Learn to play Tangram with Graph-based Reinforcement Learning

Max Zimmermann[1], Dominik Marino[2], Maximilian Langer[3], Kay Hansel, Niklas Funk

***Abstract*— Robotic assembly remains a major challenge in robotics. In particular, learning an assembly order generally suffers from high combinatorial complexity. Tangram is a dissection puzzle with the goal of reconstructing a given shape using seven polygons of different shapes and sizes. Its inherent complexity has made it a famous game often played by children who therefore have to learn goal-defined geometric reasoning. Tangram naturally abstracts from simple and plausible tasks in robotic assembly. In this work, we formalize a concept to solve the game of Tangram by decentralized graph-based decision-making using a multi-head attention graph representation. For learning, we utilize a reinforcement learning setting based on a *PyBullet* simulation environment.**

I. Introduction

The principle of the game Tangram is to connect geometric puzzle stones (tokens), like, squares, parallelograms, or triangles of different sizes into a desired shape. Fig. 1 shows an example of a predefined target shape in Tangram, where all tokens need to be moved to recreate the target shape. Correct alignment of all edges without overlapping pieces is a challenging task.

In addition, finding an ordering to correctly stack the corners and edges of several tokens to its neighbors is a NP-hard problem [1], [2], [3]. To solve Tangram, one can either pursue a combinatorial or a heuristic approach. Here, a reasonable heuristic algorithm is needed for finding a relatively good ordering to connect all the tokens in a Tangram game. Especially in the domain of the robotic field, this principle can gain huge advantages in autonomous assembly for planning a target initialization for several tokens in the work space. Currently, even slight changes to the assembly target require human intervention to adjust the robot's motion plan[4], [5].

However, once robots gain capabilities to optimize assembly plans autonomously, a drastic increase in efficiency and reduction of cost can be achieved [6]. In general, the problem of autonomous robotic motion planning lives in a continuous three-dimensional space. The transition into a two-dimensional space weakens the complexity of the combinatorial problem, but it is still present. This is where the Tangram game is located.

*This work was not supported by any organization

[1] Max Z. is a computational engineering student at the Technische Universität Darmstadt, E-Mail: max.zimmermann@stud.tu-darmstadt.de

[2] Dominik M. is a autonomous systems student at the Technische Universität Darmstadt, E-Mail: dominik.marino@stud.tu-darmstadt.de

[3] Maximilian L. is a autonomous systems student at the Technische Universität Darmstadt, E-Mail: maximilian.langer@stud.tu-darmstadt.de

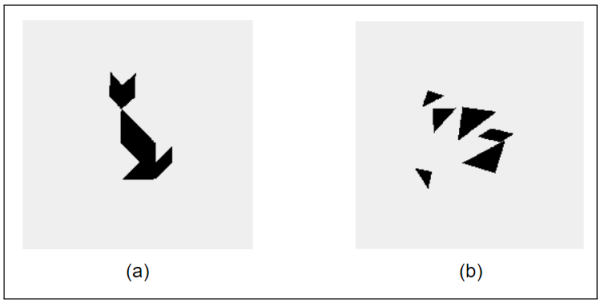

Fig. 1. An Example of solving the Tangram game by using an initialized target pattern with 6 tokens. In (a), the target shape represents a cat using all assigned tokens and (b) shows a random initial configuration state where all the token are placed on the playground.

In this regard, as it is the case for autonomous assembly, solving Tangram requires geometric reasoning under the guidance of a predefined goal. Additionally, its token-based composition motivates for graph-based approaches.

The geometric properties of the tokens can be used to build a suitable graph structure. Under the use of *graph neural networks* [7] (GNNs) we naturally incorporate a representation of the problem that depicts the need for geometric reasoning. Furthermore, we can conduct our studies under gradual increase of complexity, e.g., by varying the number of tokens present in the environment. If we are to find a proper representation by generalizing the three-dimensional problem, it benefits the research in autonomous robotic assembly.

In this work, our focus is to formalize a concept to solve Tangram based on *Reinforcement Learning* (RL) and graphs. To do this, we adjust a Multi-Head Attention graph representation [4] originally developed for autonomous robotic assembly. In contrast to deploying a decision-making module per token, we propose a decentralized decision-making strategy to achieve the global goal of correct alignment of all tokens.

We compare the performance between a *Multi-Layer Perceptron* (MLP) and a *Multi-Head Attention network* (MHA) for solving the Tangram game with a single token in a static simulation environment. We also compare different reward functions to improve the policy learning of our model.

II. Related Work

Yamada & Batagelo [2] performed a comparative study on different methods to solve Tangram. First, they evaluated a method utilizing heuristic programming proposed by Deutsch & Hayes [8]. The basic idea is to iteratively divide the target shape into multiple sub-problems until each sub-problem corresponds to a single token. Deutsch & Hayes define

10 rules using the geometric relationship of the edges to automatically dissect the target shape. Once the problem is successfully dissected into sub-problems that correspond to exactly one token each, it becomes easy to solve by trial and error. However, there is no guarantee for the algorithm to successfully dissect the problem. It is especially constrained to target shapes without holes and target tokens with specific rotations.

Another method evaluated in [2] is based on simple neural networks. Oflazer [9] proposed a representation of the position and rotation of tokens in the target shape by individual neurons. Although this approach is shown to solve common Tangram targets, it assumes the target shape being placed on a regular grid and rotations of tokens as multiples of 45 degrees.

Finally, Yamada & Batagelo reviewed two techniques originally developed for solving jigsaw puzzles. Both techniques have extensions to the game of Tangram. First, [10] utilizes genetic programming by representing tokens by string codes. Based on the string code representation and an iterative approach, a subset of target tokens are placed such that they lie inside the outline of the target shape while their boundaries coincide with the boundary of the target shape. The placed tokens now define a smaller subspace of the target shape. Further iterations try to fill the subspace using the remaining tokens. Again, this proposal assumes target shapes without holes. In [1], Kovalsky et al. represent Tangram as an edge matching puzzle. They formulate a polynomial equation system to represent a specific problem in Tangram. Their representation is based on a target shape and a definition of the corresponding tokens, assuming a fixed orientation of the individual tokens. The solution of the equation system yields the solution to the given problem. However, this approach can only deal with translations. Therefore, tokens are assumed to be given already in the final orientation. Kovalsky et al. argue that the approach can be extended to additionally support a discrete set of rotations.

Note, that all proposals evaluated in [2] pose strong assumptions on the structure of the target shape. Most important, none of the proposed techniques can solve target shapes consisting of multiple connected areas.

An early appearance of a *deep learning* (DL) model that has successfully learned control policies based on high-dimensional image input was published in [11]. The breakthrough was achieved by combining deep neural networks with the techniques of RL, introducing Deep Q-Networks. However, the solution was constrained to small discrete actions spaces.

In general, for tasks of physical control, one encounters high-dimensional continuous action spaces. Silver et al. [12] successfully proposed further advances, adjusting Deep Q-Learning to the continuous action domain. By introducing the *deep deterministic policy gradient* algorithm (DDPG), we are able to utilize deep RL to learn control policies in environments of continuous action spaces.

For the sake of generalizability, when working on optimal control problems in an RL setup, we want to avoid too strict assumptions about the agent's morphology. Instead of having the policy assume a single specific geometric shape when deciding on an action, we want to learn a global policy that enables the agent to take well-informed steps. In this regard, Huang et al. [13] proposed to create a global policy based on a collection of *shared modular policies* (SMP). Consider an environment with a robot agent of some morphology. The basic idea is to deploy an SMP per actuator of the robot. By utilizing message passing between distant SMPs, a policy can be learned that masters complex coordination, independent of the specific morphology of the agent.

In graph representations, it can be beneficial for individual nodes to learn different relations for different neighbors. [14] introduced the attention mechanism [15] for use in GNNs. Exploiting attention, individual nodes are able to efficiently learn different weights for different nodes present in the graph.

III. Background

Based on a standard RL setting, we consider an agent acting in an environment modeled as *Markov decision process* (MDP). For each discrete time step, t the agent observes a state s_t and performs action a_t. Carrying out $a_t \in \mathbb{R}^N$ yields a scalar reward $r_t \in \mathbb{R}$ and a new state s_{t+1}.

A. Deep Deterministic Policy Gradient

Following the DDPG algorithm, we concurrently learn a policy and a Q-function. DDPG is motivated by the fact that for any state s, given the optimal value-function $Q^*(s,a)$, the optimal action $a^*(s)$ can be computed by solving

$$a^*(s) = \underset{a}{argmax} Q^*(s,a) \; . \tag{1}$$

DDPG specifically works on continuous action spaces for which solving Eq. (1) becomes infeasible. This problem is overcome by approximation based on a policy $\pi(s)$, mapping states s to a probability distribution over actions a, yielding

$$\underset{a}{max} Q^*(s,a) \approx Q(s,\pi(s)) \; . \tag{2}$$

Learning exploits a replay buffer of previous observations. Observations in the replay buffer are sets (s,a,r,s',d) of state s, action a performed in s, leading to new state s' and yielding reward r. A boolean value d is stored to encode whether state s' is terminal. DDPG follows an actor-critic scheme. The critic is a neural network $Q_\phi(s,a)$ parameterized by ϕ that learns to approximate $Q^*(s,a)$. Updating the network parameters ϕ is based on minimizing the *mean-squared Bellman error* (MSBE)

$$Q_\phi(s,a) = \underset{\phi}{argmin} \underset{(s,a,r,s',d)\sim\mathbb{D}}{\mathbb{E}} \left[\left(Q_\phi(s,a) - y_i \right)^2 \right] \tag{3}$$

with observations (s,a,r,s',d) randomly sampled from the replay buffer $\mathbb{D}$ and y_i as the target. To avoid Eq. (3) directly depending on the learned parameters through y_i, a target policy $\pi_{\theta_T}(s)$ and a target Q-function Q_{ϕ_T} are introduced. Initially, parameters θ_T and ϕ_T are equal to the parameters

θ and ϕ of the respective main networks. Following the main network updates, target network parameters are updated by Polyak averaging

$$\begin{aligned}\phi_T &\leftarrow \rho\phi_T + (1-\rho)\phi \\ \theta_T &\leftarrow \rho\theta_T + (1-\rho)\theta\end{aligned} \tag{4}$$

with hyperparameter ρ. Now targets y_i can be defined as

$$y_i = r_i + \gamma Q_{\phi_T}(s_i', \pi_{\theta_T}(s')) \tag{5}$$

with γ as discount factor.

The actor is also a neural network $\pi_\theta(s)$. We learn parameters θ by maximizing the expected Q-function. Therefore, we solve

$$\pi_\theta(s) = \max_\theta \mathop{\mathbb{E}}_{s\sim\mathbb{D}} \left[Q_\phi(s, \pi_\theta(s))\right]. \tag{6}$$

B. Hindsight Experience Replay

In order to learn how to act properly, an agent in RL requires some feedback signal on performance. Typically, this signal is received in the form of a reward r after execution of an action a in some observed state s. However, especially for problems with continuous state and action spaces, positive rewards are sparse. As a consequence, the agent rarely receives a signal indicating proper behavior which makes learning extremely difficult. *Hindsight experience replay* (HER) [16] tackles this problem by additionally utilizing modified versions of undesirable outcomes. Let s_{target} be the target state. Beginning in state s_0 the agent successively performs actions $a_{0:N-1}$ until it observes terminal state s_N. This yields observed trajectories

$$\begin{gathered}\{(s_0, a_0, r_0, s_1, s_{target}), \\ (s_1, a_1, r_1, s_2, s_{target}), \\ ..., \\ (s_{N-1}, a_{N-1}, r_{N-1}, s_N, s_{target})\}.\end{gathered}$$

Whenever the set of possible targets becomes very large, many observed terminal states do not match. These observations alone do not help to find good policies. As a solution, all experiences along the observed trajectory are modified such that their target state is equal to the observed terminal state. The modified trajectory becomes

$$\begin{gathered}\{(s_0, a_0, r_0, s_1, s_N), \\ (s_1, a_1, r_1, s_2, s_N), \\ ..., \\ (s_{N-1}, a_{N-1}, r_{N-1}, s_N, s_N)\}.\end{gathered}$$

Hence, the hindsight trajectory represents a perfect play by the agent. Adding both, the modified and original trajectories to the replay buffer ensures the presence of samples with positive reward signals. Consequently, when sampling from the replay buffer during the learning phase, the probability of encountering useful feedback signals is increased.

C. Multi-Head Attention Graph Representation

In this work, we closely follow the MHA graph representation proposed in [4]. Originally developed for the problem of autonomous assembly, the general idea is to use graph-based environment state representations as input to a GNN. Based on attention, the network is able to efficiently learn different weights for different nodes in the graph. This approach enables flexibility regarding the representation of the problem at hand.

Under the assumption of a Markov decision process, we formalize the graph $G = (N, E)$ with nodes N and edges E. Nodes N encode the state of the environment. Aiming to exploit the network's ability to treat the neighbors of each node individually, N is a set of encoded states of differing nodes. The precise distinction of nodes is problem-specific. In particular, this node representation allows for explicit and implicit encoding of each node's role in the current state of the environment. For instance, characterization of constant and variable nodes in the same graph is possible.

Let C be the set of classes of nodes we distinct between. Then we have $N = \{N_c | c \in C\} = \{n_i | i = 1..N\}$.

Edges E define connections between nodes in the graph, represented by an adjacency matrix. The entry $E(i,j)$ of the adjacency matrix describes the connection between nodes n_i and n_j and is given by

$$E(i,j) = \begin{cases} 1, & \text{if } n_i \text{ and } n_j \text{ are connected} \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

We utilize the graph state representation as input to the MHA architecture and project it into a higher dimensional space. Using a *fully-connected layer* (FC) as well as a ReLU activation function, we define function $g(n) = ReLU(FC(n))$ and apply it to all nodes n_i . We get

$$n_i^{(1)} = g(n_i^{(0)}) = ReLU(FC(n_i^{(0)})), \tag{8}$$

with $n_i^{(0)}$ corresponding to the initial nodes of the graph. The higher dimensional node embedding is further processed by multiple rounds of message passing. For each round l of message passing we pass the previous embeddings $\mathscr{N}^{(l-1)}$ of all nodes through a MHA layer with M heads. For each head we compute a key $k_i = W_k n_i^{(l-1)}$, a query $q_i = W_q n_i^{(l-1)}$ and a value $v_i = W_v n_i^{(l-1)}$. W_k, W_q and W_v are weights special to each head. We can then compute the output message of each head by

$$m_i = \sum_{j \neq i} \alpha_{i,j} v_j, \tag{9}$$

with $\alpha_{i,j}$ as attention weights of the connection from node n_i to n_j. To compute the attention weights, we first need the compatibility scores $c_{i,j}$ for the corresponding node connection, which are computed by

$$c_{i,j} = \begin{cases} \frac{1}{d} q_i^T k_j & E(i,j) = 1, \\ -\infty & \text{otherwise} \end{cases} \tag{10}$$

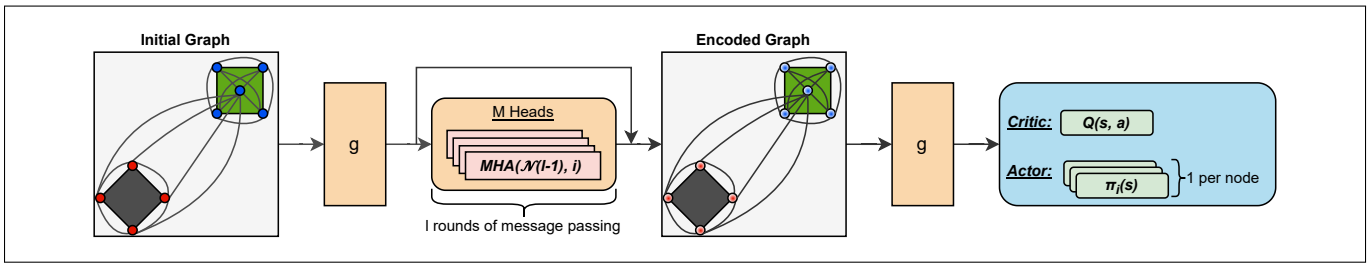

Fig. 2. Multi-head attention graph representation for Tangram with a single token. The initial graph contains all 2D corner and base positions of the tokens. In a first step, the graph is projected into a higher dimensional space. Multiple rounds of message passing yield an encoded version of the graph with a meaningful representation of the geometries. Based on this encoding, the agent can take informed actions.

with normalization constant d. The attention weights $\alpha_{i,j}$ are now computed by

$$\alpha_{i,j} = \frac{e^{c_{i,j}}}{\sum_{j' \neq i} e^{c_{i,j'}}} \, . \tag{11}$$

Finally, the MHA layer's output per node is a weighted sum of the messages of all heads

$$\mathrm{MHA}(\mathcal{N}^{(l-1)}, i) = \sum_{o=1}^{M} W_{m,o} m_{i,o} \, , \tag{12}$$

with unique weights W_m for each head. Using a skip connection $h(f(x)) = x + f(x)$, we compute node embeddings of round l by

$$n_i^{(l)} = h(g(h(\mathrm{MHA}(\mathcal{N}^{(l-1)}, i)))) \, . \tag{13}$$

We can view the overall result of message passing as an encoded version of the original graph. At this stage, the encoded graph should inherit a meaningful representation of the geometries, enabling the agent to take informed actions. Fig. 2 gives an overview of the MHA graph representation for Tangram.

IV. Proposed Method

As stated earlier, the proposed method in this work closely follows [4]. The fundamental difference lies in an adjusted version of the multi-head attention graph representation to match our problem.

The task we aim to solve is to reconstruct a randomly sampled target shape of non-overlapping tokens. For reconstruction, a learning agent has access to the same tokens that shaped the target. It should learn proper positioning of these tokens by moving them from some initial spawn point in the environment.

A. Graph Representation

We want to create a graph-based representation of the state of the environment (see Fig. 2). This graph is formalized as $G = (N, E)$ with nodes N representing characteristic points of the tokens and edges E depicting pairwise connections between the nodes. At every time step, we represent the state as a set of nodes s, to which an adjacency matrix, modelling the edges, is appended.

We consider both, the target and the tokens available to reconstruct it, as part of the state. However, we want to introduce some distinctions: First, targets are fixed structures, which means we cannot perform any actions on them. Second, for each simulation, the agent is granted only a single action per non-target token. After a token has been moved, we consider it placed for the rest of the simulation. Consequently, we get a natural distinction between 3 classes of nodes:

$$\{n_{\text{target}},\ n_{\text{token, placed}},\ n_{\text{token, unplaced}}\}$$

To capture all 3 classes, we define the following representation. Each node is represented by its x- and y-coordinate, as well as two boolean flags indicating whether it is part of the target and whether the node belongs to a placed token. Target nodes are always encoded as placed. In total, there are 4 features per node.

In deciding which parts of the token are represented as nodes, we differentiate between target and non-target tokens. Equal to both, we include their corners as nodes in the state. For non-target tokens, we include an additional node representing their base, i.e., their center, as can be seen in Fig. 2. This node is regarded as base node and will later bridge the connection to other tokens. Furthermore, all actions computed will refer to these base nodes. In total, the size of the state equals the number of corners of all tokens plus a single node per non-target token.

The graph-based representation allows us to define connections between nodes, encoded by the edges E of the graph. We stay consistent with [4] and define the adjacency matrix using Eq. (7). The connections are modeled as follows: Within each token, all corners are connected with each other. For non-target tokens, all corners are additionally connected to the base node. To enable information passing between tokens, we connect all base nodes with all other target nodes. Note, that we do not connect nodes to themselves, i.e. $E(i,i) = 0$. See Fig. 2 for an illustration. Nodes and adjacency matrix are stacked together in a single state matrix and passed to the learning algorithm.

B. Reward Signal

The reward signal is based on the overlap between the images of the current state and the target, as demonstrated in Fig. 3 (see Sec. IV). We apply the normalized sum of overlapping pixels between both images

$$r_{sparse} = \frac{\sum(px_{target} \wedge px_{state}) = 1}{\sum px_{target} = 1} \tag{14}$$

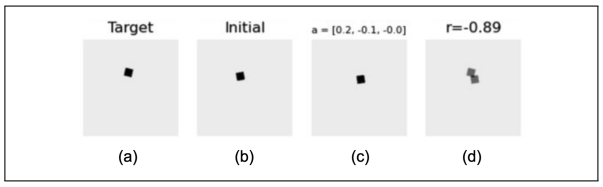

Fig. 3. Image states of the *PyBullet* Simulation, where (a) is the target state, (b) the initial randomly sampled token position, (c) the token state after action a is performed and (d) the final overlap of target and token state. Reward r is the normalized sum of overlapping pixels.

which is then normalized between $\{-1,+1\}$. Note, that this reward signal only indicates strong positive behavior if tokens and target are already very close. To combat this sparse reward, we for one employ HER (see Sec. III), but also formulate a second, dense reward signal. This reward is based on the distance of the center points between targets and tokens:

$$r_{dense} = \left\| c_{token} - c_{target} \right\|_2 \tag{15}$$

We scale the reward to output a suitable signal across all possible distances within the state space, and additionally use a logarithmic-based activation to steepen the curve at smaller distances to induce predictions that are close to the target.

C. Actor-Critic

As described in Sec. III, we are making use of a DDPG algorithm with an actor-critic scheme. Consequently, we need two slightly differing network architectures for the actor and the critic network. In general, both architectures resemble the MHA architecture. The task of the actor is to output actions $a(s) \in \mathbb{R}^3$, depending on the current state s. Actions describe token movements as velocities in x- and y-direction for translation, as well as an angular velocity around the z-axis for rotation. The choice of velocity predictions is arbitrary and makes it easy to interact with the *PyBullet* simulation environment. We can feed the network the state matrix in the representation described above without further adjustments. Note that we can only apply actions to the base node of a token, and only if the token has not been placed yet. Applying actions to the base node of a token is motivated by the fact, that every token has a center point which we can consistently use as anchor point. To keep it simple, the actor computes actions for all nodes, out of which we extract the corresponding action. Additionally, actions are normalized to the size of the state space using an *tanh*-activation to ensure that a token can reach all possible configurations in the state space within a single simulation step.

The critic's task is different in that it has to compute values $Q(s,a) \in \mathbb{R}$ depending on both, the current state s and the action a proposed by the actor. We perform two adjustments compared to the actor: First, we condition on action a by appending it to its respective base node in our state representation. Second, instead of computing a value per node, we compute a global feature and output a single value only.

V. Simulation Environment

In the following section, we describe how we simulate the Tangram game using a *PyBullet* environment. The main tasks of the simulation environment are computing executable actions and moving the tokens in the state space (see Fig. 3). Furthermore, it is used for the agent to access additional and exact information as token centers and corners to build the state representation. By exploiting exact information, we remove the impact of errors naturally inherited by state estimations. In our current simulation there are different types of tokens available, consisting of parallelograms, a square, and triangles.

The simulation is initialized, firstly, by sampling token positions and orientations from a uniform distribution parameterized by the state space. Secondly, the coordinates of the corners and the centers of all tokens are stored as the target state. Finally, all token positions and orientations are resampled by new uniform distributions, which gives the initial state for the Tangram game. During initialization, collision detection is applied to avoid overlap between tokens.

Generating the current state representation is done by using *PyBullet*'s internal methods. Originally three-dimensional, the height of the tokens is ignored, as overlapping tokens are not allowed. Sec. IV describes the structure of the state matrix, with the individual representations of the target and the current position initialization. This matrix representation is used to define the nodes of the MHA graph representation.

A virtual camera captures the target state and the current state as a grayscale image, which are used to compute the reward for the learning algorithm. For more details, see Sec. IV and Eq. (14).

VI. Experimental Result

In this section, we empirically evaluate the proposed MHA architecture employing the methods proposed in the previous sections. To verify the utility of our proposal, we compare the performances of the MHA architecture against a simple MLP acting as a baseline. We also compare different approaches to verify our methods and to evaluate possible accuracy improvements.

Following the DDPG algorithm (see Sec. III), we base our evaluation on the mean reward on 100 randomly generated simulations. For all experiments, if not specified otherwise, the model was trained using the sparse reward based on the overlapping pixels of moved token and target (see Eq. (14)). Additionally, all experiments were conducted on an environment with a single square-shaped token.

A. Training environment

Training was done on the TU Darmstadt's Lichtenberg high performance computing cluster. Each architecture was trained on 3 different random seeds and a maximum of 72 hours or 100k episodes (see. TABLE I).

For training on GPUs, we want to exploit as much parallel computation capabilities as possible. However, as a consequence of the serial nature of the agents' observations, full

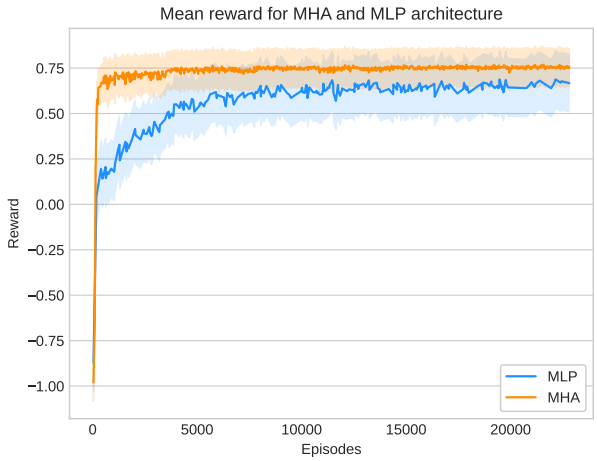

Fig. 4. Mean reward and standard deviation over 24k episodes with MHA and MLP architecture based on 3 different random number seeds. 100 tests were evaluated in each episode.

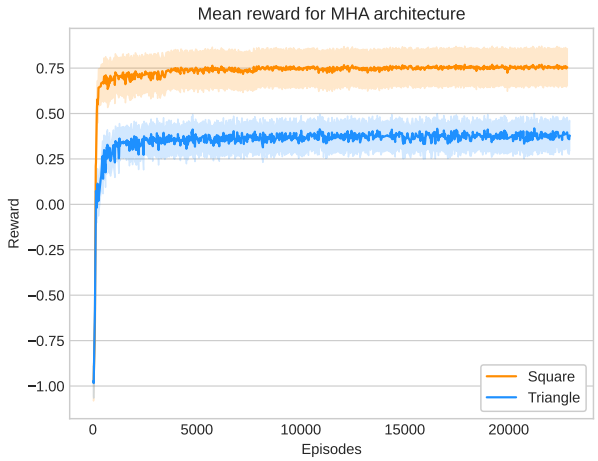

Fig. 5. Mean reward and standard deviation over 24k episodes, with 100 tests per episode for MHA architecture trained on square and triangle.

parallelism is impossible. Therefore, we stick to a Master-Worker pattern. We initialize a single master running on GPU and multiple workers running on CPU. By using multiple workers, we are running simulations in parallel, even though the progress of individual simulations is serial. The task of each worker is to accumulate observations of the agent by acting in the environment. Here, each worker has its own copy of the same agent. Observations are stored in a replay buffer of finite size. The master randomly samples batches from the replay buffer and performs agent updates based on DDPG. After every update, the parameters of each workers' agent are updated accordingly. Consequently, throughout training, observations stored in the replay buffer increase in quality.

B. Performance of MHA architecture vs. MLP architecture

Fig. 4 shows that the mean reward in case of the MHA architecture surpasses the mean reward of the MLP architecture. The MHA attention model is able to reliably reach 75% overlap of token and target after approx. 1500 episodes, while the MLP architecture reliably reaches 70% overlap after approx. 19k episodes. Additionally, the maximum mean reward of the MLP model is exceeded by the MHA model after a few hundred episodes. Finally, the variance over single episodes is much lower in the MHA setting. However, the MHA model reaches its plateau after about 5k episodes. This is consistent with the empirical observation that correct translation of the token is learned quickly, while correct rotation is not. One can conclude, that the mean reward does not increase after 5k episodes, as the model does not manage to correctly orientate the token.

It is important to note that there is a significant difference in complexity between the MHA model and the MLP model (see. TABLE I). Still, after 100k episodes of training the MLP model, no improvements can be observed. Motivated by this observation, we also train a more complex version of the MLP model with a number of parameters close to the MHA architecture. However, the more complex version does not improve in terms of mean reward.

C. Solving orientation by applying a sequence of actions

As mentioned in Sec. VI-B no model presented learned correct rotation of the token with a single action. Therefore, we test whether applying multiple actions in sequence predicted by a MHA model trained on single actions improves performance. As before, we evaluate the performance by the mean reward over 100 single simulations. TABLE II shows that the mean reward does not increase when applying up to 5 actions in sequence. This emphasizes the problems of our learned model to predict correct rotations of the token, as center points of target and token are already aligned after the first action. However, the tested model is trained on single actions only. Due to time restrictions, we present no tests on a model trained on multiple actions and reserve this test for future work.

Model Architecture	# Parameters	Run Time	# Episodes
MHA	504k	72h	28k
MLP	14k	48h	100k

TABLE I

COMPARISON OF COMPLEXITY AND LENGTH OF TRAINING PHASE FOR MLP AND MHA ARCHITECTURES.

	Mean reward $\bar{r}$ over 100 tests after n actions				
n	1	2	3	4	5
$\bar{r}$	0.758	0.754	0.754	0.764	0.750

TABLE II

MEAN REWARD OVER 100 INDIVIDUAL TEST SIMULATIONS FOR 1 TO 5 ACTIONS EXECUTED IN SEQUENCE AND PREDICTED BY A MHA MODEL TRAINED ON SINGLE ACTIONS.

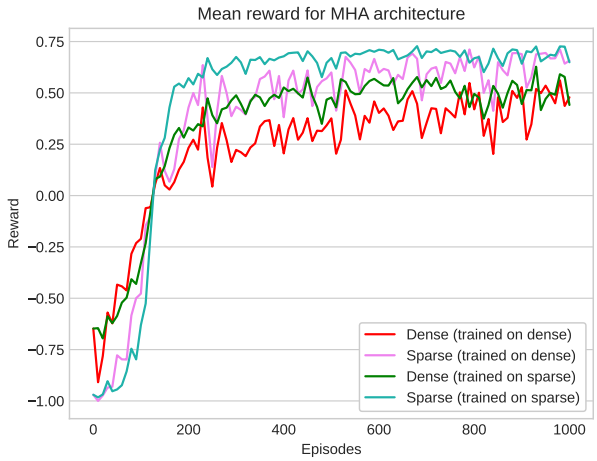

Fig. 6. Mean reward based different reward functions. Sparse reward is the normalized sum of overlapping pixels between target and token. Dense reward is distance of center of target and token.

D. Influence of polygon shape on learning orientation

We can observe, that correct translation is learned significantly better than correct rotation when training with a square-shaped token. One hypothesis is, that the pixel overlap based reward is high, even if the orientation of token and target do not match. Consequently, we conduct tests with a token of right-angled shape, where the number of overlapping pixels is significantly reduced with orientation error. Based on the MHA architecture, we compare the performance of training with a square-shaped token and training with a right triangle shaped token. Here, the mean reward for the triangle is significantly lower than the mean reward for the square (see Fig. 5). However, we can observe that correct positioning is successfully learned for both token. In both cases, correct rotation still appears to be difficult. This implies, that getting high rewards by correct rotation is a more difficult task for the triangle than for the square.

E. Sparse reward vs. dense reward

Sparse reward signals indicating proper behavior lead to slower training. We evaluate whether a dense reward signal based on the distance of the centers of target and token leads to faster learning of correct translation (see Eq. (15)). It is worth noting, that this formulation of a dense reward signal does not emphasize correct orientation.

Based on the MHA architecture, we train models on the sparse and dense reward. Each model is evaluated on both rewards, with 100 tests per episode. Fig. 6 shows, that training based on the sparse reward leads to higher mean rewards evaluated on both reward formulations. We can conclude, that using our formulation of a dense reward does not improve learning speed and performance. One hypothesis is, that decreasing the distance of the centers of target and token leads to signals indicating proper behavior even if the token does not overlap the target. On the other hand, using the sum of overlapping pixels strongly emphasizes that token and target should be correctly aligned. In the beginning of training, problems of sparse reward signals are overcome by utilizing HER.

VII. Conclusion

In this work, we present a graph-based approach to solve the game of tangram. We follow the MHA graph representation proposed by [4]. Adjusting the state representation to match our goals allows us to utilize the proposal for learning how to properly position a single token to reconstruct a target shape. The *PyBullet* simulation environment increase our control by allowing the agent to access exact observations instead of relying on state estimations. We show that the MHA architectures performs better than the MLP architecture. However, both architectures are not capable of solving the orientation problem for a single token. Since correct translation is successfully learned, the MHA architecture might also learn correct rotation when trained on a higher number of episodes. Moreover, one can expect that a reward signal with stronger emphasis on correct orientation can lead to performance improvements.

Furthermore, in our evaluations, performing multiple actions in the simulation environment does not lead to any improvements. Nonetheless, when the environment includes multiple tokens and collisions are possible, there must be multiple successive actions. Consequently, we can conclude that in the case of multiple tokens, the model should be trained on multiple actions.

VIII. Future Work

Our proposed approach does not successfully learn correct rotation of a token. For this problem, a new formulation of a reward with strong emphasis on correct rotation might lead to performance improvement.

Currently, our approach is only designed for moving one token. In future work, this can be extended to multiple tokens. Ultimately, we want a learning algorithm to solve the game of Tangram for the original setup with seven tokens.

Additionally, when approaching multiple tokens, it is unclear how to define a suitable sampling strategy of target shapes close to the real distribution of target shapes in tangram. This problem gets increasingly complex with increasing number of polygons in the environment. One approach can be to sample target shapes from a two-dimensional uniform distribution. In this case, it is possible to introduce a curriculum learning approach by shrinking the covariance over time. In the beginning of learning, this strategy will produce targets of unconnected polygons. Ultimately, with a shrinking covariance, the polygons will begin to connect.

Lastly, we exploit information only accessible through our *PyBullet* simulation environment. Using this additional and exact information is simplifying the problem. To represent the state of the simulation, we include the corner coordinates of each polygon, even for polygons building the target shape. In our simulation environment, we can access the corresponding coordinates directly. Ideally, we want to solve the problem with grayscale images as the

only source of information. However, converting images into our proposed state representation requires reliable corner detection. We can expect a drastic increase in difficulty as the state representation loses some of the corner nodes in case of connected surfaces. Having access to all corner nodes of each polygon in the target shape gives strong geometric cues to the agent on where to position which polygon. Additionally, we now face the risk of false positives when detecting corners. As a consequence, errors are introduced to the state representation.

ACKNOWLEDGMENT

This paper and the research behind would not have been possible without the support of our supervisors Kay Hansel and Niklas Funk. Their experience and knowledge have been an inspiration and kept this work on track. Without their provision of their preliminary work, this work would have been much more difficult. This thanks also goes to Janosch Moos. We would also like to thank the Technical University of Darmstadt for providing the hardware during this research. We are also grateful to all the people for their comments on an earlier version of the paper.

REFERENCES

[1] S. Z. Kovalsky, D. Glasner, and R. Basri, "A global approach for solving edge-matching puzzles," 2014. [Online]. Available: https://arxiv.org/abs/1409.5957

[2] F. Yamada and H. Batagelo, "A comparative study on computational methods to solve tangram puzzles," 10 2017.

[3] S. Z. Kovalsky, D. Glasner, and R. Basri, "A global approach for solving edge-matching puzzles," 2014. [Online]. Available: https://arxiv.org/abs/1409.5957

[4] N. Funk, G. Chalvatzaki, B. Belousov, and J. Peters, "Learn2assemble with structured representations and search for robotic architectural construction," in *5th Annual Conference on Robot Learning*, 2021. [Online]. Available: https://openreview.net/forum?id=wBT0lZJAJ0V

[5] A. George, A. Bartsch, and A. B. Farimani, "Minimizing human assistance: Augmenting a single demonstration for deep reinforcement learning," 2022. [Online]. Available: https://arxiv.org/abs/2209.11275

[6] R. B. Grando, J. C. de Jesus, V. A. Kich, A. H. Kolling, R. S. Guerra, and P. L. J. Drews-Jr, "Deterministic and stochastic analysis of deep reinforcement learning for low dimensional sensing-based navigation of mobile robots," 2022. [Online]. Available: https://arxiv.org/abs/2209.06328

[7] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, 2009.

[8] E. S. Deutsch and K. C. H. Jr, ""a heuristic solution to the tangram puzzle."." In: Machine Intelligence 7. Bernard Meltzer and Donald Michie (1972), pp. 205–240. Edinburgh University Press.

[9] A. C. Approach, "Solving tangram puzzles," 1993.

[10] D. Bartoněk, "A genatic algorithm how to solve a puzzle and its using in cartography," *Acta Chiropterologica*, vol. 11, 2005.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013. [Online]. Available: https://arxiv.org/abs/1312.5602

[12] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015. [Online]. Available: https://arxiv.org/abs/1509.02971

[13] W. Huang, I. Mordatch, and D. Pathak, "One policy to control them all: Shared modular policies for agent-agnostic control," 2020. [Online]. Available: https://arxiv.org/abs/2007.04976

[14] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2017. [Online]. Available: https://arxiv.org/abs/1710.10903

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[16] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," 2017. [Online]. Available: https://arxiv.org/abs/1707.01495