

Supplementary Material

For ChatGPT4PCG Competition: Character-like Level Generation for Science Birds

Table of Contents

EXAMPLE OF A CHATGPT RESPONSE	1
FULL VERSION OF MODIFIED PROMPTS.....	2
V1	2
V2	3
V3	4
V4	5
V5	6
FULL VERSION OF SUPPLEMENTARY MATERIAL.....	7

Example of a ChatGPT response

To generate a stable structure that looks like the character "I" and meets all the hard constraints, we need to drop the blocks in a specific order and position to ensure stability. We can use the following strategy:

1. First, drop b31 horizontally on layer 0 with its center at slot 8.
2. Then, drop b13 vertically on top of b31 with its center at slot 8.
3. Finally, drop another b31 horizontally on top of b13 with its center at slot 8.

Here is the code to implement this strategy:

```
...
# initialize the structure as an empty WxH grid
structure = [[' ']*W for _ in range(H)]

# drop b31 horizontally on layer 0 with its center at slot 8
ab_drop('b31', 8)

# drop b13 vertically on top of b31 with its center at slot 8
ab_drop('b13', 8)

# drop another b31 horizontally on top of b13 with its center at slot 8
ab_drop('b31', 8)
...
```

This should generate a stable structure that looks like the character "I" and meets all the hard constraints. Note that this strategy satisfies the no boundary intrusion constraint, as all the blocks are dropped within the boundary of the map.

Full version of modified prompts

V1

Use `ab_drop()` function to generate a stable structure that looks like the character <OBJECT>—the goal—and meets all the hard constraints. Dropping position and order are crucial, and they must be determined using techniques in the block-stacking problem.

1. Role

You are a player of the Tetris game who aims to generate a structure that meets the goal while satisfying all the hard constraints.

2. Definitions

Slots: The map's width is equally partitioned into W slots where $W = 20$, with slots 0 and 19 being the most left and right, respectively.

Layers: The map's height is equally partitioned into H layers where $H = 16$, with layers 0 and 15 being the bottom and top layers, respectively.

Base: The bottom of the map, i.e., layer 0.

Map Initialization:

```
# initialize the structure as an empty WxH grid
structure = [[' ']*W for _ in range(H)]
```

3. Environment

There are three block types as follows:

b11, a square block whose width is 1 unit and height is 1 unit

b31, a horizontal block whose width is 3 units and height is 1 unit

b13, a vertical block whose width is 1 unit and height is 3 units

4. Tool

Use the following function to vertically drop a block from layer H such that its center is at slot y and drop earlier blocks representing more bottom parts of the structure.

```
ab_drop(x, y),
```

where x is a block type, and y is the slot number from 0 to $W-1$ where the block center is aligned. After vertically falling down, the block will end up at either the layer on top of the base or a previously dropped block. This function is defined as follows:

```
def ab_drop(x, y):
```

```
    # x is the block type, y is the slot number from 0 to W-1 where the block center is aligned
```

```
        # initialize the drop position at the top of the map
        drop_pos = (H-1, y)
```

```
        # drop the block from the top and move it down until it lands on the base or another block
```

```
        while drop_pos[0] > 0:
```

```
            drop_pos = (drop_pos[0]-1, y)
```

```
            if structure[drop_pos[0]+1][drop_pos[1]] != ' ':
```

```
                break
```

```
        # place the block on the structure
```

```
        structure[drop_pos[0]][drop_pos[1]] = x
```

5. Constraints:

The relevant constraints are given in the following.

5.1 No boundary intrusion: This is a soft constraint that should be met if possible.

Namely, blocks should not intrude on the boundary of the map. In other words, the area of intrusion regions should be zero.

Use `ab_drop()` function to generate a stable structure that looks like the character `<OBJECT>`—the goal—and meets all the hard constraints. Dropping position and order are crucial, and they must be determined using techniques in the block-stacking problem.

1. Role

You are a player of the Tetris game who aims to generate a structure that meets the goal while satisfying all the hard constraints.

2. Definitions

Slots: The map's width is equally partitioned into W slots where $W = 20$, with slots 0 and 19 being the most left and right, respectively.

Layers: The map's height is equally partitioned into H layers where $H = 16$, with layers 0 and 15 being the bottom and top layers, respectively.

Base: The bottom of the map, i.e., layer 0.

3. Environment

There are three block types as follows:

b11, a square block whose width is 1 unit and height is 1 unit

b31, a horizontal block whose width is 3 units and height is 1 unit

b13, a vertical block whose width is 1 unit and height is 3 units

4. Tool

Use the following function to vertically drop a block from layer H such that its center is at slot y and drop earlier blocks representing more bottom parts of the structure.

`ab_drop(x, y)`,

where x is a block type, and y is the slot number from 0 to $W-1$ where the block center is aligned. After vertically falling down, the block will end up at either the layer on top of the base or a previously dropped block.

5. Constraints:

The relevant constraints are given in the following.

5.1 Stability: This is a hard constraint that must be met. Because of gravity, as in the block-stacking problem, a block must be dropped at a proper slot to ensure that the current structure and the resulting structure are stable and do not collapse. Poor weight distribution caused by incorrect placement can lead to structural collapse.

5.2 No boundary intrusion: This is a soft constraint that should be met if possible. Namely, blocks should not intrude on the boundary of the map. In other words, the area of intrusion regions should be zero.

Use `ab_drop()` function to generate a stable structure that looks like the character <OBJECT>—the goal—and meets all the hard constraints. Dropping position and order are crucial, and they must be determined using techniques in the block-stacking problem.

1. Role

You are a player who aims to generate a structure that meets the goal while satisfying all the hard constraints.

2. Definitions

Slots: The map's width is equally partitioned into W slots where $W = 20$, with slots 0 and 19 being the most left and right, respectively.

Layers: The map's height is equally partitioned into H layers where $H = 16$, with layers 0 and 15 being the bottom and top layers, respectively.

Base: The bottom of the map, i.e., layer 0.

Map Initialization:

```
# initialize the structure as an empty WxH grid
structure = [[' ']*W for _ in range(H)]
```

3. Environment

There are three block types as follows:

b11, a square block whose width is 1 unit and height is 1 unit

b31, a horizontal block whose width is 3 units and height is 1 unit

b13, a vertical block whose width is 1 unit and height is 3 units

4. Tool

Use the following function to vertically drop a block from layer H such that its center is at slot y and drop earlier blocks representing more bottom parts of the structure.

```
ab_drop(x, y),
```

where x is a block type, and y is the slot number from 0 to $W-1$ where the block center is aligned. After vertically falling down, the block will end up at either the layer on top of the base or a previously dropped block. This function is defined as follows:

```
def ab_drop(x, y):
```

```
    # x is the block type, y is the slot number from 0 to W-1 where the block center is aligned
```

```
    # initialize the drop position at the top of the map
    drop_pos = (H-1, y)
```

```
    # drop the block from the top and move it down until it lands on the base or another block
```

```
    while drop_pos[0] > 0:
```

```
        drop_pos = (drop_pos[0]-1, y)
```

```
        if structure[drop_pos[0]+1][drop_pos[1]] != ' ':
            break
```

```
    # place the block on the structure
```

```
    structure[drop_pos[0]][drop_pos[1]] = x
```

5. Constraints:

The relevant constraints are given in the following.

5.1 Stability: This is a hard constraint that must be met. Because of gravity, as in the block-stacking problem, a block must be dropped at a proper slot to ensure that the current structure and the resulting structure are stable and do not collapse. Poor weight distribution caused by incorrect placement can lead to structural collapse.

5.2 No boundary intrusion: This is a soft constraint that should be met if possible. Namely, blocks should not intrude on the boundary of the map. In other words, the area of intrusion regions should be zero.

```
---
```

Use `ab_drop()` function to generate a stable structure that looks like the character <OBJECT>—the goal—and meets all the hard constraints. Dropping position and order are crucial.

1. Role

You are a player of the Tetris game who aims to generate a structure that meets the goal while satisfying all the hard constraints.

2. Definitions

Slots: The map's width is equally partitioned into W slots where $W = 20$, with slots 0 and 19 being the most left and right, respectively.

Layers: The map's height is equally partitioned into H layers where $H = 16$, with layers 0 and 15 being the bottom and top layers, respectively.

Base: The bottom of the map, i.e., layer 0.

Map Initialization:

```
# initialize the structure as an empty WxH grid
structure = [[' ']*W for _ in range(H)]
```

3. Environment

There are three block types as follows:

b11, a square block whose width is 1 unit and height is 1 unit

b31, a horizontal block whose width is 3 units and height is 1 unit

b13, a vertical block whose width is 1 unit and height is 3 units

4. Tool

Use the following function to vertically drop a block from layer H such that its center is at slot y and drop earlier blocks representing more bottom parts of the structure.

```
ab_drop(x, y),
```

where x is a block type, and y is the slot number from 0 to $W-1$ where the block center is aligned. After vertically falling down, the block will end up at either the layer on top of the base or a previously dropped block. This function is defined as follows:

```
def ab_drop(x, y):
```

```
    # x is the block type, y is the slot number from 0 to W-1 where the block center is aligned
```

```
    # initialize the drop position at the top of the map
    drop_pos = (H-1, y)
```

```
    # drop the block from the top and move it down until it lands on the base or another block
```

```
    while drop_pos[0] > 0:
```

```
        drop_pos = (drop_pos[0]-1, y)
```

```
        if structure[drop_pos[0]+1][drop_pos[1]] != ' ':
            break
```

```
    # place the block on the structure
```

```
    structure[drop_pos[0]][drop_pos[1]] = x
```

5. Constraints:

The relevant constraints are given in the following.

5.1 Stability: This is a hard constraint that must be met. Because of gravity, a block must be dropped at a proper slot to ensure that the current structure and the resulting structure are stable and do not collapse. Poor weight distribution caused by incorrect placement can lead to structural collapse.

5.2 No boundary intrusion: This is a soft constraint that should be met if possible. Namely, blocks should not intrude on the boundary of the map. In other words, the area of intrusion regions should be zero.

```
---
```

Use `ab_drop()` function to generate a stable structure that looks like the character <OBJECT>—the goal—and meets all the hard constraints. Dropping position and order are crucial, and they must be determined using techniques in the block-stacking problem.

1. Role

You are a player of the Tetris game who aims to generate a structure that meets the goal while satisfying all the hard constraints.

2. Definitions

Slots: The map's width is equally partitioned into W slots where $W = 20$, with slots 0 and 19 being the most left and right, respectively.

Layers: The map's height is equally partitioned into H layers where $H = 16$, with layers 0 and 15 being the bottom and top layers, respectively.

Base: The bottom of the map, i.e., layer 0.

Map Initialization:

```
# initialize the structure as an empty WxH grid
structure = [[' ']*W for _ in range(H)]
```

3. Environment

There are three block types as follows:

b11, a square block whose width is 1 unit and height is 1 unit

b31, a horizontal block whose width is 3 units and height is 1 unit

b13, a vertical block whose width is 1 unit and height is 3 units

4. Tool

Use the following function to vertically drop a block from layer H such that its center is at slot y and drop earlier blocks representing more bottom parts of the structure.

```
ab_drop(x, y),
```

where x is a block type, and y is the slot number from 0 to $W-1$ where the block center is aligned. After vertically falling down, the block will end up at either the layer on top of the base or a previously dropped block. This function is defined as follows:

```
def ab_drop(x, y):
```

```
    # x is the block type, y is the slot number from 0 to W-1 where the block center is aligned
```

```
    # initialize the drop position at the top of the map
    drop_pos = (H-1, y)
```

```
    # drop the block from the top and move it down until it lands on the base or another block
```

```
    while drop_pos[0] > 0:
```

```
        drop_pos = (drop_pos[0]-1, y)
```

```
        if structure[drop_pos[0]+1][drop_pos[1]] != ' ':
            break
```

```
    # place the block on the structure
```

```
    structure[drop_pos[0]][drop_pos[1]] = x
```

5. Constraints:

The relevant constraints are given in the following.

5.1 Stability: This is a hard constraint that must be met. Because of gravity, as in the block-stacking problem, a block must be dropped at a proper slot to ensure that the current structure and the resulting structure are stable and do not collapse. Poor weight distribution caused by incorrect placement can lead to structural collapse.

5.2 No boundary intrusion: This is a soft constraint that should be met if possible. Namely, blocks should not intrude on the boundary of the map. In other words, the area of intrusion regions should be zero.

```
---
```

Full version of supplementary material

For the full version of supplementary material containing experiment data, please visit <http://chatgpt4pcg.github.io/supplement>.