Frontend Exercise Godot English

Video Bingo

Video Bingo is a game of chance played with a set of uniquely numbered balls and at least one card possessing a set of unique numbers. On each round, a certain number of balls are extracted, and for each ball, its number is marked across all cards. The aim of the game is to mark the numbers of the card, forming prize patterns, with the biggest of them being to complete the card (called bingo). We want you to build a small application similar to this game. using the Godot Engine, taking into account the following guidelines.

The game should have a set of 30 balls, each with a different number (default range from 1 to 60). The balls must have different colors, organized by groups of 10 (for example, balls 1 to 10 are blue, balls 11 to 20 are red, etc.). When the extraction starts the balls should follow a linear path from the right of the screen to the left, with some margin on both sides. Balls must leave within a time interval between them and the complete extraction animation of each ball must last around 3 seconds. The paths should not overlap at any moment. When the ball reaches the end of the path one of two sounds should be played. One sound for when the ball corresponds to a number in the card (see below) and the other for when it doesn't.

The game should also have a card, with a matrix of random unique numbers (same range as the balls). The matrix should be displayed in 3x5 format and the numbers must be ordered. Whenever a ball reaches the end of the path (and the sound is played), if the number is present in a card, it (the number) must be marked in a different color.

Finally, the game should have a play button. Clicking the button should start the ball extraction process without the need for more user interaction. While the extraction is in progress, clicking the button should pause the extraction and all ball animations; and a new click should resume them. When the extraction is finished, clicking the button should reset the game (balls return to the original position, card marks should be clean, new numbers for the card should be generated, etc.) and start a new play with newly randomized balls.

The graphic design is purely secondary, but it should be organized and readable. You are also free to use whatever graphic and sound assets you deem necessary but refrain from using the AssetLib or external scripts. Do also consider all values presented here as default values and be mindful of making them parameterizable. We will value how easy it is to adapt the game parameters (ex: changing the range to 1<>90).

GL;HF

Analyze the code

Write an analysis/opinion about the code snippets shown below. For example, do you see something you would consider good practice or something you think you could do better? Why? How would you do it?

In this section, we are not looking for a description of how the code works.

The snippets are written in Python but they could be written in any other language. The aim here is to judge your technical capacities and your way of thinking, not your knowledge of Python.

Snippet 1

```
# Parse information regarding the prizes that give JACKPOT.
for i in range(0, self.total_number_of_prizes):
    if parameters[13] & (1 << i):
        self.jackpot_prizes.append(i)
        # Parse information regarding the prizes that give BINGO.
for i in range(0, self.total_number_of_prizes):
    if parameters[14] & (1 << i):
        self.bingo_prizes.append(i)
        # Parse information regarding the prizes that give BONUS.
for i in range(0, self.total_number_of_prizes):
    if parameters[15] & (1 << i):
        self.bonus_prizes.append(i)
```

Snippet 2

```
if previous_symbol == params[21] - 3:
    prize_masks[previous_symbol] = prize_by_symbol
unique_symbols = range(1, int(params[21]) + 1)
```

Snippet 3

```
def take(iterable, n, exc_type=ValueError):
    """Take `n` elements from `iterable` (an iterable or iterator).

    If `iterable` is already an iterator, this function will take the next `n`
    elements from it.

    If fewer than `n` elements are available, an exception (default is `ValueError`)
    is raised. This may be disabled by passing `exc_type` a `None` value.
    """
    iterator = iter(iterable)  # this is a no-op on iterators (a Good(r) thing)
    result = []
    while len(result) < n:
        elem = next(iterator, UNDEF)
        if elem is UNDEF:
            break
        result.append(elem)
    if len(result) < n and exc_type is not None:
        raise exc_type("insufficient elements in iterable")
    return result
```

Snippet 4

```
import base64


def base64url_decode(data):
    if isinstance(data, str):
        data = data.encode("ascii")

    rem = len(data) % 4
    if rem > 0:
        data += b"=" * (4 - rem)

    return base64.urlsafe_b64decode(data)
```

Snippet 5

```
try:
    do_some_stuff()
except Exception as error:
    logger.exception(error)
    raise Exception("an error has occurred: {}".format(error))
```

Snippet 6

```
if x == 'A':
    step1(x + "A")  # do step 1 for A
    step2(x + 'A')    # make step 2 for A
    stepN(x + "A")
if x == 'B':
    step1(x + "B")  # do step 1 for A
    step2(x + 'B')    # make step 2 for A
    stepN(x + "B")
if x == 'C':
    step1(x + "C")  # do step 1 for A
    step2(x + 'A')    # make step 2 for A
    stepN(x + "C")
if x == 'D':
    step1(x + "D")  # do step 1 for A
    step2(x + 'D')    # make step 2 for A
    stepN(x + "D")
```

Snippet 7

```
TRUTHY = ("true", "t", "yes", "y", "on", "1")
FALSY = ("false", "f", "no", "n", "off", "0")


def parse(val):
    """Parse a boolean value from a string `val`. This function recognizes common
    ways to represent booleans in other languages (e.g. YAML). Raises `TypeError` if
    `val` is not a string, or `ValueError` if `val` is not recognized.
    """
    check_type(val, str)
    normalized_val = val.strip().lower()
    if normalized_val in TRUTHY:
        return True
    if normalized_val in FALSY:
        return False
    raise ValueError(f"unrecognized boolean value {val!r}")
```

Snippet 8

```
def draw(self, cards, open_cards, bet, eligible, icarus_debug=None, seed_1=None, seed_2=None, debug_draw=None,
         session_token=None, balls=None):
    """
    Please refer to BingoMath interface documentation.

    :param cards:
    :param open_cards:
    :param bet:
    :param eligible:
    :param icarus_debug:
    :param seed_1:
    :param seed_2:
    :param debug_draw:
    :param session_token:
    :param balls: Wildball extraction. When this field is not None the math method used
    will be getGameAfterWild.
    :return:
    """
```

Snippet 9

```
bonus_finish = [False, True][kwargs["bonus_finish"] == str(1)]
```