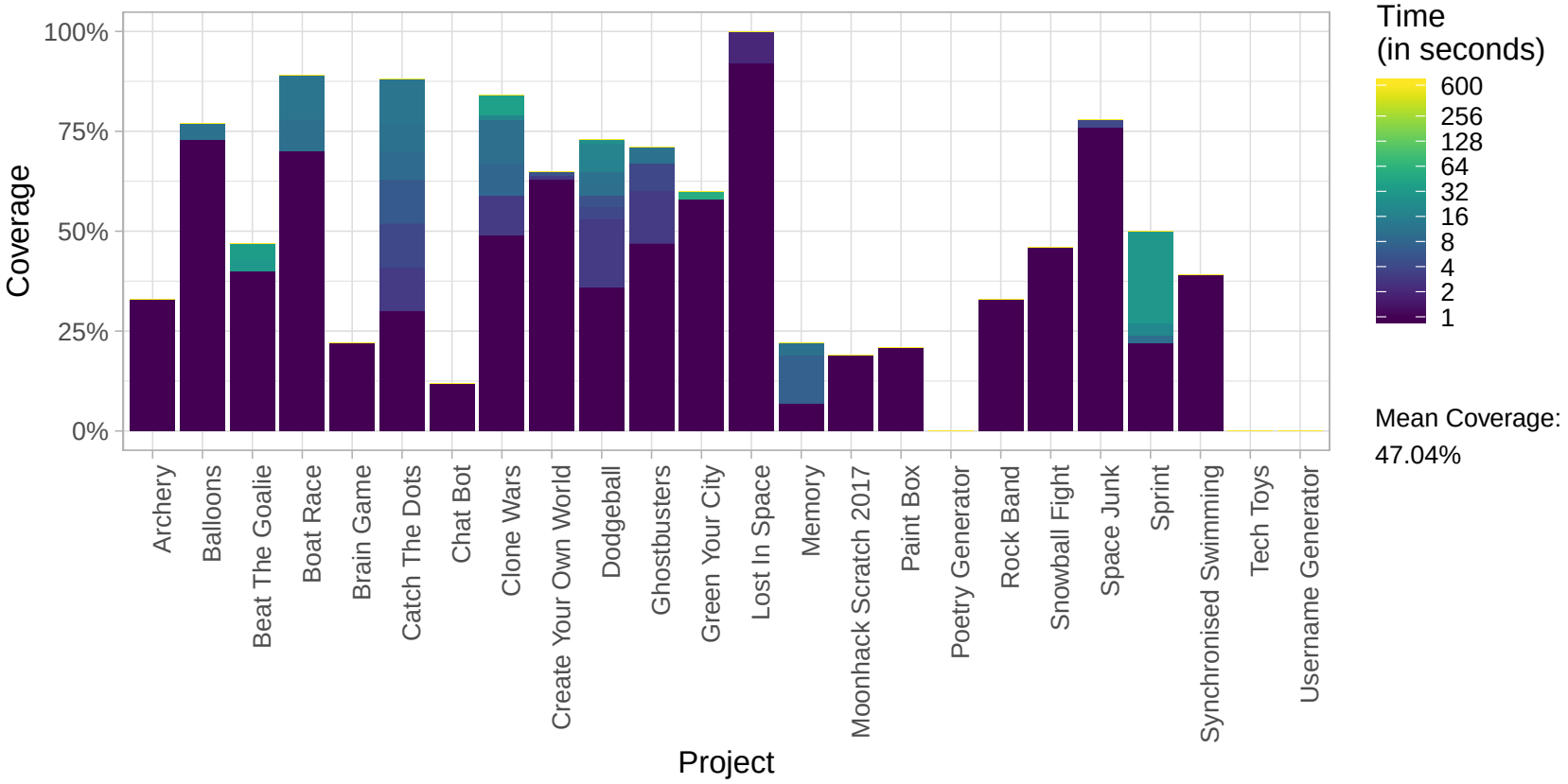


Figure: Coverage per project, 1st run

Evaluation, Coverage of Automated Input

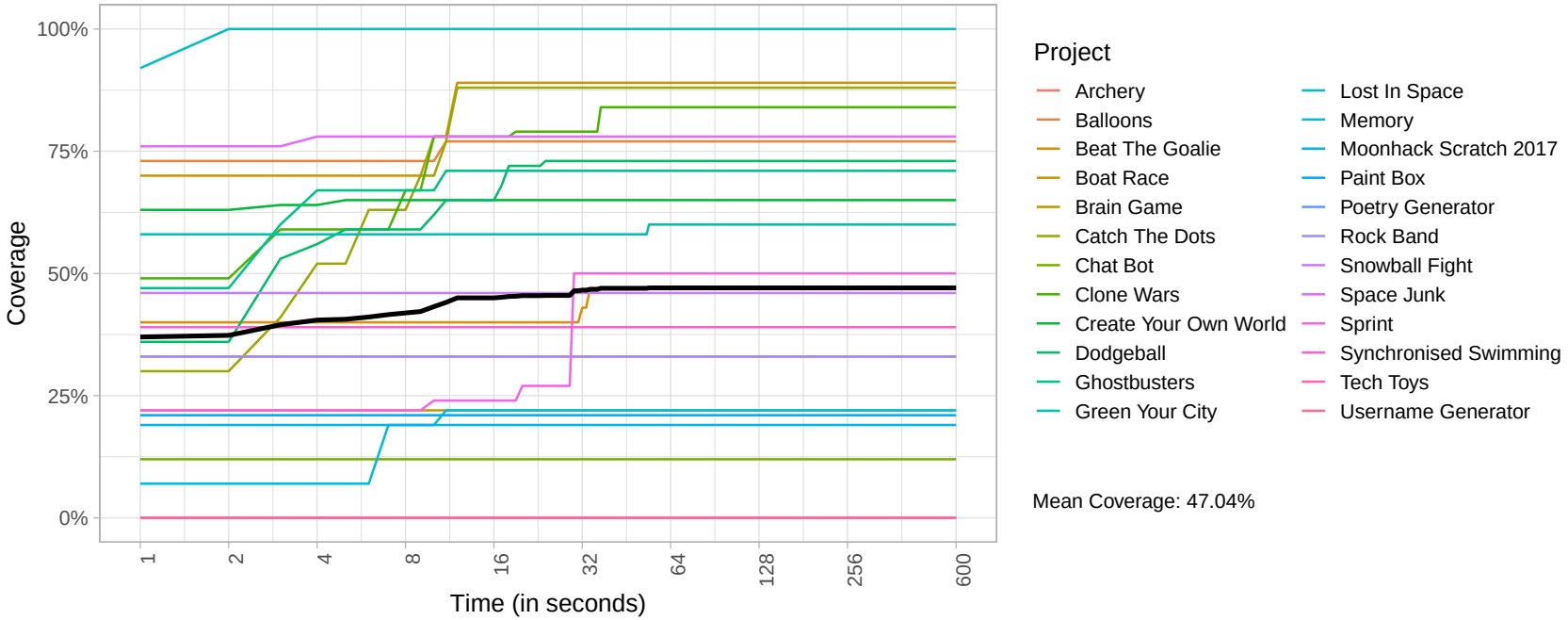


Figure: Coverage over time, 1st run

Evaluation, Coverage of Automated Input

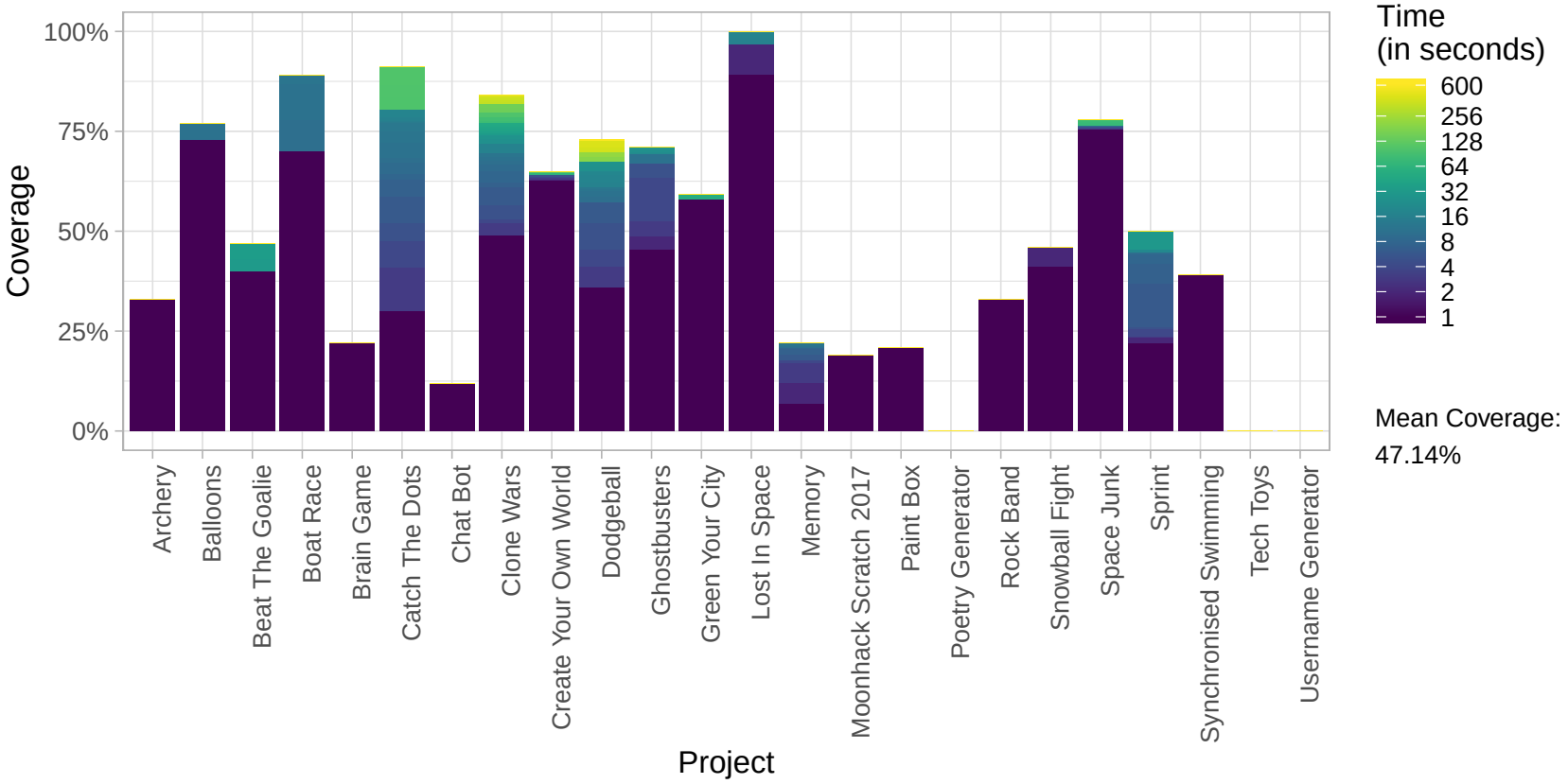


Figure: Coverage per project, average over 10 runs

Evaluation, Coverage of Automated Input

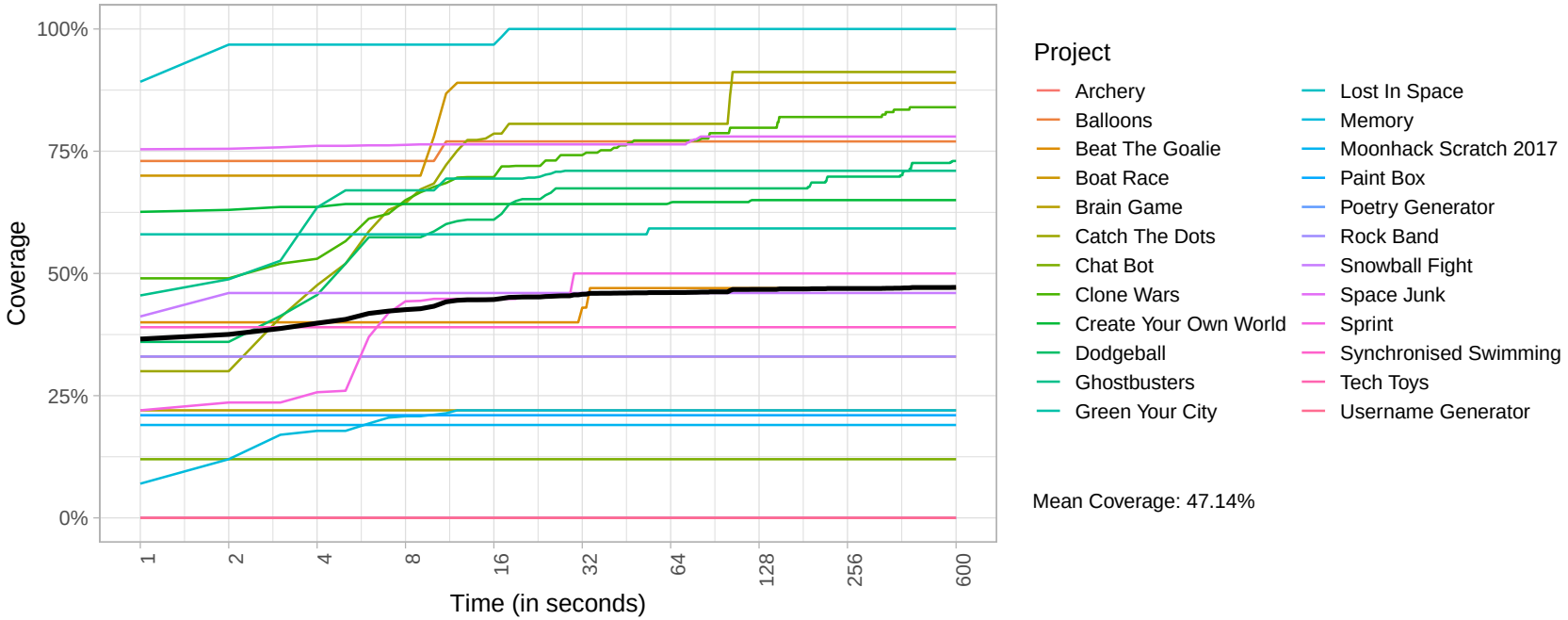


Figure: Coverage over time, average over 10 runs

Future Work

Future Work

- Standalone Electron-UI in addition to the web interface
  - Loading and saving to disk requires user interaction in web UI
- Support for audio and Scratch extensions
  - Currently only graphical output accessible
- Seeding randomness

Thank you for listening!

References

Claessen, K., and Hughes, J.
Quickcheck: A lightweight tool for random testing of haskell programs.
Proceedings of the ACM SIGPLAN International Conference on Functional Programming, ICFP 46 (01 2000).

Johnson, D. E.
Itch: Individual testing of computer homework for scratch assignments.
In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (New York, NY, USA, 2016), SIGCSE '16, ACM, pp. 223–227.

Keller, S.
Using plan-driven development in educational programming projects, 2018.