The Alexa Files

Genre: SMT & Modeling

By Lilymoon, Jessica,
and Audrey

Outline

01.
Overview of modeling project

02.
River Crossing Problem

03.
Liars vs Truth Tellers

04.
Age Riddle

05.
Mislabeled Boxes

06.
Additional Puzzles + Failed Attempts

1. Overview

Foundation

- ★ River Crossing Puzzles ⏖□⏖☼★.˚
    - 🔧 Modeled using Alloy
    - 🔗 River Crossing Puzzle – Wikipedia
- ★ Lying and Truth-Teller Puzzles ⋆˚࿔□⚔️□࿔˚⋆
    - 🔧 Solved using Z3
    - 🔗 Knights and Knaves – Wikipedia
- ★ Age Riddles (❀ˆᴗˆ)(•́ᴗ•̀✿)
    - 🔧 Solved using Z3
    - 🔗 Ages of Three Children Puzzle – Wikipedia
- ★ Mislabeled Boxes ˙ ☆°. 🍎 ⋆。˚□
    - 🔧 Solved using Z3
    - 🔗 Mislabeled Boxes - Mind Your Decisions.

Target

- ★ Generate random variations of word puzzles
- ★ Website component - hard code puzzles

Reach

- A cohesive storyline between stages
- transitions from one puzzle to another
- connect pipeline from Z3 to html/js so we have random iterations of puzzle
- Add additional logic puzzles

Why Logic Puzzles?

- ★ We model word puzzles using formal verification tools like Z3 and Alloy
- ★ Rules + constraints easily transformed into logical formulas +| relational structures
- ★ Word games have defined rules, logical dependencies, and hidden relationships

2.River Crossing

Waban Crossing Problem

A student with a fox, goose, and lettuce must cross Lake Waban by boat. The boat can carry only the student and a single item. If left unattended together, the fox would eat the goose, or the goose would eat the lettuce. How can they cross the river without anything being eaten

Variables + Setup

{River Crossing Problem in Alloy}

```
--Variable declarations ---
abstract sig Item{}
one sig Farmer, Wolf, Goat, Cabbage extends Item{}
var sig ThisSide in Item{}
var sig OtherSide in Item{}

---the initial setup of everything---
pred init(ts: ThisSide, os: OtherSide){
    no os
    ts = Item
}
fact eventuallyAllOtherSide{
    eventually OtherSide = Item
}
```

Actions

{River Crossing Problem in Alloy}

1. pred doNothing{}
2. pred crossThisToOther(i: Item){

```
//preconditions
Farmer in ThisSide
i in ThisSide

//actions being done
ThisSide' = ThisSide - Farmer - i
OtherSide' = OtherSide + Farmer + i
}
```

3. pred crossOtherToThis(i: Item){almost identical code as above...}

Rules

{River Crossing Problem in Alloy}

- Constraints modeled as "fact rules{}"

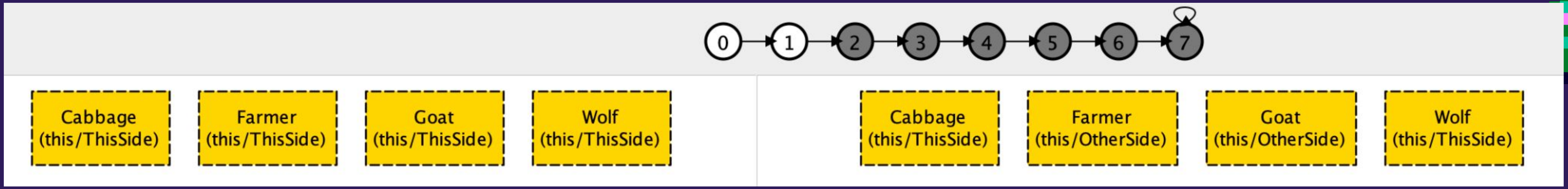
```
// 1. Goat  can't be alone with cabbage
always{
        (Farmer in ThisSide and Goat in OtherSide) => not(Cabbage in OtherSide)
        (Farmer in OtherSide and Goat in ThisSide) => not(Cabbage in ThisSide)
}
// 2. Goat  can't be alone with wolf
always{
        (Farmer in ThisSide and Goat in OtherSide) => not(Wolf in OtherSide)
        (Farmer in OtherSide and Goat in ThisSide) => not(Wolf in ThisSide)
}
```

How to make sure progress is made?

- Valid traces!

// force progress something or nothing crosses but farmer has to move back and forth
(OtherSide != Item) =>(crossOtherToThis[Goat] or crossThisToOther[none] or crossThisToOther[Goat] or crossOtherToThis[none] or crossThisToOther[Wolf] or crossOtherToThis[Wolf] or crossThisToOther[Cabbage] or crossOtherToThis[Cabbage])

// Can only do nothing when everyone is on the other side
(OtherSide = Item) => doNothing



Now it is your turn…

3. Liars vs. Truth

Liars vs Truth Tellers

{Knights vs Knaves Problem}

A very special island is inhabited only by knights and knaves. Knights always tell the truth, and knaves always lie. You meet two inhabitants: Alice and Bob. Bob says, “we are both knaves.” Can you determine who’s a knight and whose a knave?

Model in Z3

{Knights vs Knaves Problem}

```
a = Bool("a") # Alice
b = Bool("b") # Bob

s = Solver()

# Constraints based off Bob's Statement
s.add(Implies(b, And(Not(b), Not(a))))
s.add(Implies(Not(b), Or(b, a)))
```

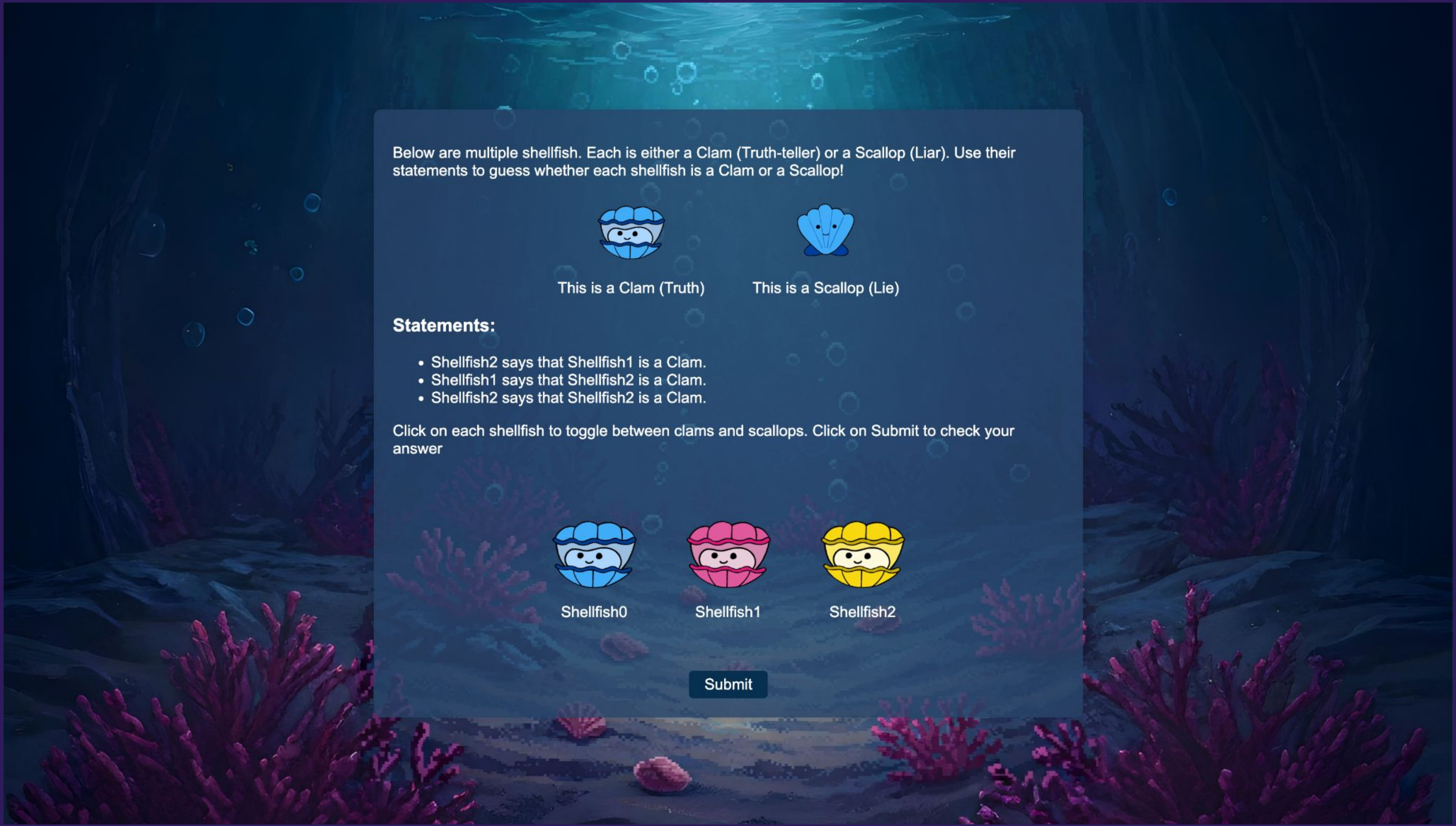
Random Variations

(Knights vs Knaves Problem)

```
generate_instances(n):
    s = Solver()
    for i in range(n):
      person1 = random.choice(bools)
      person2 = random.choice(bools)
# True implies person 1 says that person 2 is a
scallop (lie)
# False implies person 1 says that person 2 is a clam
(truth)
says_is_knave = random.choice([True, False])
# Then adds implications based on bool
# Keep generating instances if not valid
```

Let's Play

{Knights and Knaves}



4.Ages of 3 Children

Ages of Three Children puzzle

{Census Taker Problem}

A woman leaning on her gate, number 13, and asks about her children. She says, "I have three children and the product of their ages is thirty-six. The sum of their ages is the number on this gate.

What are the three ages

Modeling in z3

{Census Taker Problem}

```
age1 = Int('age1')
age2 = Int('age2')
age3 = Int('age3')

solver = Solver()

solver.add(age1 * age2 * age3 == 36)
solver.add(age1 + age2 + age3 == 13)
```

Random Variations

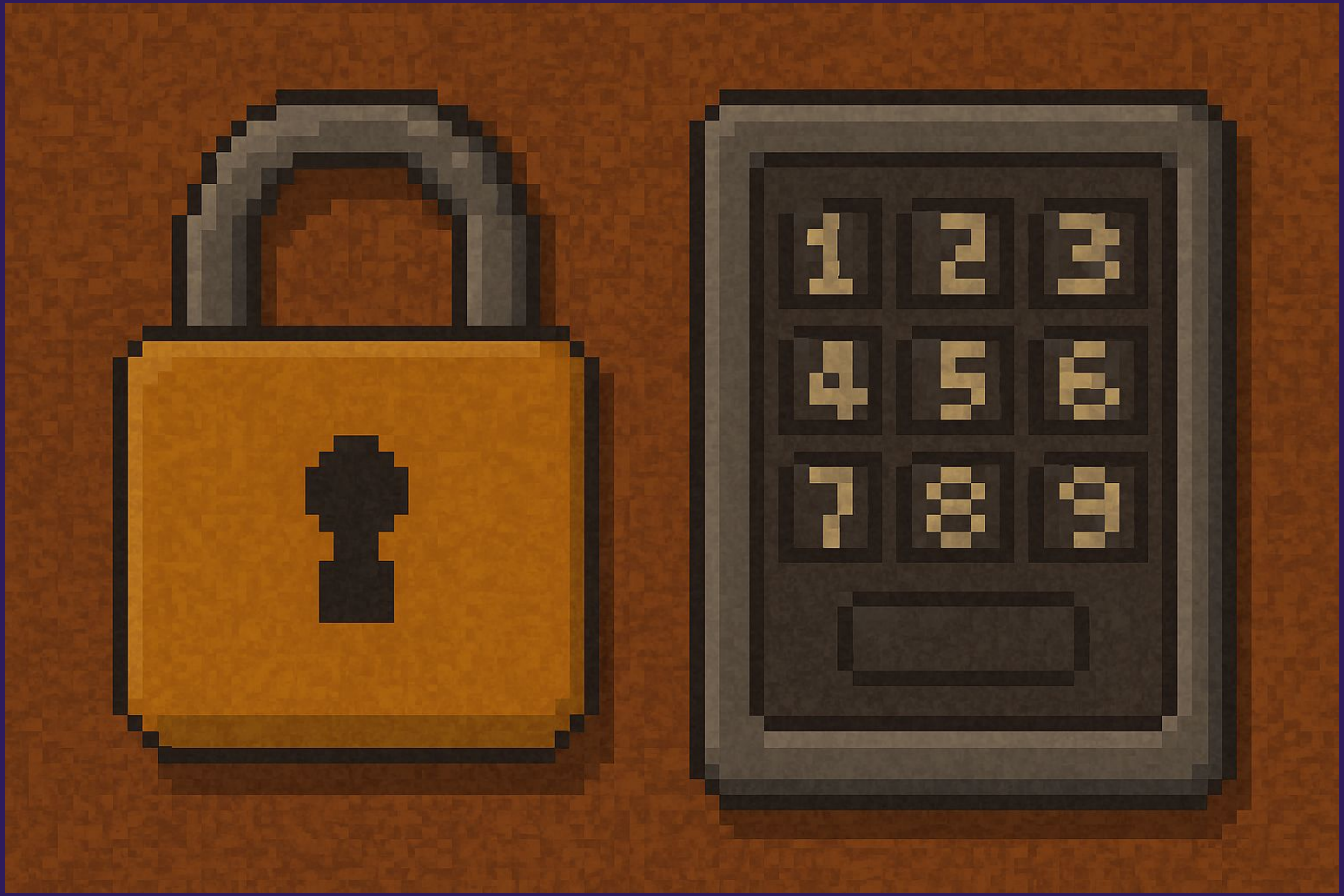
{Census Taker Problem}

```
For in range 100
    random_product = random.randint(1, 100)
      random_sum = random.randint(3, 50)
```

Now it is your turn…

Let's Play

{Census Taker Problem}



8.Mislabled Boxes

Mislabeled Boxes

Currently the first box has the label "apples," the second "oranges," and the third "apples and oranges." Unfortunately all of the labels are wrong. Your job is to fix the labels.

Model

```
contents = ['Apples', 'Oranges', 'Mixed']

box1 = Int('box1')
box2 = Int('box2')
box3 = Int('box3')
```

Constraints

```
s.add(Distinct(box1, box2, box3))
```

Boxes need to be one of the three labels

Generate a random permutation of the labels

```
s.add(box1 != random_labels[0])
s.add(box2 != random_labels[1])
s.add(box3 != random_labels[2])
```

Let's Play

Website Design

Tech Stack: We used Flask to connect 23 and Python scripts with our HTML, CSS, and JavaScript frontend, enabling interactive logic puzzles within the game. We used generative AI to help with the website portion.

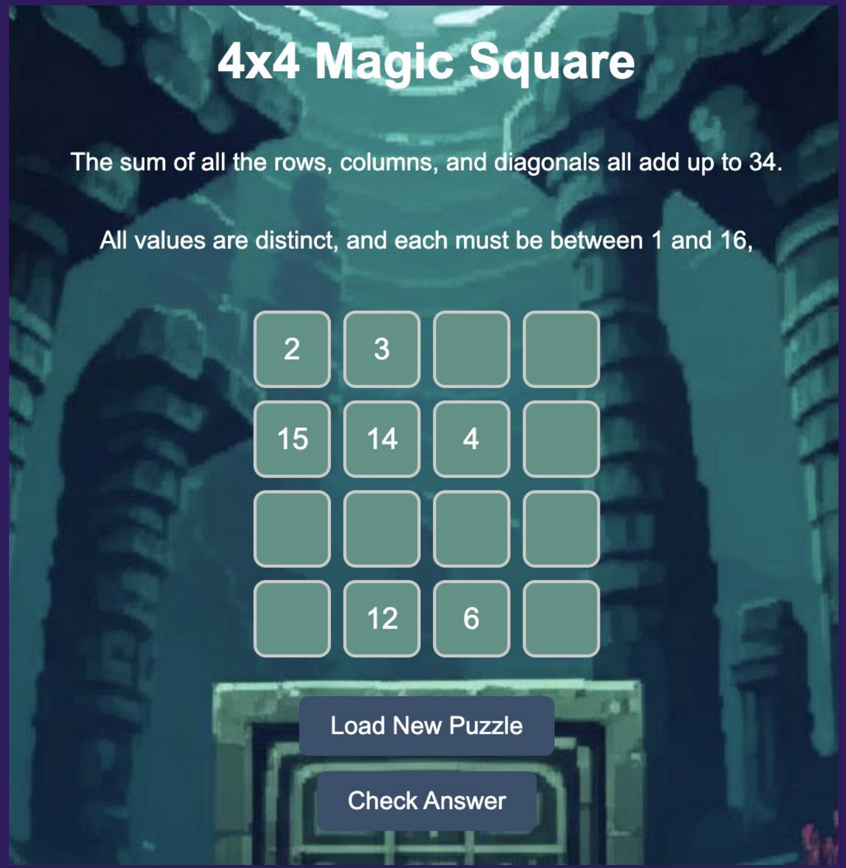
Demo

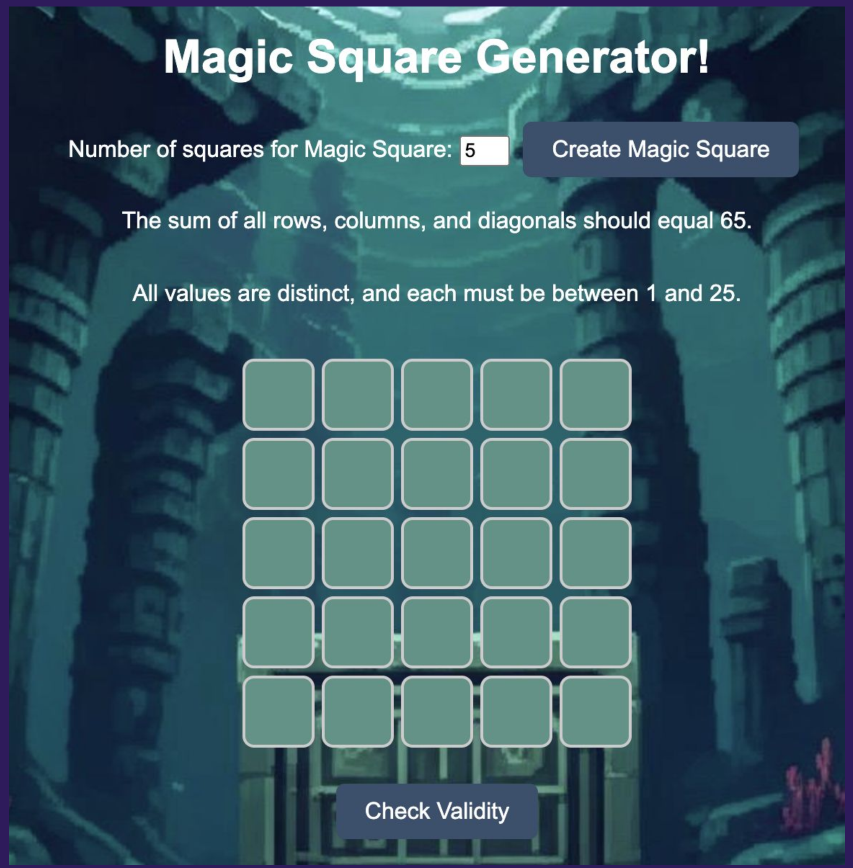
6. Additional Puzzles + Failed Attempts

Magic Squares

{Additional Puzzle}

```
# Constraints:
for sqr in squares:
        s.add(sqr <= n * n)
        s.add(sqr >= 1)
s.add(Distinct(squares))
s.add(sum(hor_vals) == magic_sum)  # rows
s.add(sum(ver_vals) == magic_sum)  # columns
s.add(Sum(diag1) == magic_sum)  # diagonal
s.add(Sum(diag2) == magic_sum)  # diagonal
```





Water Jugs

{Failed Attempt 1 in Alloy}

```
---- VARIABLES AND SETUP ----
abstract sig Jug {
    var amount: Int,
    capacity: Int,
    var whichStep: Step
}
one sig Jug1, Jug2 extends Jug {}
one sig TargetAmount {
    value: Int
}
abstract sig Step {}
one sig Fill, Empty, Pour, Raise, Nothing extends Step {}

var sig FlagRaised in Jug {}
```

Why the failure?

- awkward arithmetic + limited integer support
- Alloy's integers are tiny (–8 to 7) and overflow easily, making it bad at modeling arithmetic heavy problems like the jug puzzle.

Blue Eyes

{Failed Attempt in 23}

```
# Base Case (k = 1):
# - The single blue-eyed person sees 0 others with blue eyes.
# - They reason: "If I did not have blue eyes, then no one would have blue eyes.
#   But the visitor said at least one person does, so it must be me."
# - So they leave on Day 1.

# Inductive Step (Assume true for k = n - 1):
# - Suppose there are k = n blue-eyed people.
# - Each blue-eyed person sees (n - 1) others with blue eyes.
# - They initially assume: "Maybe I have brown eyes, and the n - 1 people I see are the only blue-eyed ones."
# - Under this assumption, those n - 1 people should leave on Day (n - 1), since each would see (n - 2) blue-eyed individuals.
# - When no one leaves on Day (n - 1), this disproves the assumption.
# - Therefore, each of the n blue-eyed islanders realizes they must also have blue eyes.
# - All n blue-eyed people leave on Day n.

# Why brown-eyed people leave:
# - Brown-eyed people see k blue-eyed individuals.
# - They assume: "If I had blue eyes, others would see k - 1 blue-eyed people."
# - But because blue-eyed people do leave on Day k, the brown-eyed people observe this and safely conclude
#   they must not be among the blue-eyed group.
# - Hence, brown-eyed people leave on the next day .
```

Why the failure?

- Direct rule (e.g., “leave on day k”) is easy to add, but it misses the point of the puzzle.
- The puzzle is about knowledge and reasoning, not fixed timing.
- They infer their own eye color by observing:
  - Modeling this is hard because:
    -
- It requires simulating how beliefs change over time.
  -
- You need to represent what each person believes about what others believe (nested reasoning).
  - Modeling in alloy might be easier

Thank you for playing