A1-Assisted Programming

Live Coding - number guessing

AI Assisted - Number Guessing (2 rounds)

2675

```
I generated a random 4-digit string.
You have 20 guesses to guess the secrete string.


Enter a 4-digit number with no repeated digits: 0123
Attempt-1: Correct-0, Misplaced-1
Enter a 4-digit number with no repeated digits: 4567
Attempt-2: Correct-0, Misplaced-3
Enter a 4-digit number with no repeated digits: 9045
Attempt-3: Correct-1, Misplaced-0
Enter a 4-digit number with no repeated digits: 9048
Attempt-4: Correct-0, Misplaced-0
Enter a 4-digit number with no repeated digits: 1290
Attempt-5: Correct-0, Misplaced-1
Enter a 4-digit number with no repeated digits: 6715
Attempt-6: Correct-1, Misplaced-2
Enter a 4-digit number with no repeated digits: 2675
Congratulations! You have guessed the number in 7 attempt(s)
kinleylam@Kinleys-MacBook-Air 2024-Copilot %
```

Guess	Misplaced	Correct
0123	1	0
4567	3	0
9045	0	1
9048	0	0
1290	1	0
6715	2	1
2675	0	4

1st Round
Live Coding,
Problem
Decomposition,
Design

2nd Round
Live Coding
using AI assisted tool

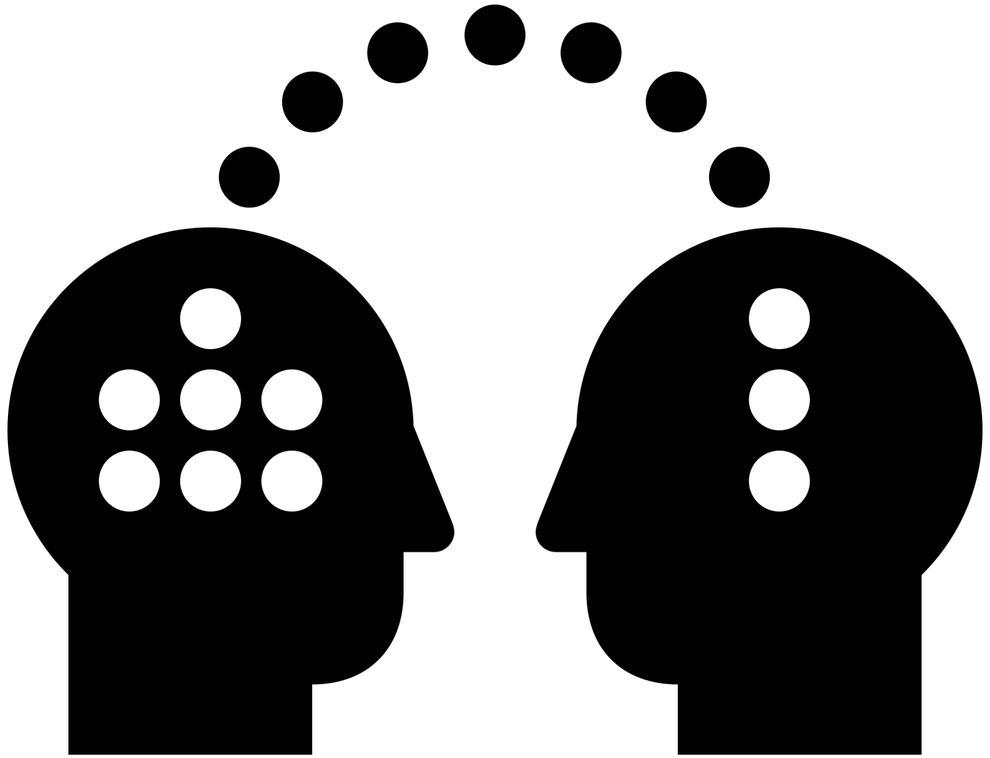
Game Program - Number Guessing

2-player game

- One player (A) picks a secret number and let the other player (B) to guess. Player B has 20 attempts to guess. The game ends when either a correct guess is made or the number of attempts reaches 20.
- The secret number is a 4-digit number, with no repeated digits, leading zero is fine.
  - For examples: 1345, 0913 and 3491 are valid numbers, while 0232 is not.
- As each guess is made, player B will receive two additional pieces of information: correct count and misplaced count.
  - Correct count is the number of digits that appear both in the guess and in the secret number AND in the same position.
  - Misplaced count is the number of digits that appear both in the guess and in the secret number BUT in different positions.
- Examples: secret-number=1357, guess=9371
  - correct count=1 (digit 3), misplaced count=2 (digits 1 and 7)

Volunteers : Player A & Player B

Goal #3 - Knowledge Transfer



1st Round - Player A Role

Secret Number

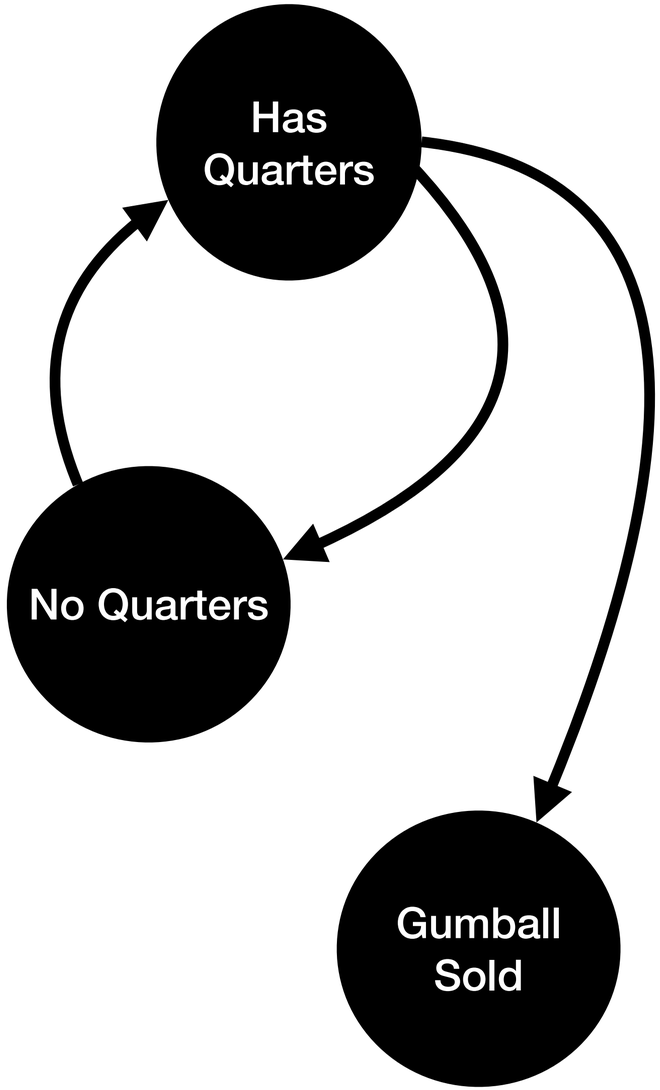
- Live Coding
- Top-Down Design
- Problem Decomposition
- Clean Code

Top-Down Design

Where to Start

Where to start ?????

Out of Gumballs



Begin with bubble diagrams

Try organizing bubbles

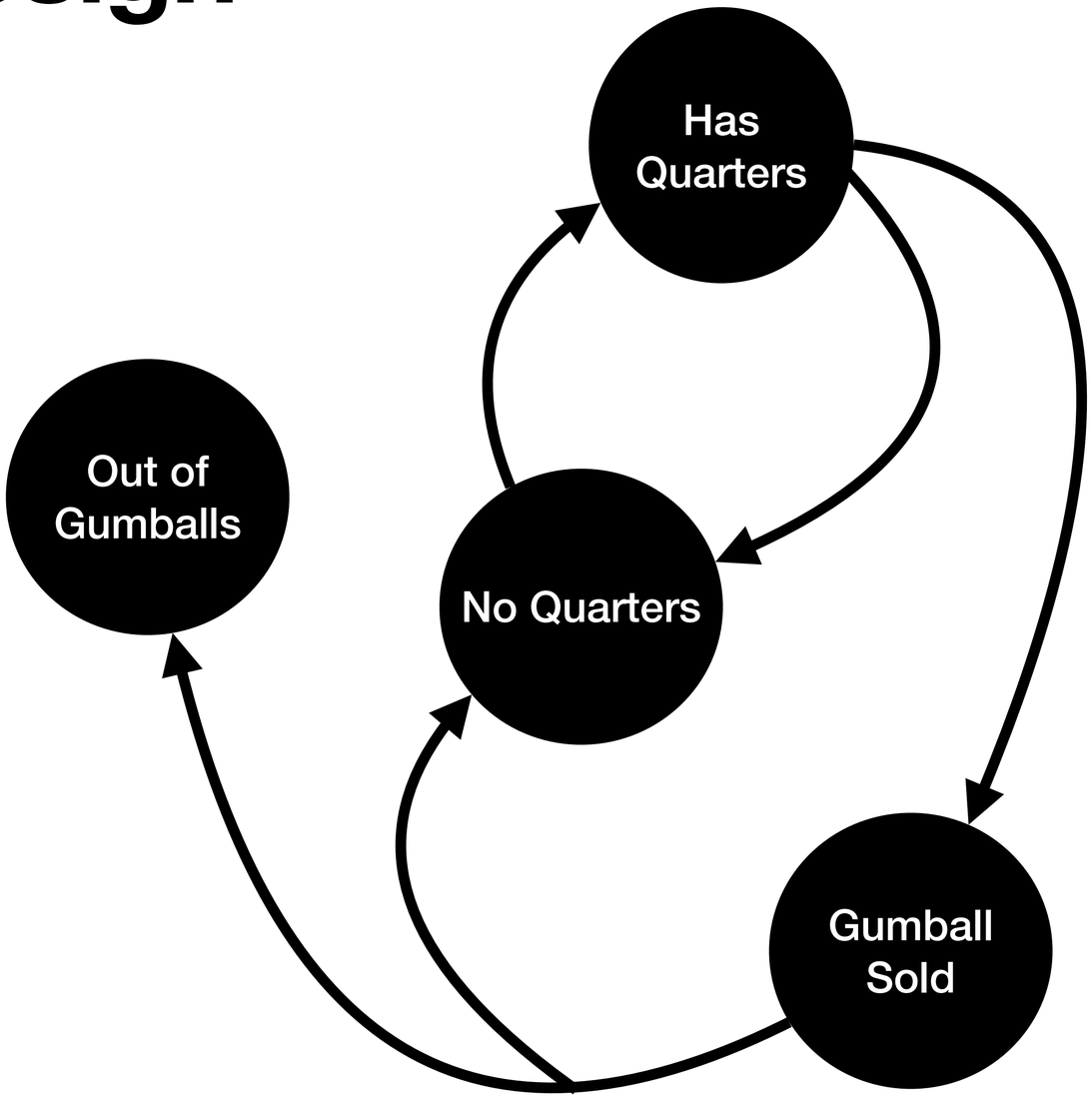
Try connecting bubbles

Try Labelling lines

Top-Down Design

Where to Start

Where to start ?????



Begin with bubble diagrams

Try organizing bubbles

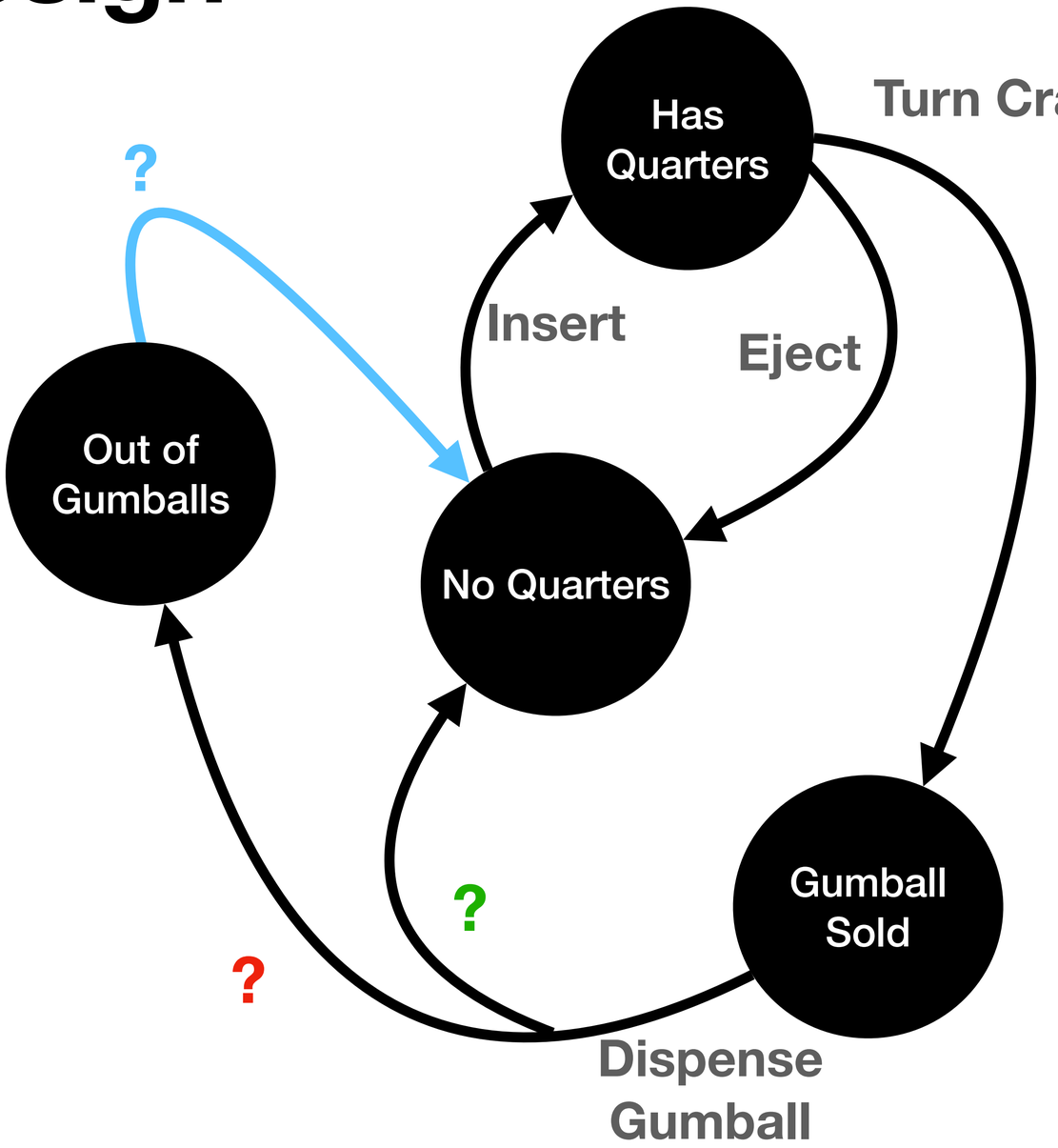
Try connecting bubbles

Try Labelling lines

Top-Down Design

Where to Start

Where to start ?????



Begin with bubble diagrams

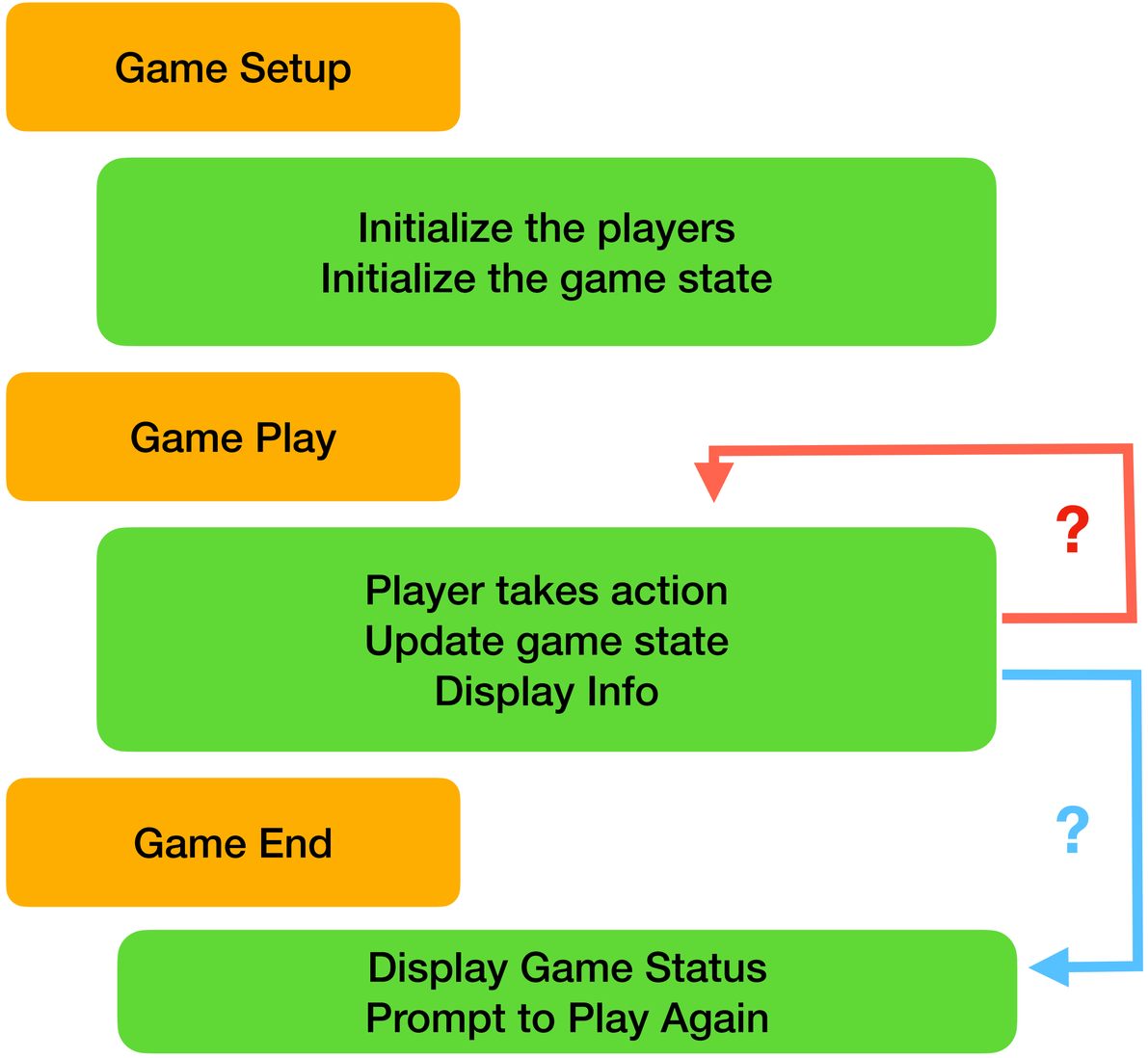
Try organizing bubbles

Try connecting bubbles

Try Labelling lines

General Game Flow

Problem Decomposition - from original requirements



Where to start ?????

Begin with bubble diagrams

Try organizing bubbles

Try connecting bubbles

Try Labelling lines

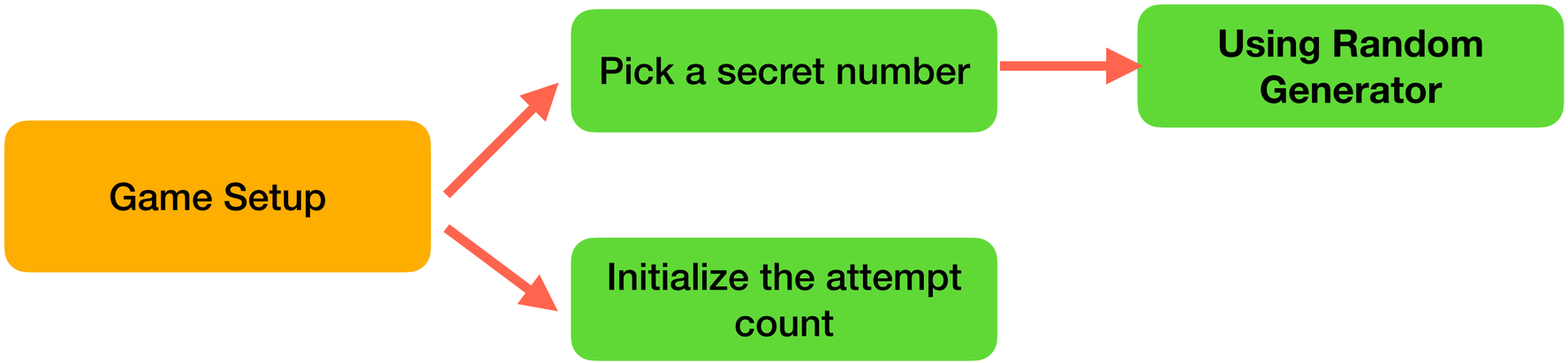
Top-Down Design

Problem Decomposition (breakdown problem into sub-problems)

- Game Setup
  - pick a secret number (random, 4-digit, unique)
  - Initialize the attempt count
- Game Play
  - Prompt Player B to make a guess
  - Validate the guess input (4-digit, unique)
  - Check if the guess matches the secret number
    - Terminate game if correct guess is entered
  - Display correct count and misplaced count
  - Update number of attempts made
    - Terminate game number of attempts made reaches the limit

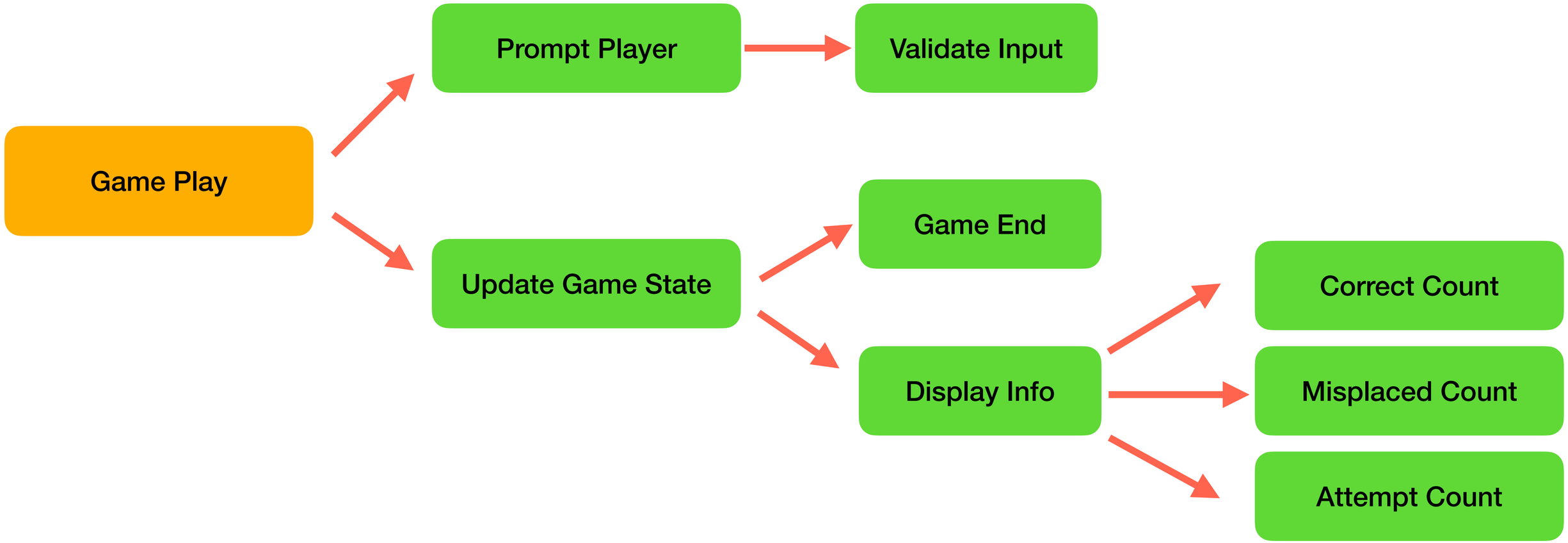
Top-Down Design - Game Setup

Problem Decomposition



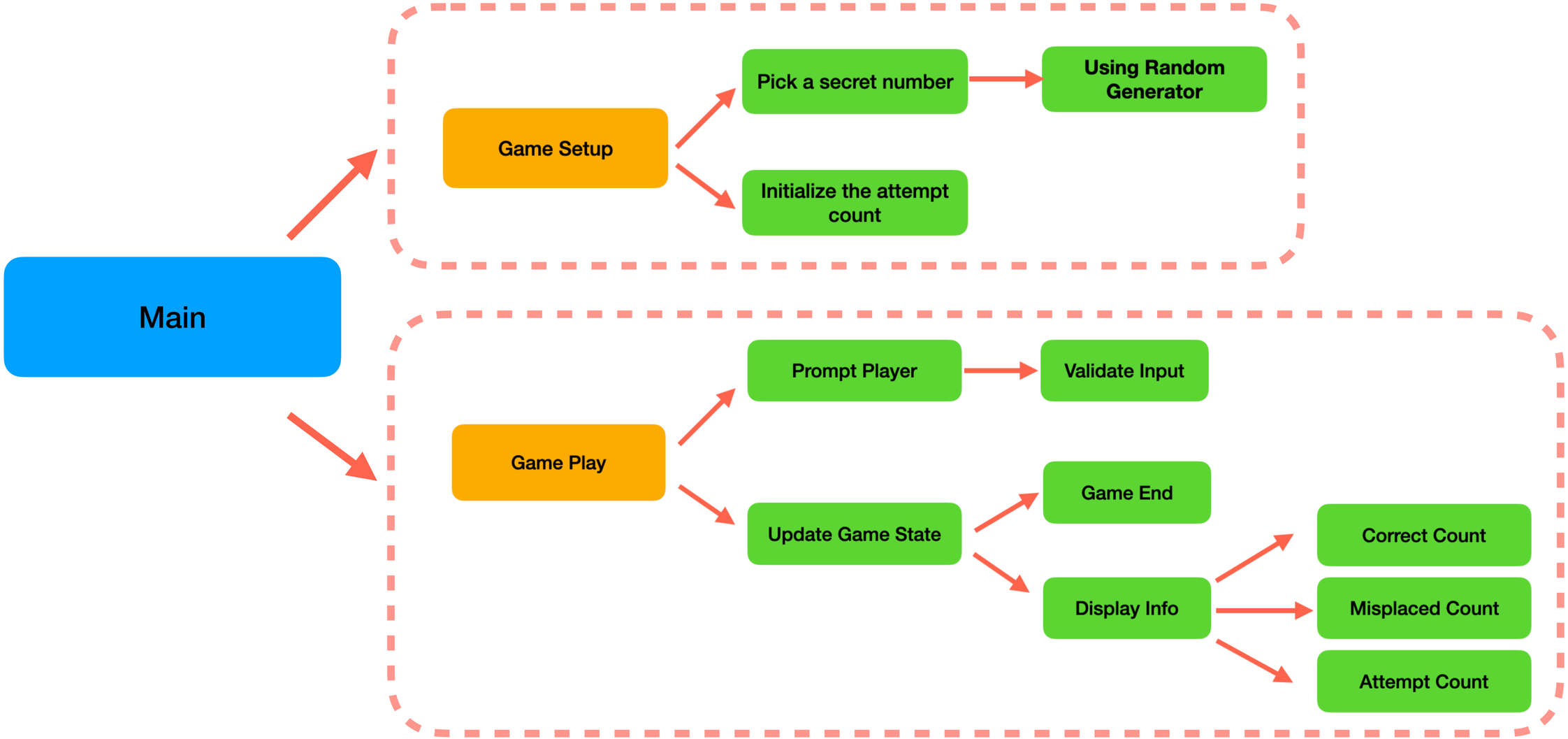
Top-Down Design - Game Play

Problem Decomposition

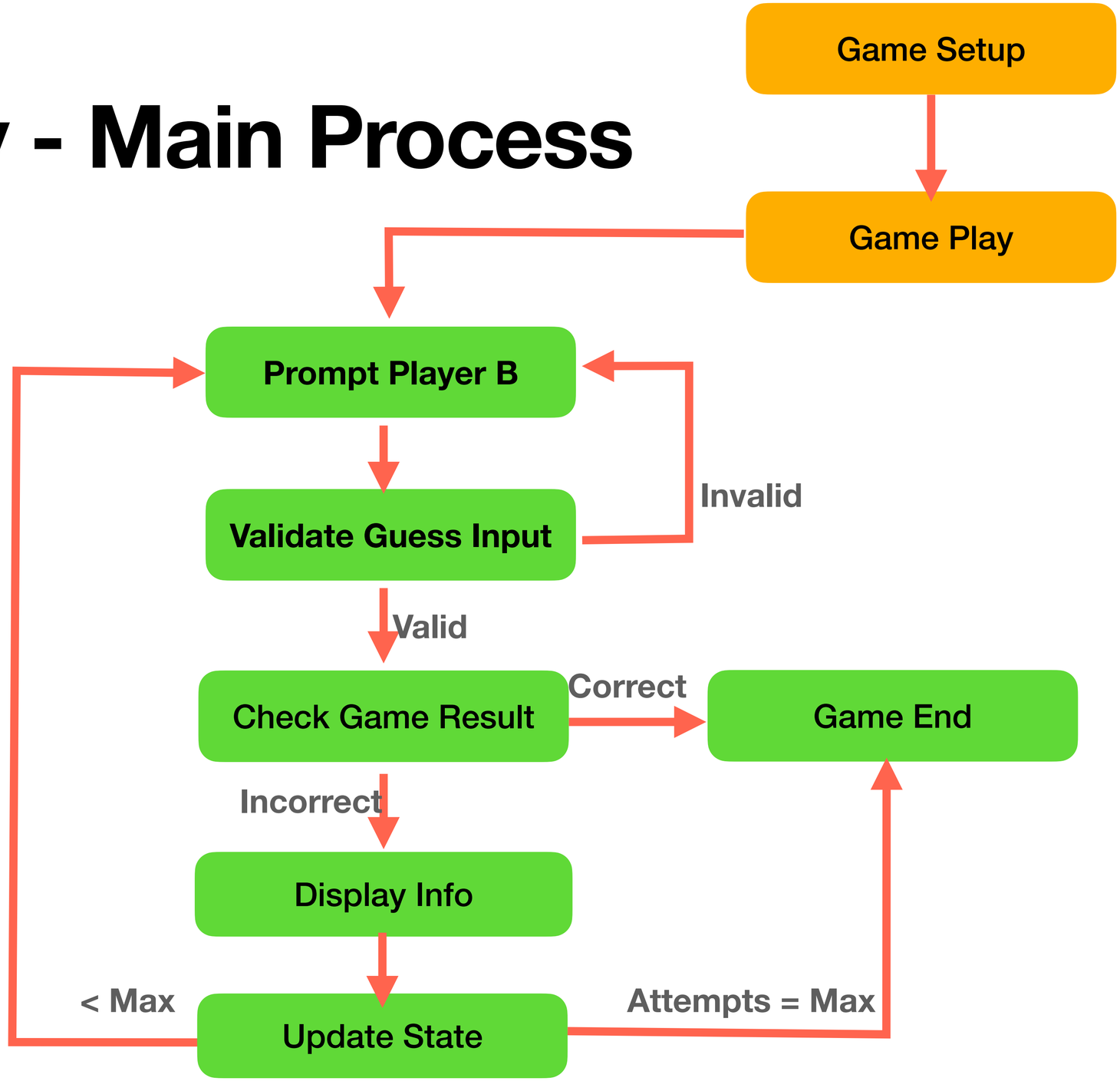


Top-Down Design - Main Process (Flow)

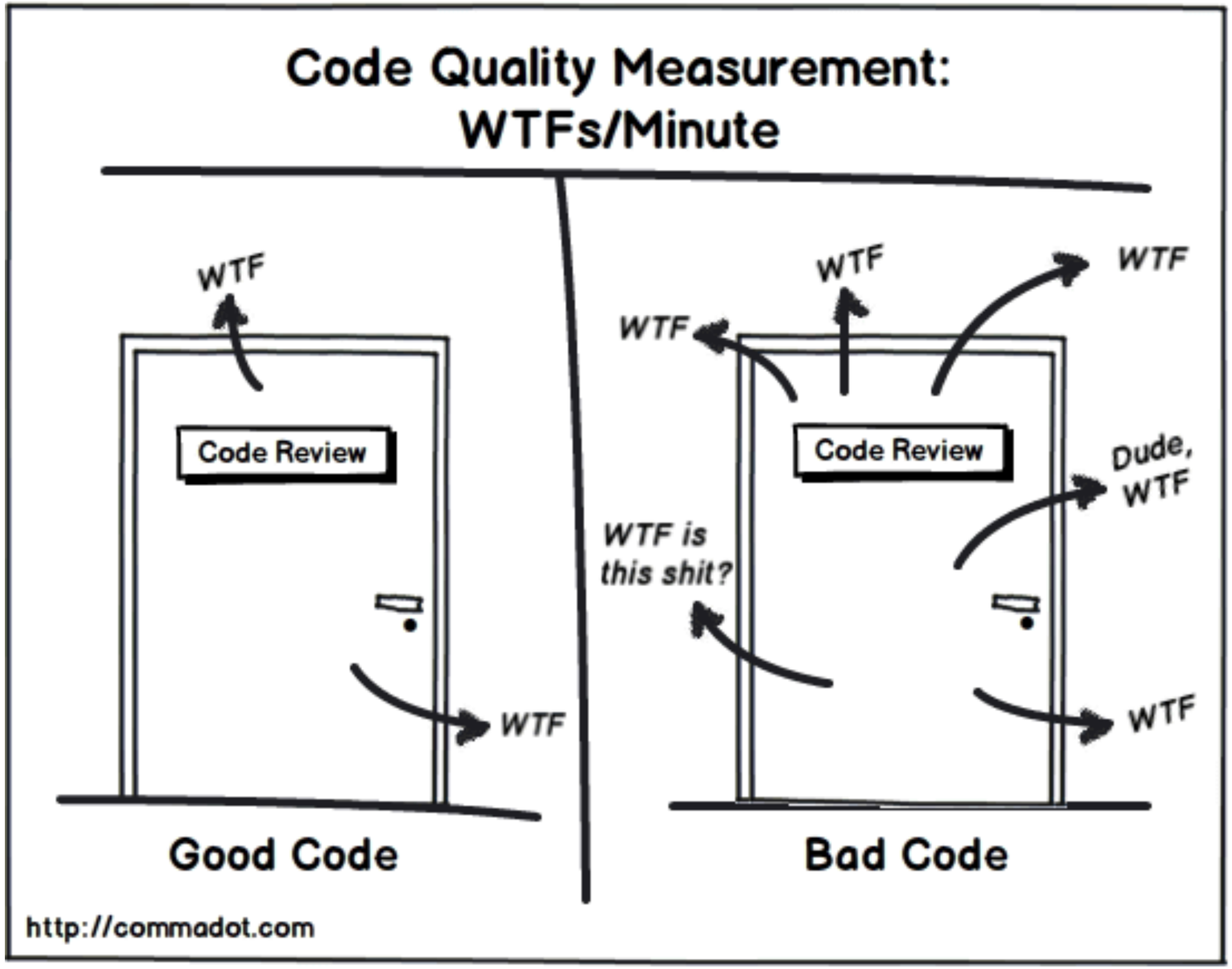
Problem Decomposition



Game Play - Main Process



Goal #2 - Clean Code

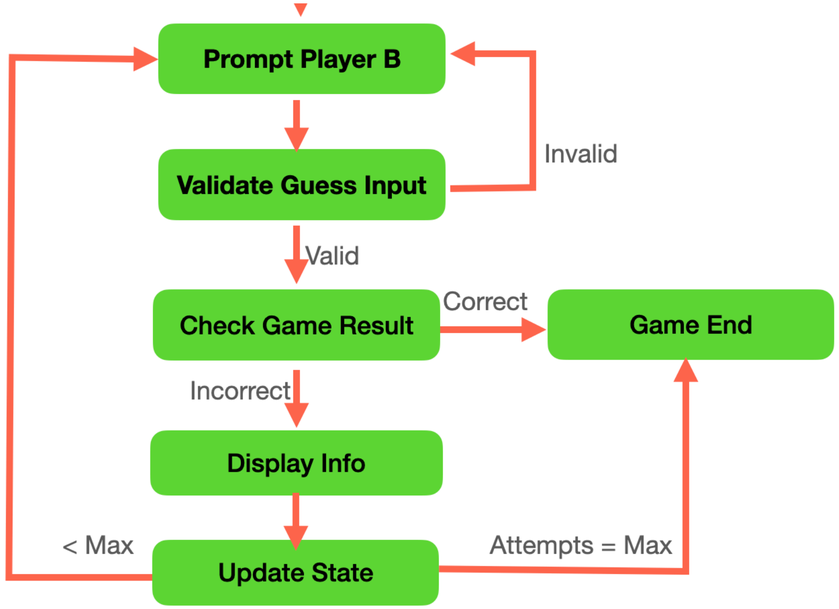


Design Decision To Make

- Internal Representation of the Secret Number
  - Four Seperate Variables: d1, d2, d3, d4
  - Dictionary: {‘d1’:1, ‘d2’:3, ‘d3’:4, ‘d4’:8}
  - String: “1348”
  - List or Tuple of Digits: [1, 3, 4, 8]
  - List of Tuple of Digits as Characters: (‘1’,’3’,’4’,’8’)
- Function: names, input parameters and return
- Note: A design is an on-going decision-making process (remember CHANGES), there is NO one absolute design that works forever in all situations, a trade-off in one direction or another to reach a best solution that satisfies current needs. Designers make the decisions to balance among various factors.

Main Process - Play()

program to interfaces, not implementation



- play(secret: str, max_attempt: int) -> None
  - prompt_guess(length: int) -> str
  - display_hints(secret: str, guess: str, attempt: int) -> None

```
63    def play(secret, max_attempt):
64        attempt = 0
65        while attempt < max_attempt:
66            guess = prompt_guess(len(secret))
67            if guess == secret:
68                print('Winner!')
69                return
70            display_hints(secret, guess, attempt)
71            attempt += 1
72        print('Game Over!!')
```

Think of function as partiocular behaviour interface

Program to interfaces (behaviours)

```python
def pick_a_number(length: int) -> str:
    return '1234'

def get_correct_cnt(secret:str, guess:str) -> int:
    return 0

def get_misplaced_cnt(secret:str, guess:str) -> int:
    return 0

def prompt_guess(length:int) -> str:
    guess = input('Enter your guess >')
    # validate input
    return guess
```

```python
def main():
    length = 4
    max_attempt = 20
    secret = pick_a_number(length)
    print(f'secret number is {secret}')
    play(secret, max_attempt)
```

```python
def display_hints(secret, guess, attempt) -> None:
    correct = get_correct_cnt(secret, guess)
    misplaced = get_misplaced_cnt(secret, guess)
    print(f"Attempt-{attempt} Correct-{correct} Misplaced-{misplaced}")
```

Test Main Process

```
secret number is 1234
Enter your guess >2134
Attempt-0 Correct-0 Misplaced-0
Enter your guess >5678
Attempt-1 Correct-0 Misplaced-0
Enter your guess >84556
Attempt-2 Correct-0 Misplaced-0
Enter your guess >1234
Winner!
```

Function Design

Parameters (Input) and Return (Output)

Single Responsibilty

- pick_a_number(length: int) -> str
- get_correct_count(secret: str, guess: str) -> int
- get_misplaced_count(secret: str, guess: str) -> int

Encapsulate What Varies

- prompt_guess(length: int) -> str
- display_hints(secret: str, guess: str, attempt: int) -> None
- play(secret: str, max_attempt: int) -> None

Decoupling

- main(**argv)
  - secret, max_attempt
  - default: call pick_a_number() to generate the secret number

Goals of Function

- Low Cognitive
- Reduce Repetitive Logic
- Improve Testing
- Improve Overall Code Readability
- Improve Productivity - Scalability

Live Coding - Implementation

Python Review

Random, List, Typing

- random.randint(start,end) -> int
- random.shuffle(sequence) -> None
  - in-place shuffle, sequence must be mutable
- random.sample(sequence, cnt) -> list
- Convert List to string
  - ‘’.join([‘9’,’1’,’4’,’3’]) -> ‘9143’
- Enumerate over list
  - for index, value in enumerate([‘a’, ‘b’, ‘c’, ‘d’])
- Typing Hint:
  - def foo(secret: str, guess: str) -> str

```
>>> random.randint(0,9)
1
>>> random.randint(0,9)
8
>>> random.randint(0,9)
9
>>> random.randint(0,9)
8
>>> random.randint(0,9)
8
>>>
```

randint

shuffle

```
>>> x = [0,1,2,3,4,6]
>>> random.shuffle(x)
>>> x
[3, 6, 2, 1, 4, 0]
>>> random.shuffle(x)
>>> x
[4, 3, 1, 0, 2, 6]
```

```
>>> x = [0,1,2,3,4,6]
>>> random.sample(x, 4)
[0, 3, 2, 1]
>>> random.sample(x, 3)
[2, 4, 0]
>>> random.sample(x, 4)
[2, 3, 1, 4]
>>> random.sample(x, 4)
[3, 6, 4, 1]
```

sample

pick_a_number()

```python
def pick_a_number(length: int) -> str:
    '''…
    digits = string.digits
    selected_digits = random.sample(digits, length)
    return ''.join(selected_digits)
```

Which one you prefer ?

```python
def pick_a_number(length):
    '''…

    digits = list(range(10))
    random.shuffle(digits)

    number = ''
    for i in range(length):
        number += str(digits[i])

    return number
```

```python
def pick_a_number(length: int) -> str:
    numbers = [str(i) for i in range(10)]
    secret = ''
    for _ in range(length):
        digit = random.choice(numbers)
        while digit in secret:
            digit = random.choice(numbers)
        secret += digit
    return secret
```

prompt_guess() - seperate logic from error checking

```
def prompt_guess(length):
    while True:
        try:
            guess = input('Enter your guess >')
            validate_input(guess, length)
            return guess
        except ValueError as ve:
            print(ve)
```

```
def validate_input(guess, length):
    if len(guess) != length:
        raise ValueError(f"Your input must be a {length}-digit number!")

    if guess.isdigit() == False:
        raise ValueError(f"Your input must be all digits!")

    if len(set(guess)) != length:
        raise ValueError(f"Your input must be unique digits!")
```

get_correct_cnt(), get_misplaced_cnt(), display_hints()

```python
def get_correct_cnt(secret, guess):
    assert(len(guess)==len(secret))
    correct = [s for s, g in zip(secret, guess) if s == g]
    return len(correct)

def get_misplaced_cnt(secret, guess):
    assert(len(guess)==len(secret))
    misplaced = [s for s, g in zip(secret, guess) if s != g and g in secret]
    return len(misplaced)

def display_hints(secret, guess, attempt):
    correct = get_correct_cnt(secret, guess)
    misplaced = get_misplaced_cnt(secret, guess)
    print(f"Attempt-{attempt} Correct-{correct} Misplaced-{misplaced}")
```

Highlights

Design Process - Architecture

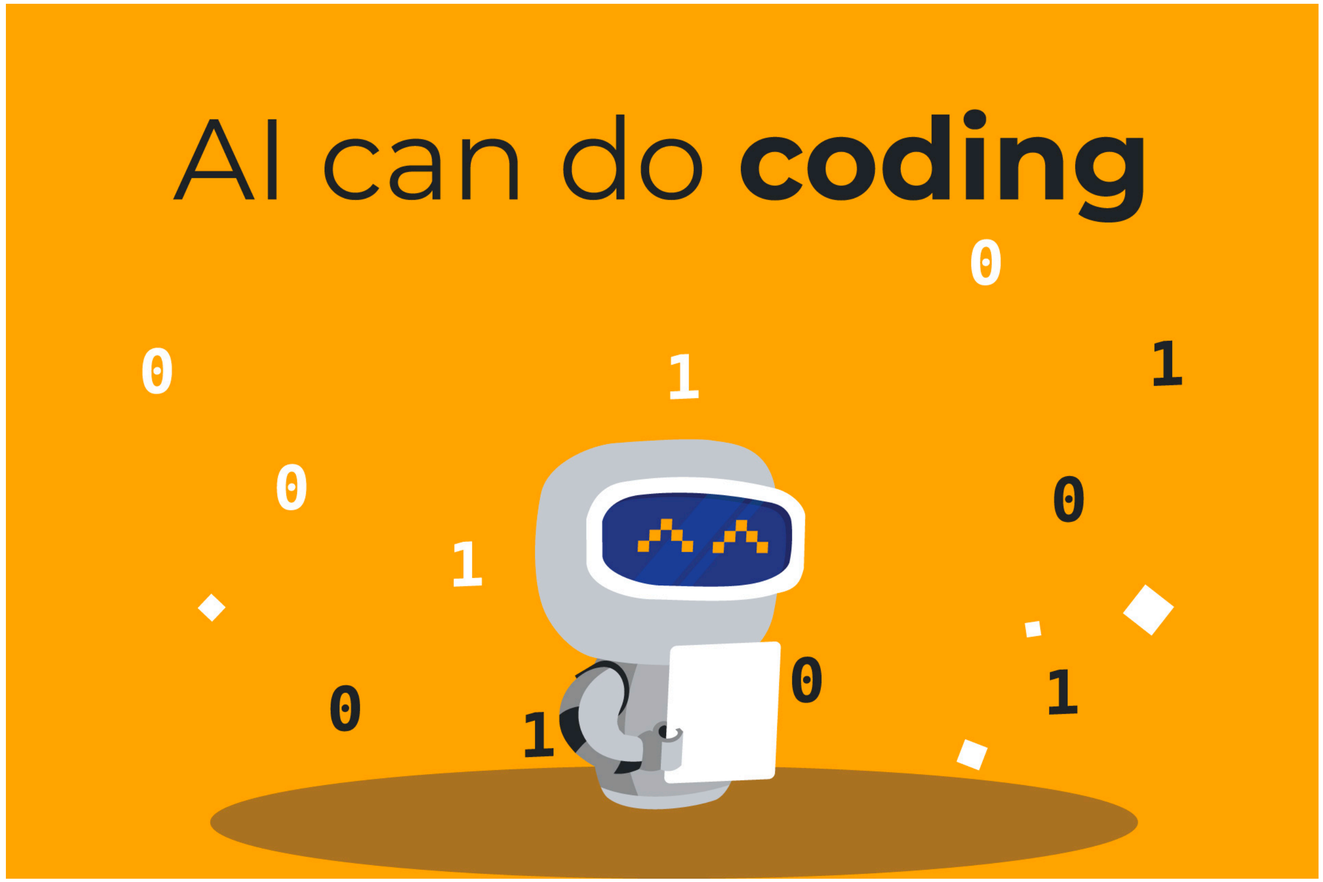
Design Principles

- Design is On-going Process, a decision-making process to finding a Trade-off between technical and bussiness decisions (performance, throughput, time, resource, skills, opportunity cost, market timing, …etc)
- A few design principles
  - Seperate what varies from what static (Decoupling)
    - Identify the aspects of your application that vary and separate them from what stays the same.
  - Program to interfaces, not implementation
  - Single Responsibility
    - seperate or combine: get_correct_cnt(), get_misplaced_cnt()

Home Work
(0) complete the template
(1) Insert general description as docstring to all functions
(2) Design game program for Player B

Goal #1 - AI Assisted Coding

CHAT



Future Programming - AI as Programming Copilot

- When first learning to program, learners often dedicated a significant amount of time working with the syntax and fundamental structure of programs.
- As AI technology advances, the focus for learners will shift towards design, coding strucrture and testing.
- In today's lecture, we will introduce students to the use of Open AI technology as a programming copilot to aid in the development of a game program called Number Guessing.
- Through guided interaction called prompt, we will demonstrate how the AI Assistant facilitates the completion of coding in each function based on the descirbed behavior from the prompt, plus the following common tasks:
    - inserting inline comments, generating test cases, asserting input parameters, and adding function descriptions.
- By the end of the lecture, students will have gained a good understanding of how AI technology is reshaping the future of programming development, including how to integrate AI technology as a programming helper and how to leverage the AI Assistant to bootstrap software development.
- Additionally, we will present instances where the AI Assistant falls short.

AI-Assisted Programming Flow