RLSS: A Deep Reinforcement Learning Algorithm for Sequential Scene Generation

Azimkhon Ostonov, Peter Wonka, Dominik L. Michels
KAUST Visual Computing Center
Campus, Bldg 1, Thuwal 23955, KSA
{azimkhon.ostonov, peter.wonka, dominik.michels}@kaust.edu.sa

Abstract

We present RLSS: a reinforcement learning algorithm for sequential scene generation. This is based on employing the proximal policy optimization (PPO) algorithm for generative problems. In particular, we consider how to effectively reduce the action space by including a greedy search algorithm in the learning process. Our experiments demonstrate that our method converges for a relatively large number of actions and learns to generate scenes with predefined design objectives. This approach is placing objects iteratively in the virtual scene. In each step, the network chooses which objects to place and selects positions which result in maximal reward. A high reward is assigned if the last action resulted in desired properties whereas the violation of constraints is penalized. We demonstrate the capability of our method to generate plausible and diverse scenes efficiently by solving indoor planning problems and generating Angry Birds levels.

1. Introduction

Generative modeling has seen drastic improvements in recent years. Especially for images, state-of-the-art generative adversarial networks (GANs) produce fantastic results [11, 15, 16]. Even though generative models such as GANs or variational autoencoders (VAEs) [19] are considered to be unsupervised methods, a large amount of data is still required. While multiple attempts have been made in reproducing the success of image-based generative models in other domains, such as scenes, meshes, and point clouds, the results are far behind. A major obstacle is the lack of data, as there are no high quality data sets of scenes that contain many models. Many data sets are small, e.g. [35], or contain too many low quality models. For example, the SUNCG data set [32] contains multiple low quality models generated by amateur modelers that violate commonly accepted hard constraints (in addition, the data set is currently unavailable due to a legal dispute).

In order to make progress on generative modeling for scene generation, we propose to build a learning framework that does not require a large amount of training data. In this context, reinforcement learning is an ideal choice as the need for training data is replaced by reward function design. As reinforcement learning is traditionally applied to maximize expected cumulative reward, it is not generally used to generate a large variety of scenes. In this work we propose a reinforcement learning algorithm to be able to generate a large variety of scenes. We call our approach RLSS. It operates sequentially and places scene objects one-by-one.

Specifically, our major contributions are as follows.

- We present RLSS: (to our knowledge) the first reinforcement learning based scene generation algorithm. The distinguishing characteristic is that RLSS can generate a large variety of scenes for the same input (i.e. scene boundary) taking into account domain constraints.

- We propose an efficient approach for scene synthesis problem which enables to solve this task by separating the problem requirements into two different categories: hard constraints that are included in the environment and predefined design objectives, which the network learns how to achieve, given different initial scene boundaries, during the training process.

- Our reward designing approach makes it easy to adapt this method for many scene synthesis problems without spending too much time on reward calculation for actions.

- We demonstrate the advantages of our method to generate plausible and diverse scenes efficiently by solving indoor planning problems and generating Angry Birds levels.

2. Related Work

Employing reinforcement learning in the context of generative problems has been addressed in the community.

SPIRAL [10] is a reinforcement learning based adversarial agent which learns to synthesize visual programs for graphic engines in order to generate images. It has been shown that their method works well for image reconstructions on MNIST [21] and OMNIGLOT [20] data sets but fails to synthesize new examples. Model-based reinforcement learning [14] was applied to image reconstruction for stroke-based paintings. The fundamental difference between these methods and our method is that in contrast to rewarding the agents based on the discriminators' outputs, in our case this is done by simulating the environment and checking relevant constraints.

Formalizing reinforcement learning as a probabilistic inference technique has been explored in [4, 1, 22]. Connecting reinforcement learning with posterior regularization [9] to incorporate the domain constraints to deep generative models has been considered in [13]. Compared to this approach, our method is based on directly employing reinforcement learning as a generative model.

Deep generative models [11, 19] have been extensively used by scientific community to synthesize scenes in indoor planning [38, 24, 37, 29, 41]. These models are mostly image-based and therefore use pixel-level reasoning to distinguish objects and extract spatial features from an image. Which can be poorly adapted when handling various domain-based constraints in scene synthesis problems. As a result, one either need to generalize the problem and accept more simplified domain constraints [24], or use constraint violation check at each step along with generative models to deal with the problem [38, 37]. However, this is associated with costs: in the first case, we will solve the sub-problem instead of solving the real problem, in the second case, this leads to an increase in the synthesis time.

Another major direction to generate indoor scenes is related with using Markov chain Monte Carlo (MCMC) methods [28, 25, 40, 17]. In these methods problem specific objective functions are optimized based on some rule based criteria. The main drawback of using MCMC sampling in these methods is generation time, which requires thousands of iterations to synthesize one plausible scene.
We also would like to mention the reinforcement learning has been used in several contexts different from generative problems [23, 27, 42, 36, 12].

3. Method

In this section, we provide an overview of RLSS: our reinforcement learning based sequential scene generation algorithm. We provide required background information and discuss relevant components in detail.

3.1. Overview

We developed RLSS based on the proximal policy optimization (PPO) [31] method for scene generation problems. The scene we would like to generate is represented as an environment in which objects are added iteratively one-by-one. The neural network is trained on a 2D representation of the scene. If the visual content we are generating has three dimensions, then its most informative 2D view is used during the training. The set of all actions which can be applied in the environment is denoted with A. The reward which the agent obtains after taking an action $a \in A$ is denoted by r. This formulation of the scene allows us to use reinforcement learning algorithms. However, the use of standard reinforcement learning is limited to optimization problems, where the agent needs to collect as much reward as possible in a virtual environment. In each step the agent chooses an action among the set of possible actions which at the end will result in the highest reward. Therefore, in the standard reinforcement learning setup, the main target of the agent is to maximize the cumulative reward. In our setting we additionally consider increasing variety, which is achieved by introducing randomness in the action selection process.

We introduce the following two problem-specific key definitions which will help us to effectively explain our method.

- **Hard constraints** are a set of constraints that should always be satisfied when synthesizing a scene. Usually, this type of constraints come from the problem domain and violating these constraints makes it impossible to continue the generation process. In our method, these constraints are included in the environment.

- **Design objectives** comprise a set of requirements which shape the general appearance of the generated scene. Each successfully generated scene should have these features. This can be understood as a broad definition of a scene we would like to synthesize. Only such scenes which fulfill predefined design objectives could be considered as valid examples for good scenes.

Different from the most of the scene synthesis algorithms our method learns how to generate successful scene based on predefined criteria (design objectives) and domain knowledge is applied to place objects in the scene. This method has particular advantages where relationship or rules exist for placing objects. For example, in physics based games or in indoor planning.

3.2. Scene Abstraction

A scene consists of the initial scene boundary and set of objects, each defined with its center's position $P(x, y, z)$,

bounding box (box with minimal volume which can fully contain this object) and orientation α relative to local coordinate axis.

Structures (groups). First, we find independent structures in the problem domain. Each structure consists of one or more objects and together these objects play some function in the current problem domain. These structures can be derived from acceptable scene examples. We define the set of all structures we use in the problem as $\mathcal{S}$. Each structure $st \in \mathcal{S}$ has a complexity (the number of different objects in this structure) defined as $C(st)$.

Placements. Many structures have similar arrangements in the scene. For example, table and chair, dresser and ottoman, sofa and coffee table can be placed in a similar way. For a given set of objects we define set of possible placements $\mathcal{P}$ to place these objects in the scene. Objects in the placements are defined by their top left coordinate of the bounding box and orientation. These placements are extracted from acceptable scene examples. Objects are placed in the scene according to one of the placements which makes generated scenes more realistic.

3.3. Basic Network and State Representation

Reinforcement learning (RL) learns how to control an agent in an environment, to maximize cumulative reward. Given an observation o_t at time t, the agent performs an action a_t according to its policy π and receives a reward r_t for the current action. The policy is a mapping between state s_t, the environment's current condition to the action. The environment can be described as a Markov Decision Process, i.e., the current state of the environment fully characterizes the process. The total reward from time step t is defined as

$$R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}(s_i, a_i)\,,$$

$\gamma \in (0, 1]$ in this regard is a discount factor which indicates how important are the future rewards at the current point.

Policy based methods are based on adjusting policy in order to maximize the expected reward. We use PPO [31] with actor-critic network with shared parameters between the policy $\pi(a_t|s_t;\theta)$ and value function $V(s_t;\theta_v)$. The policy is updated with clipped policy gradient objective

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right],$$

where $\hat{A}_t$ advantage function estimator and $r_t(\theta) = (\pi(a_t|s_t;\theta))/(\pi(a_t|s_t;\theta_{old}))$ probability ratio. The final objective term includes value updates and entropy bonus to encourage exploration [31]

$$L_t(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi(a_t|s_t;\theta)](s_t)\right].$$

The value function ($L_t^{VF}(\theta)$) is updated with squared-error loss between the value function output and the total reward from time step t.

State consists of: 2D representation of the scene which includes scene boundary and objects; object existence per category indicates whether the scene has at least one object from this category, takes value from $\{0, 1\}$; object availability per category which is equal to 1 if number of objects is less than maximal allowed 0 otherwise; scene condition; step indicator.

3.4. Action Space Separation

One main concern which needs to be taken into account when using RL is dealing with a large action space. For the scene synthesis problems action space A formed from the Cartesian product of two sets O and P, $A = O \times P = \{(o, p)|o \in O \text{ and } p \in P\}$. Where O denotes the set of all objects and P denotes the set of all positions to place these objects in the scene. As a result of this, action space could grow rapidly even for a smaller number of objects and positions. Thus creating problems for RL algorithms to learn efficient policies. In our approach we limit action space with objects only, $A = O$. The positions for placing these objects are determined by a greedy algorithm, based on the current state and the object being added to the scene. More concretely, $p_i = \arg\max_{j \in P} r(s, a, j)$, p_i - position for the current object, $r(s, a, j)$ - reward function.

3.5. Reward Designing

At each step the scene generated with our algorithm is checked with respect to the following conditions: successful scene conditions ($CheckSuccessfulCondition()$), failure conditions ($CheckFailureCondition()$), constraints on the number of objects of the same type ($CheckObjectCountCondition()$). Each episode ends with a successful or failed scene generation. If after some steps the generated scene has met the failure condition, then the corresponding reward is -1. In contrast, once all of the predefined design objectives are obtained, which means a successful scene generated, then we assign the maximum reward. For both of these cases the scene generation ends and it will start again from the beginning. Otherwise, the reward is calculated depending on the type of design objectives which are achieved taking the last action.

In each step, only one type of object $tp \in O$ chosen by the RL algorithm can be placed within the scene. The objects which were already placed in the scene can be formally written as a union of substructures located in the different positions of this scene. A substructure is a part of some structure $st \in \mathcal{S}$. We first find the maximum complexity of substructures existing in the scene ($Search()$), which the current object can be grouped with, while not violating hard constraints. If there are more than one substructures with maximum complexity, then the choice is arbitrarily done between them. Then the current object is grouped with this substructure. The resulting substructure is part of

some structure $st^{'} \in \mathcal{S}$. The reward for the placing of this object depends on the complexity of the substructure we can get from this placement. The higher the resulting complexity, the higher is the reward. This encourages the RL algorithm to find optimal policies to get a higher cumulative reward. From the other side, we use only l different positive rewards not depending on the type of the structure, one for each complexity: $r_1 \leq r_2 \leq \cdots \leq r_l$, which serves to increase the variety of generated scenes. Where, $l = \max_{st \in \mathcal{S}} C(st)$ is maximum complexity among all structures. Constraints on the number of elements of the same type are applied by assigning negative rewards. Successful and failed generation conditions depend on the problem domain. All the rewards in our method are taken from the $[-1, 1]$ interval. The generation process is summarized in Algorithm 1. $PlaceObject()$ function in this scope uses one of the placement functions depending on the state, current object and complexity.

Predefined design objectives. Structures in the scenes exhibit different relationships between objects. As we quantify these relationships with numbers (rewards), we can numerically assess the current condition of the generated scene. In this paper, the sum of all rewards taken so far $\sum_{i=1}^{t} r_i$ represents the current condition of the scene. From positive examples we can get minimal value R_m for a scene to be considered successfully generated. Also, we can enforce other requirements along with minimal value, depending on the problem. These conditions follow form predefined design objectives.

Algorithm 1 Reward Assigment

Input: current state, current action, reward vector sorted in descending order

Output: reward for current action

procedure AssignReward($state, action, r[\,]$)
  if $CheckFailureCondition(state)$ **then**
    return -1
  if $CheckObjectCountCondition(state)$ **then**
    return -0.1
  $l \leftarrow length(r)$
  for $i \leftarrow 1$ to l **do**
    $c \leftarrow l - i + 1$ ▷ Complexity
    $s \leftarrow Search(state, action, c)$
    if $s.not\_empty()$ **then**
      $p \leftarrow random(1, length(s))$
      $PlaceObject(state, action, p)$
      if $CheckSuccessfulCondition(state)$ **then**
        return $1 + r[i]$
      else
        return $r[i]$
  return -0.1

3.6. Network Training

The main objective of our method is to generate a wide variety of scenes which satisfy hard constraints and at the same time have predefined properties. In each step we add an object to the scene, until we found that the final condition is reached, successful or failed scene generated. We denote this process as an episode, and the scene as an environment.

Action sampling. The objective in standard RL is to maximize $\mathbb{E}[R_t]$ expected cumulative reward. In our setting we additionally would like to increase variety of generated scenes.

We normalize $\pi(a_i|s;\theta)$ policy output (unnormalized) for a scene s, using the softmax function with temperature τ:

$$P^{\tau}(a_i|s;\theta) = \frac{e^{\pi(a_i|s;\theta)/\tau}}{\sum_{z=1}^{N} e^{\pi(a_z|s;\theta)/\tau}}\,.$$

Here, $P^{\tau}(a_i|s;\theta)$ specifies the probability of taking action a_i given state s. In order to balance between exploration and exploitation, τ is steadily decreased from 1 completely randomly to 0 greedy action sampling. And then we keep greedy action sampling until convergence during the training process.

During the inference time we sample actions with some τ value. As τ close to 1 the results show more variety but the percent of scenes which have predefined properties will be small. Conversely, for the values of τ close to 0 the results have less variety and predefined properties achieved in more scenes. We use Jensen-Shannon divergence to quantify similarity between resulting distribution and uniform distribution. Uniform distribution is chosen as a perfect case for variety, the more the resulting distribution is close to the uniform distribution the more the results show diversity. In this paper we choose τ from the following condition:

$$\tau_{optimal} = \arg\max_{0<\tau\leq 1} \min(V(\tau), W(\tau))\,,$$
$$V(\tau) = 1 - JSD(P^{\tau}, \mathcal{U})\,.$$

Here, $V(\tau), W(\tau)$ mean variety and percentage of successful scenes for fixed value of τ respectively. $JSD(P^{\tau}, \mathcal{U})$ is Jensen-Shannon divergence between resulting distribution and uniform distribution (the base 2 logarithm is used when calculating JSD, $0 \leq JSD(P^{\tau}, \mathcal{U}) \leq 1$). Actions can be sampled by uniform distribution for several starting steps in the episode when it does not affect the accuracy.

We use the PPO algorithm for several purposes: this algorithm uses both policy and value based updates, and it has been shown that this method is much faster than other reinforcement learning implementations like A2C [26], A2C with trust region [39] and TRPO [30]. We also implemented our method with A3C [26]. In both cases the agent

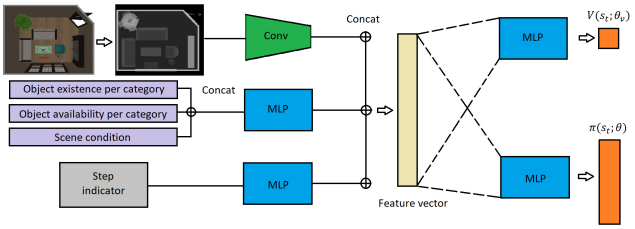


Figure 1. Network architecture for our RLSS method. State includes top-down representation of the indoor scene, per category object existence and availability in the scene, scene condition and step indicator.

learned efficient policies, but PPO required much less iterations than A3C to converge.

4. Experiments and Results

In order to evaluate our RLSS method, we address two problems: indoor planning as well as the generation of Angry Birds game levels. The initial setting is the same for all cases considered except for the number of operations.

4.1. Implementation Details

Framework. We implement our method within the ChainerRL framework [8]. In order to use it, the problem we would like to solve must be formulated as an environment. We create our own environment within the Open AI Gym [3] framework and add required functionalities to it.

Network architecture and data representation. The network architecture we use is illustrated in Fig. 1. MLP refers to fully connected layers, Conv specifies convolutional layers. Three convolutional layers are used for pre-processed input image and a fully connected layer for each of other two inputs, all of the layers are followed by ReLU nonlinearity [2]. Then three outputs are concatenated to one feature vector. This is then continued by MLPs into two final outputs one for each of $V(s_t;\theta_v)$ value and $\pi(s_t;\theta)$ policy. Each MLP consisted of two fully connected layers, first of which is followed by ReLU. The representation of the scene is consists of only the last frame. As a pre-processing, at first, we convert the scene to heightmaps [38, 7], a top-down depth rendered view of the scene, then downsize the image to 128×128 size. Object existence and availability are vectors consisting of $\{0, 1\}$ per object category, scene condition is normalized dividing by R_m and is also represented as a vector. These three are concatenated and fed to MLP. Step indicator is represented as a one-hot encoding.

Parameters. We use a discount factor of $\gamma = 0.99$, the agents update of the network parameters after every 2048 actions, minibatch size is equal to 64 and number of epochs is equal to 10. As for the optimization, we use Adam optimizer [18] with a learning rate $lr = 3 \cdot 10^{-4}$. In order to stabilize the learning process we use a reward scaling with a factor of $rs = 10^{-2}$. In our experiments, the value loss coefficient is set to $c_1 = 1$ and the entropy regularization weight is equal to $c_2 = 0.01$.

Training. We have used non-parallel implementation with only a single actor-learner. The neural network training took around 13 hours for 4 million steps. In the last million steps we applied only greedy action sampling.

4.2. Indoor Planning

In this problem, different room models are given, i.e. the geometry of the room including walls, doors, floors, and ceilings. Three different types of rooms are considered: living rooms, bedrooms, and offices. Our task is to propose a generative method for placing objects, i.e. furniture, within the room.

Our approach is to group objects based on their functionalities, and place the current object to one of its appropriate group by analyzing existing not completed groups in the room. This helps us to generate rooms which are more similar to human planned ones. For example, some groups we use for synthesizing bedrooms are: (1) bed, nightstand, floor lamp, and ottoman; (2) desk and chair; (3) dressing table and ottoman. In order to find groups of objects that appear together in the room, we use human planned room models. We also analyze the arrangements, how the group objects come together with usual placing relative to a room architecture and instantiate objects according to this. Our representation to train the network combines a top-down view of the generated scene including positions of the door and the windows.

We consider several hard constraints for this problem. First, two different objects should not share a common area (collision avoidance). Next, it should be possible for a human to move in the room conveniently. Finally, no object should block another object in the room, i.e. all objects should be accessible for a human. We assign rewards for each complexity of structures and an additional reward for some important objects for this type of the room, e.g., placing a bed in the bedroom. The additional reward is assigned only once when the important object(s) is first time placed in the room during the current episode.

Comparison. To compare our work with the state-of-the-art, we consider recent convolutional neural network based approaches [38, 24, 37, 29] and an optimization based method [40]. We compare these methods on 60 randomly taken generated examples for each type of the room (bedrooms, living rooms, and offices) using several constraints including object-with-object and object-with-room boundary collisions, object accessibility and door blocking problems; see Table. 1. Our method handles these constraints much more accurately compared to other methods. However, given the wide variety and complexity of the room boundaries, our method also allows for a small percentage

of constraint violations, especially with object accessibility related problems. For a fair comparison we applied our structure based rules on [40] and implemented their code as it was not available. It should be noted that [40] rearranges the selected set of objects in the room, which severely limits the variety. We also consider another baseline to evaluate the importance of learning. We apply Greedy Search to place objects in the scene with our rule based *Assign-Reward()* function with and without rewards. In the first case objects are sampled uniformly, in the second case objects are sampled by rewards which are calculated for each object. Results show Table. 2, that our RLSS outperforms the baseline method by large margins. Scene complexity in this table specifies the maximal complexity of the structures present in the scene averaged over tests. In general, non-learning based baselines generate scenes for smaller rooms, but failed for large rooms, also reward calculation for each object results to an increase in synthesis time.

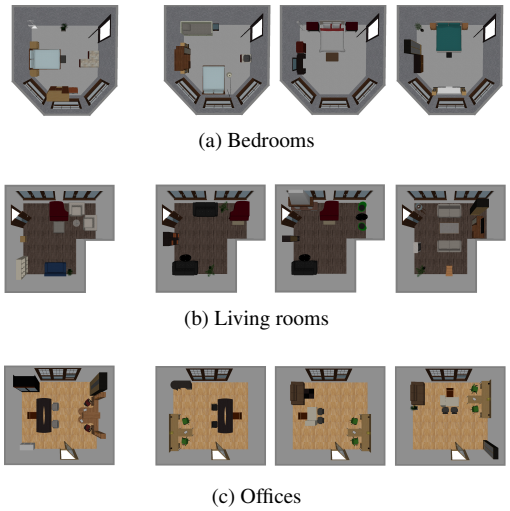
We find that handling different room boundaries is one of the fundamental limitations in indoor planning. Methods based on generating scenes for the rectangular room boundaries [24, 17, 28] or for rooms which should be encircled with walls [37] might work well for special case but might not generalize well for other cases. Moreover, using the same rendering for the synthesized results is important to compare results fairly. For these purposes, we compare our method with [29]. Visual comparison of the synthesized scenes with two different methods can be found in Fig. 3.

To evaluate the diversity of the generated scenes we consider graph kernels [6]. Fig. 2 shows similar scenes to the given example scene amongst 1000 generated scenes, for each type of the room. Scene similarity in this example is evaluated based on proximity and common function of objects (relationship) in the scene. Numerical assessment of the diversity of generated scenes based on Kullback Leibler (KL) divergence of object category distribution and uniform distribution is given in the Table 1.

Moreover, we evaluated the performance of our RLSS method and related work as illustrated in Table 1. As illustrated there, our RLSS method is very competitive when with respect to its performance. Also, our approach is not data-driven, it does not require training data and is capable of handling different room boundaries.

4.3. Level Generation

As an additional benchmark from another context, we apply our RLSS method in order to generate levels for the physics-based game Angry Birds aiming for a wide variety of stable game levels. Blocks used in the game might differ by their size, shape, and material. Some blocks can be placed in the game environment in a horizontal as well as in a vertical position. All blocks can be classified as regular and irregular blocks. We build stable structures from regular blocks and add irregular blocks into these. All regular blocks have a box shape.



(a) Bedrooms

(b) Living rooms

(c) Offices

Figure 2. Illustration of the diversity of scenes generated with our RLSS method. The leftmost image in each column represents an example image, other images are the nearest neighbors amongst 1000 generated scenes. Scene similarity is measured using graph kernels [6].

The game area where we build stable blocks comprises a 800×800 pixel rectangle. Possible actions in this problem are to place a certain block in the scene. The last block is placed onto the top of the highest block in the given position. As a hard constraint we take stability.

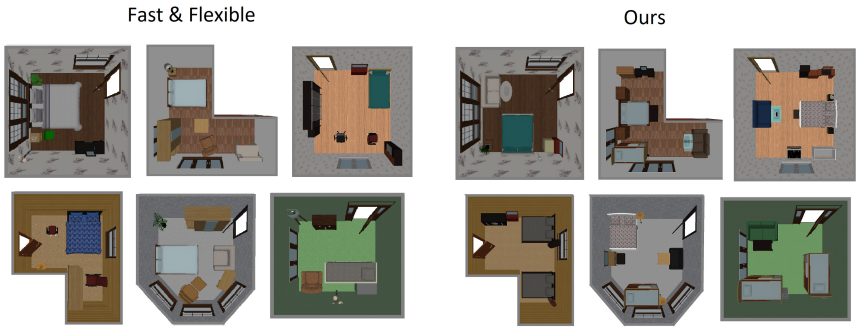
Comparison. During testing, we maintain stable structures in the memory and build levels out of these. We compare our method with the MSG v2.0 winning generator for the 2017 and 2018 Angry Birds AI (level generation) competition [34]. MSG v2.0 is based on generating the game content procedurally. This generator also generates levels from stable structures [33]. We find that MSG v2.0 takes on average about 22 seconds to generate a single level, while our method performs this task in only about 0.5 seconds. A visual comparison of the generated levels is given in the Fig. 4. Levels rendered with Science Birds [5], an open-source, Unity-based clone of the Angry Birds game. KL divergence of object category distribution and uniform distribution for MSG v2.0 and our method is approximately equal to 0.23 and 0.21, respectively. The comparison results with the non-learning baselines introduced above are presented in the Table. 2.

4.4. Quantitative Evaluations

To quantitatively evaluate results generated by our method we conducted a 2 alternative forced choice comparison on Amazon Mechanical Turk. Participants were asked to choose the most plausible scene out of two synthesized scenes placed side-by-side, one generated with our

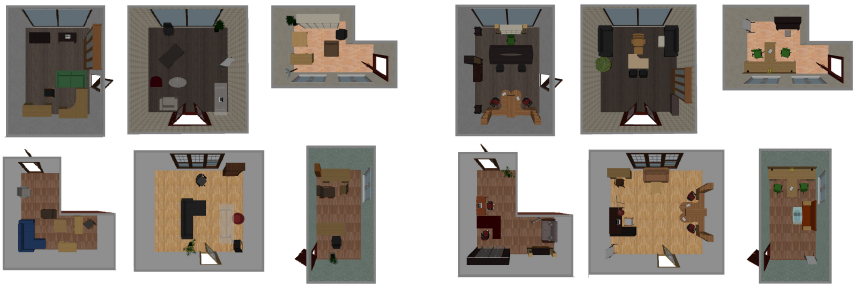
Method	Generation Time	Room Boundary	Acceptable scenes (by hard constraints)			KL divergence
			Bedroom	Living	Office	
Deep Priors [38]	$\sim$ 240 sec.	**any**	60 %	83.3 %	61 %	0.89
GRAINS [24]	**0.1027 sec.**	rectangular only	64 %	80 %	52 %	0.83
Fast&Flexible [29]	1.858 sec.	**any**	88.3 %	86.7 %	83.3 %	0.91
PlanIT [37]	$\sim$ 72 sec.	closed room boundaries	80 %	83.3 %	80 %	**0.72**
Make it Home [40]	$\sim$ 22 sec.	—	81 %	81 %	—	—
Ours	$\sim$ 1 sec.	**any**	**97.5** %	**98** %	**96** %	0.81

Table 1. Classification and performance comparison of different indoor scene generation methods. Better results or options are indicated with bold text in each column. — means no available results.



(a) Different bedroom scenes generated with Fast & Flexible (left) and our RLSS method (right).

(b) Different living room scenes generated with Fast & Flexible (left) and our RLSS method (right).



(c) Different office scenes generated with Fast & Flexible(left) and our RLSS method (right).

Figure 3. Qualitative comparison of scenes generated with Fast & Flexible [29] and our method.

RLSS method and another one with other method. For indoor planning problem each participant performed overall 63 comparison task for each method, 20 for each room type and 1 for vigilance test. Images in these comparisons represent top-down view of a scene, rendered such that all objects are visible and colored with solid colors to help participants to choose scenes by the object arrangements not by colors or other not important factors. For each task we considered answers from 10 participants, who passed all the vigilance tests. The results of this perceptual study is summarized in

Method	Accuracy		Structure complexity		Synthesis time	
	IP	**AB**	**IP**	**AB**	**IP**	**AB**
GSearch w/o r	10 %	4 %	1.5	2.7	8 sec.	5 sec.
GSearch w r	25 %	19 %	2.3	2.8	67 sec.	37 sec.
RLSS	**62 %**	**58 %**	**3.4**	**3.0**	**1 sec**	**0.5 sec.**

Table 2. Comparison with Greedy Search with and without rewards. IP refers to Indoor Planning, AB specifies Angry Birds levels. Better results are indicated with bold.

Method	Bedroom	Living	Office
Deep Priors [38]	**76.2 ± 6.0**	**68.1 ± 10.0**	**81.4 ± 5.6**
GRAINS [24]	**72.3 ± 8.5**	**78.6 ± 9.1**	**80.1 ± 5.2**
Fast & Flexible [29]	59.5 ± 10.8	**67.6 ± 8.7**	**73.8 ± 8.5**
PlanIT [37]	**69.0 ± 9.5**	**68.1 ± 11.1**	**70.0 ± 7.5**
Make it Home [40]	**64.8 ± 9.4**	**61.4 ± 8.4**	—

Table 3. Forced choice perceptual study results. Bold means our scenes are preferred with 95% confidence (± standard error), regular text means no preference. — means no available results. Higher is better.

Method	Accuracy	KL divergence
Naive PPO	0 %	1.5634
PPO+ASE+RD	28 %	0.6377
RLSS	**58 %**	**0.2144**

Table 4. Ablation study results: Accuracy and diversity of generated scenes, for our method and its variations. For the accuracy column, higher is better, while for the KL divergence column, lower is better.

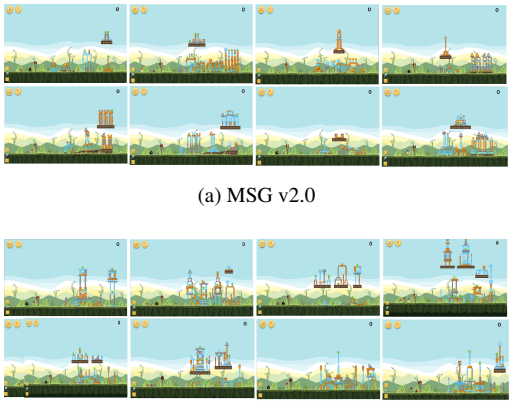
the Table 3. As it can be seen from the data, that our method shows the best results for almost all comparisons.

We conducted similar perceptual study to compare generated Angry Birds levels, with 30 images and 15 participants. The result of this comparison show that our levels were preferred with **62.2** ± **6.9** (mean ± standard error) margins with 95% confidence.

Ablation. We train the neural network without Action Separation (ASE) & Reward Designing (ASE+RD) and also without our Action Sampling (ASA) for Angry Birds level generation problem. The experiments show that Naive PPO does not learn to achieve the predefined design objectives, instead it learns to pile up box objects to maximize reward. PPO with ASE+RD and sampling the next action from the top 3 with ϵ exploration at the beginning ($\epsilon = 0.6$) does not demonstrate enough variety and accuracy Table 4.

5. Conclusion

In this paper, we proposed a reinforcement learning algorithm for scene generation called RLSS. To the best of our knowledge, we are the first to propose such an algorithm that can produce a wide variety of scenes. To this end, we modify the state-of-the-art PPO reinforcement learning algorithm to sample from a large set of possible scenes. We also propose a suitable solution for reward function design, which includes two components. Hard constraints describe scene attributes that have to be fulfilled and that are strictly enforced. Predefined design objectives describe design goals or scene configurations that make a scene more desirable. Our results show that RLSS can produce a large variety of scenes with significantly higher quality than the current state of the art.



(a) MSG v2.0

(b) RLSS

Figure 4. Illustration of several Angry Birds game levels generated with MSG v2.0 (top) and our RLSS method (bottom).

5.1. Limitations and Future Work

A limitation of our method is that it currently does not have a mechanism to combine learning from reward functions and scene examples at the same time. In our future work, if high quality scene data bases become available, we would like to investigate combinations of generative adversarial networks and reinforcement learning to tackle this challenging research problem. Also, extracting structures and implementing placement functions take additional time.

In addition, we believe that a great research direction is to extend reinforcement learning to learn scene generation from given input images. One possible approach is to design reward functions that include an estimation of the similarity of generated scenes and images. While this approach is even more difficult than learning from scene examples, it has the advantage that image data sets are much easier to come by than scene data sets.

Acknowledgements

This work was funded by KAUST through baseline funding. The valuable comments of the anonymous reviewers that improved the manuscript are gratefully acknowledged.

References

[1] Abbas Abdolmaleki, Jost Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. 06 2018.

[2] Abien Fred Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[4] Peter Dayan and Geoffrey E. Hinton. Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278, 1997.

[5] Lucas Ferreira and Claudio Toledo. A search-based approach for generating angry birds levels. In *Proceedings of the 9th IEEE International Conference on Computational Intelligence in Games*, CIG'14, 2014.

[6] Matthew Fisher, Manolis Savva, and Pat Hanrahan. Characterizing structural relationships in scenes using graph kernels. *ACM Trans. Graph.*, 30:34, 07 2011.

[7] M. Fisher, M. Savva, Y. Li, P. Hanrahan, and M. Nießner. Activity-centric scene synthesis for functional 3d scene modeling. *ACM Transactions on Graphics (TOG)*, 34(6), 2015.

[8] Yasuhiro Fujita, Toshiki Kataoka, Prabhat Nagarajan, and Takahiro Ishikawa. Chainerrl: A deep reinforcement learning library. In *Workshop on Deep Reinforcement Learning at the 33rd Conference on Neural Information Processing Systems*, Dec. 2019.

[9] Kuzman Ganchev, João Graça, Jennifer Gillenwater, and Ben Taskar. Posterior regularization for structured latent variable models. 01 2009.

[10] Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, S. M. Ali Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning, 2018.

[11] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[12] Jing Hu, Ziwei Luo, Xin Wang, Shanhui Sun, Youbing Yin, Kunlin Cao, Qi Song, Siwei Lyu, and Xi Wu. End-to-end multimodal image registration via reinforcement learning. *Medical Image Analysis*, 68:101878, 2021.

[13] Zhiting Hu, Zichao Yang, Ruslan Salakhutdinov, Xiaodan Liang, Lianhui Qin, Haoye Dong, and Eric Xing. Deep generative models with learnable knowledge constraints, 2018.

[14] Zhewei Huang, Shuchang Zhou, and Wen Heng. Learning to paint with model-based deep reinforcement learning. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct 2019.

[15] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[16] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. *arXiv preprint arXiv:1912.04958*, 2019.

[17] Zeinab Sadeghipour Kermani, Zicheng Liao, Ping Tan, and Hao (Richard) Zhang. Learning 3D Scene Synthesis from Annotated RGB-D Images. *Computer Graphics Forum*, 2016.

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[19] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013.

[20] Brenden Lake, Ruslan Salakhutdinov, and Joshua Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350:1332–1338, 12 2015.

[21] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

[22] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. 2018. cite arxiv:1805.00909.

[23] Guohao Li, Matthias Müller, Vincent Casser, Neil Smith, Dominik L. Michels, and Bernard Ghanem. Teaching uavs to race with observational imitation learning. *CoRR*, abs/1803.01129, 2018.

[24] Manyi Li, Akshay Patil, Kai Xu, Siddhartha Chaudhuri, Owais Khan, Ariel Shamir, Changhe Tu, Baoquan Chen, and Daniel Cohen-Or. Grains: Generative recursive autoencoders for indoor scenes. *ACM Transactions on Graphics*, 38:1–16, 02 2019.

[25] Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. Interactive furniture layout using interior design guidelines. *ACM Transactions on Graphics*, 30, 07 2011.

[26] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[27] Matthias Müller, Vincent Casser, Neil Smith, Dominik L. Michels, and Bernard Ghanem. Teaching uavs to race using ue4sim. *CoRR*, abs/1708.05884, 2017.

[28] Siyuan Qi, Yixin Zhu, Siyuan Huang, Chenfanfu Jiang, and Song Zhu. Human-centric indoor scene synthesis using stochastic grammar. 06 2018.

[29] Daniel Ritchie, Kai Wang, and Yu-An Lin. Fast and flexible indoor scene synthesis via deep convolutional generative models. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2019.

[30] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

[31] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[32] Shuran Song, Fisher Yu, Andy Zeng, Angel Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. pages 190–198, 07 2017.

[33] Matthew Stephenson and Jochen Renz. Generating varied, stable and solvable levels for angry birds style physics games. pages 288–295, 08 2017.

[34] Matthew Stephenson, Jochen Renz, Xiaoyu Ge, Lucas N. Ferreira, Julian Togelius, and Peng Zhang. The 2017 AIBIRDS Level Generation Competition. *IEEE Transactions on Games*, PP:1–10, 07 2018.

[35] Jerry Talton, Lingfeng Yang, Ranjitha Kumar, Maxine Lim, Noah Goodman, and Radomir Mech. Learning design patterns with bayesian grammar induction. *UIST'12 - Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, 10 2012.

[36] Burak Uzkent, Christopher Yeh, and Stefano Ermon. Efficient object detection in large images using deep reinforcement learning. *CoRR*, abs/1912.03966, 2019.

[37] Kai Wang, Yu-An Lin, Ben Weissmann, Manolis Savva, Angel Chang, and Daniel Ritchie. Planit: planning and instantiating indoor scenes with relation graph and spatial prior networks. *ACM Transactions on Graphics*, 38:1–15, 07 2019.

[38] Kai Wang, Manolis Savva, Angel Chang, and Daniel Ritchie. Deep convolutional priors for indoor scene synthesis. *ACM Transactions on Graphics*, 37:1–14, 07 2018.

[39] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *CoRR*, abs/1611.01224, 2016.

[40] Lap-Fai Yu, Sai Yeung, Chi-Keung Tang, Demetri Terzopoulos, Tony Chan, and Stanley Osher. Make it home: Automatic optimization of furniture arrangement. *ACM Trans. Graph.*, 30:86, 07 2011.

[41] Zaiwei Zhang, Zhenpei Yang, Chongyang Ma, Linjie Luo, Alexander Huth, Etienne Vouga, and Qixing Huang. Deep generative modeling for scene synthesis via hybrid representations. *ACM Transactions on Graphics*, 39:1–21, 04 2020.

[42] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016.