RLSS: A Deep Reinforcement Learning Algorithm for Sequential Scene Generation (Supplemental Material)

1. Indoor Planning

In indoor planning, our task is to place furniture objects given specific room boundaries. We consider three types of rooms: bedrooms, living rooms, and offices.

The reward function is a mapping from $r : S \times A \to \mathbb{R}$, where S is the state space and A is the action space. For a specific action $a_i \in A$ selected in a state $s_i \in S$, the reward is denoted with $r(s_i, a_i)$. The reward function follows common pattern introduced in the paper. The mapping $r : (s_i, a_i) \mapsto r(s_i, a_i)$ is calculated in the following order

$$r(s_i,a_i) := \begin{cases} -1, & \text{if CheckFailureCondition}(s_i) = \text{True}, \\ -0.1, & \text{elseif CheckElementCountCondition}(s_i,a_i) = \text{True}, \\ r_3+\delta, & \text{elseif Search}(s_i,a_i,3) = \text{True}, \\ r_2+\delta, & \text{elseif Search}(s_i,a_i,2) = \text{True}, \\ r_1+\delta, & \text{elseif Search}(s_i,a_i,1) = \text{True}, \\ 1, & \text{if CheckSuccessfulCondition}(s_i) = \text{True}, \\ -0.1, & \text{otherwise.} \end{cases} \quad (1)$$

The functions returning a Boolean value are defined hereafter:

- CheckFailureCondition(s_i, a_i) returns True if the failure condition is fulfilled. Failure condition in this problem is 0.75 percentage of the room area or more is occupied by furniture and still plausible arrangement is not generated.

- CheckElementCountCondition(s_i, a_i) returns True if the last object exceeds the predefined maximum amount for its category.

- Search$(s_i, a_i, 3)$ returns True if after placing the last object some existing structure is completed in the room. Structure complexity in the completed structure should be greater or equal 3.

- Search$(s_i, a_i, 2)$ returns True if the last object is placed in the existing substructure. Structure complexity is greater or equal 2.

- Search$(s_i, a_i, 1)$ returns True if the last object is placed in the room. Structure complexity is equal 1.

Table 1. Parameter values used for indoor planing.

Parameter	Value
r_3	0.3
r_2	0.2
r_1	0.1

- CheckSuccessfulCondition(s_i) returns True if the successful scene condition is reached. The successful condition depends on the area of the room and the total accumulated reward: $(total\_object\_area/room\_area \geq 0.4\ or\ total\_reward >= R\_thr)\ and\ important\_object\_fnd == True$. Where $total\_object\_area$ is total area of all objects in the room, $room\_area$ is room area and R_thr threshold reward. In order to find the criteria for a "successful scene", we analyze human generated scenes and for each room calculate its score (depending existing objects in this room), which is the total reward required to generate this scene by our definition. R_thr is average of these scores. $important\_object\_fnd$ is True if important object(s) is placed in the room.

Search$(s_i, a_i, option)$ function only analyses existence of a substructure which also should satisfy hard constraints, then PlaceObject$(s_i, a_i, option)$ function is called to place this object with given option, where $option \in \{1, 2, 3\}$. $\delta > 0$ ($\delta = 0.3$) if important object for this room is first time added to the scene, otherwise it is equal to zero. Important objects for bedrooms is bed, for living rooms is sofa, for office is desk.

We use one representative for each object category during the training process. In the inference time for each generated scene wide variety of models are used. Models are selected based on whether they violate hard constraints or not. Such a representation makes it possible for a network to learn efficiently.

Following substructures are used to generate different rooms:

- Bedrooms: 1) double bed, nightstand, ottoman and laptop 2) desk, chair (armchair, office chair) and laptop 3) dressing table and ottoman 4) wardrobe cabinet and (dresser, shoes cabinet) 6) single bed, nightstand and table lamp 7) double bed and floor lamp 8) single bed and dresser 9) dresser and dressing table 10) double bed and shelving 11) bunker bed and floor lamp 12) tv stand and loudspeaker 13) sofa and coffee table 14) plant 15) hanger 16) baby bed 17) pedestal fan 23) kitchen cabinet

- Living rooms: 1) sofa, (coffee table, floor lamp), laptop 2) piano and ottoman 3) sofa and plant 4) coffee table and chair 5) coffee table and ottoman 6) shelving 7) fireplace 8) wardrobe cabinet 9) nightstand 10) dresser 11) tv stand and tv 12) hanger 13) plant
- Offices: 1) desk, (office chair, armchair) and laptop 2) sofa and (coffee table, plant) 3) tv stand and loudspeaker 4) piano and ottoman 5) wardrobe cabinet and dresser 6) desk, office chair and armchair 7) shelving 8) plant 9) wardrobe cabinet 10) floor lamp 11) workplace 12) nightstand 13) whiteboard 14) dresser

Initially, we find plausible placements and orientations for each object. First object in the structure is placed according to this domain knowledge. Object positions and orientations are uniformly selected to increase variety. Each substructure has its own placements relative to a room boundary. During the training one of these placements selected based on hard constraints and existing objects in the room. Object types and orientations can be adjusted to increase diversity.

2. Angry Birds Level Generation

For generating Angry Birds levels, we employ as a predefined design objective to have at least 4 closed structures in each scene, which are the defense houses to place pigs. The game environment consists of an 800×800 pixel rectangle. To make levels more interesting, we symmetrically place two instances of each object for elements which are not in the middle.

The reward function is a mapping from $r : S \times A \to \mathbb{R}$, where S is the state space and A is the action space. For a specific action $a_i \in A$ selected in a state $s_i \in S$, the reward is denoted with $r(s_i, a_i)$. The mapping $r : (s_i, a_i) \mapsto r(s_i, a_i)$ is defined as follows:

$$r(s_i,a_i) := \begin{cases} -1, & \text{if CheckFailureCondition}(s_i) = \text{True}, \\ -0.1, & \text{elseif CheckElementCountCondition}(s_i,a_i) = \text{True}, \\ r_4, & \text{elseif Search}(s_i,a_i,4) = \text{True}, \\ r_3, & \text{elseif Search}(s_i,a_i,3) = \text{True}, \\ r_2, & \text{elseif Search}(s_i,a_i,2) = \text{True}, \\ r_1, & \text{elseif Search}(s_i,a_i,1) = \text{True}, \\ 1, & \text{if CheckSuccessfulCondition}(s_i) = \text{True}, \\ -0.1, & \text{otherwise.} \end{cases} \tag{2}$$

We divide all object categories into two groups by their size, first group $object\ width \geq object\ height$, and all other objects considered to be the second group. Some of the following functions work with first or second group objects only. The functions returning a Boolean value are defined hereafter:

- CheckFailureCondition(s_i, a_i) returns True if the failure condition is fulfilled. If the plausible scene is not generated after maximal episode length (60 for this case), then this generation is considered failure.
- CheckElementCountCondition(s_i, a_i) returns True if the last object exceeds the predefined maximum amount for its category.
- Search$(s_i, a_i, 4)$ returns True if placing the last object creates a closed structure to place a pig. Works only with the first group objects. Structure complexity is equal to 4 (including pig).
- Search$(s_i, a_i, 3)$ returns True if placing the last object creates a closed structure (not necessarily big to place a pig). Works only with the first group objects. Structure complexity is equal to 3.
- Search$(s_i, a_i, 2)$ returns True if it is possible to vertically pile up the last object to make it in equal height with another object. Structure complexity is equal to 2.
- Search$(s_i, a_i, 1)$ returns True if there is enough space to place the last object or two of them (will be randomly chosen) on the ground or on top of other object or if it is possible to place the last object on top of another object not violating the hard constraint (stability). Structure complexity is equal to 1.
- CheckSuccessfulCondition(s_i) returns True if the successful condition is reached. The successful scene condition in this case: $total\_reward \geq R\_thr$ In order to find the criteria for a "successful level", we analyze Angry Birds game levels and for each level calculate its score (depending existing objects in this level and their position), which is the total reward required to generate this scene by our definition. R_thr is average of these scores.

As in the previous problem, Search$(s_i, a_i, option)$ function only analyses existence of a substructure, then PlaceObject$(s_i, a_i, option)$ function is called to place this object with given option, where $option \in \{1, 2, 3, 4\}$. Detailed network architecture is given in Fig. 1. Some examples of generated structures for Angry Birds game given in the Fig. 2.

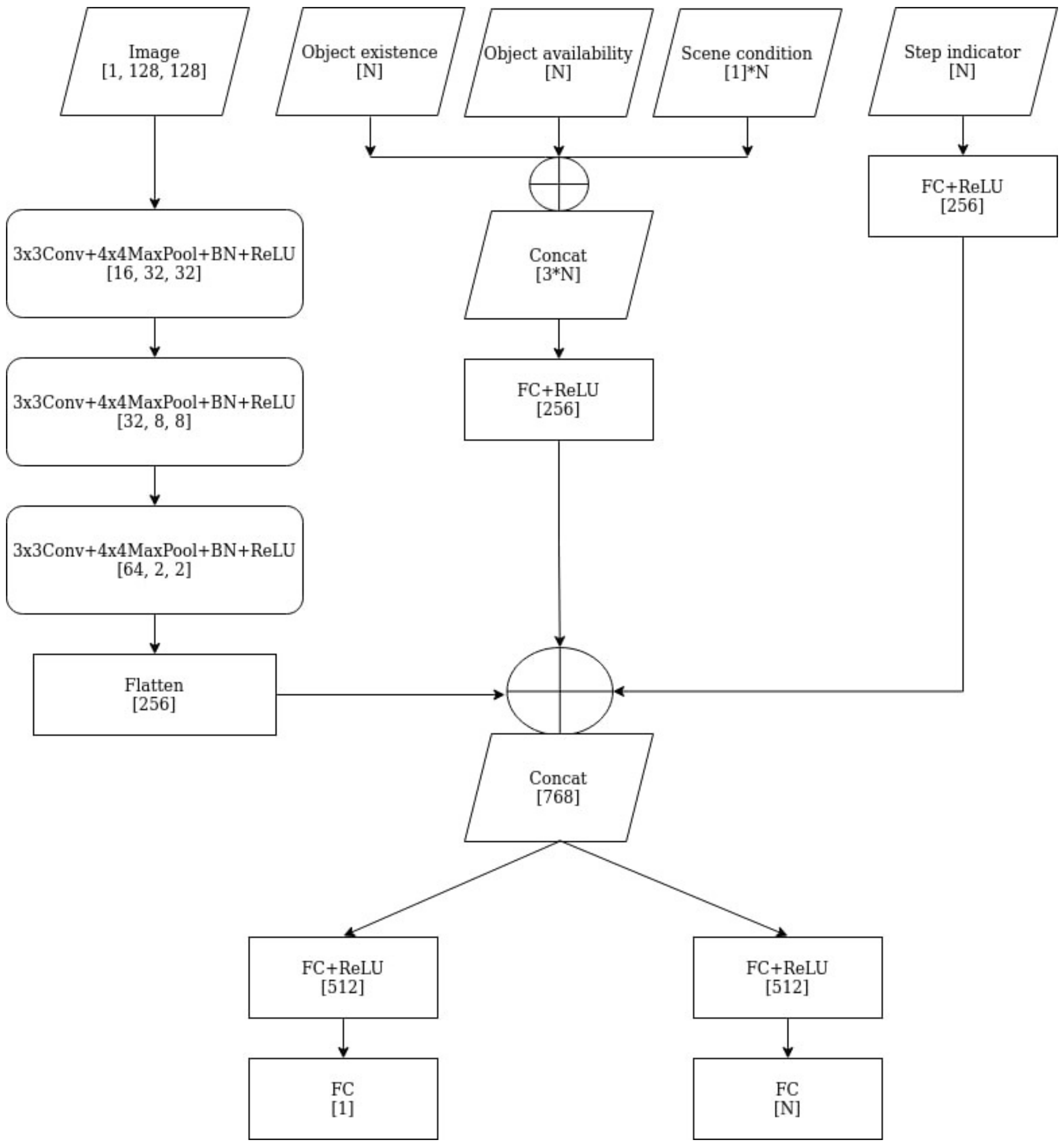

Figure 1. Neural network architecture. N refers to the number of objects.

Table 2. Parameter values used for level generation.

Parameter	Value
r_4	0.5
r_3	0.2
r_2	0.1
r_1	0.05

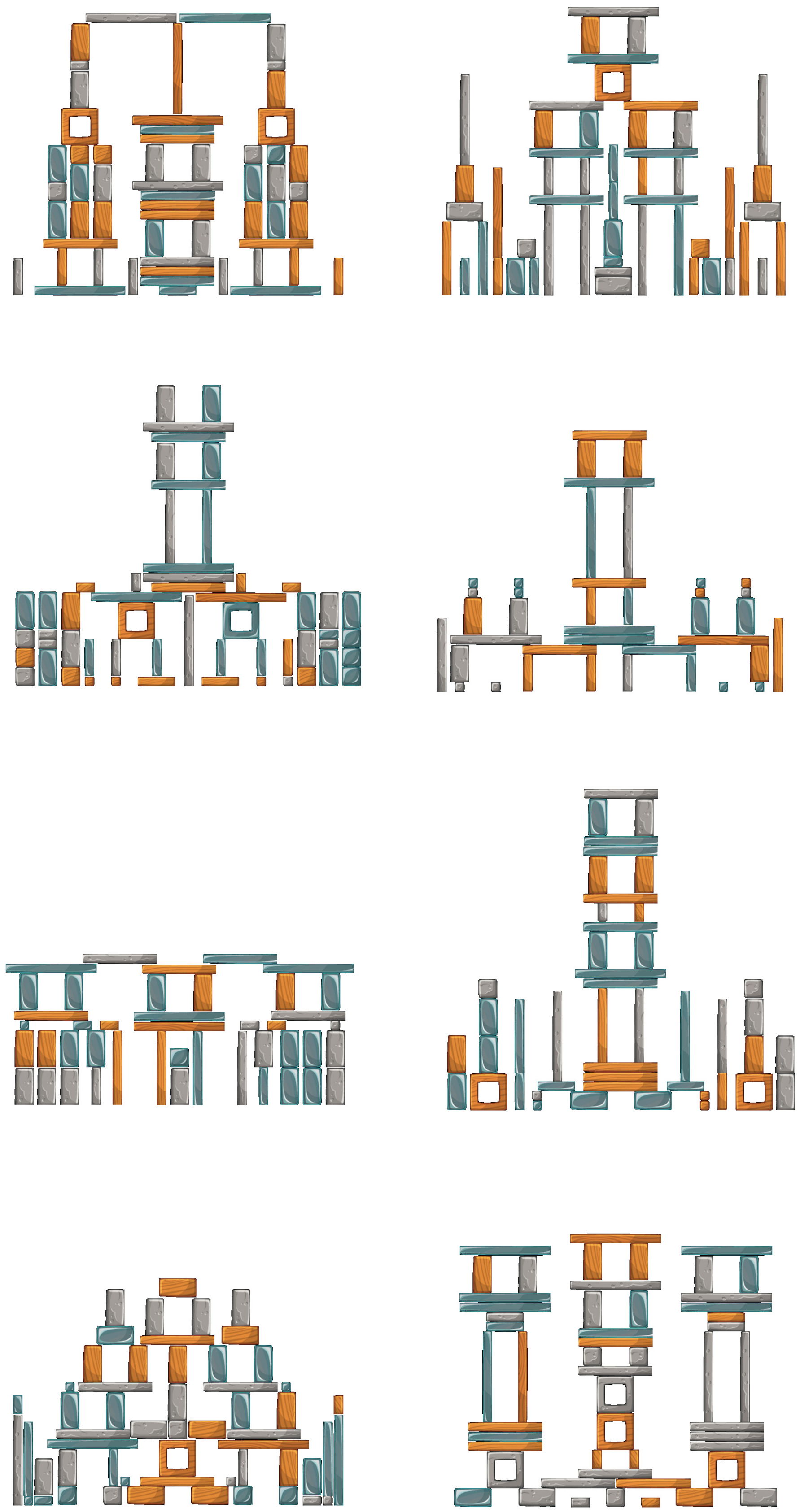
Figure 2. Examples of generated structures.