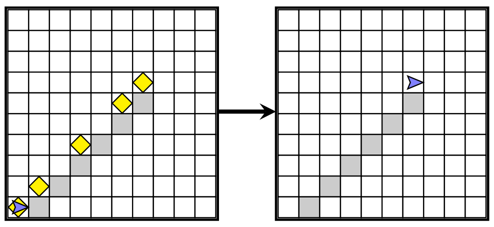
# Hands-on Project for IJCAI'21 Tutorial: Reinforcement Learning for Education

Presenter: Adish Singla (adishs@mpi-sws.org)

## 1 Background

Visual programming environments such as Scratch [1] and Karel [2] are used nowadays to introduce programming concepts to beginners. In this context, it is important to be able to generate new tasks [3, 4] as well as synthesize solution codes for any given task [5, 6]. In this project, we focus on the problem of automatically synthesizing solution codes for visual programming tasks [5, 6]. Next, we describe the elements of a Karel programming task.



(a) $T$

```
def RUN(){
  WHILE(no pathAhead){
    IF(marker){
      pickMarker
    }
    turnLeft
    move
    turnRight
    move
  }
  pickMarker
}
```

(b) $C$

Figure 1: A typical programming task from the Karel environment. $C$ is the solution code of task $T$. The structure of code $C$ includes a WHILE loop with a nested IF.

**Description of visual Karel tasks** $(T)$**:** The visual task in Karel comprises of a Pre-grid and a Post-grid (see visual grids in Fig.1a). The elements of a grid are:

- AVATAR: A Karel AVATAR is characterized by it's current location and orientation. It's orientation can be one of {NORTH, EAST SOUTH, WEST} and it's location can be any of cells of a grid. The blue dart in Fig.1a depicts the location and orientation of the AVATAR.
- MARKER: These are represented as the yellow diamonds in Fig.1a. The AVATAR can pick MARKERS from cells or put MARKERS in them.
- WALL: These are the grey cells in Fig.1a. These cells cannot be accessed by the AVATAR.
- EMPTY GRID CELL: These are the free white cells in Fig.1a. The AVATAR can freely move about these cells.

**Solution code** $(C)$**:** The solution code of the visual task, when executed, transforms the Pre-grid to the Post-grid. The codes are based on the Karel programming language. This language includes different programming structures such as WHILE, REPEAT, IF, ELSE, etc. The Boolean conditionals can take values such as `pathAhead`, `marker`, `noMarker`, etc. And finally, the language includes basic action blocks such as `move`, `turnLeft`, `turnRight`, `pickMarker`, and `putMarker`. The solution codes using these structures can be made as complex as desired.

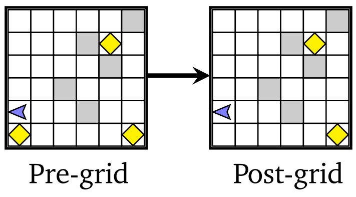
## 2 Project Details

Automatically synthesizing solution codes for arbitrary Karel tasks is challenging (see [5, 6]). In this project, we consider "simple" Karel tasks whose solution codes only use basic actions blocks as described below.

### 2.1 Objective

The objective of this project is to synthesize solution codes for simple Karel tasks using Reinforcement Learning (RL) techniques. For the purposes of this project, we restrict the solution codes of the tasks to include only the following basic actions blocks:

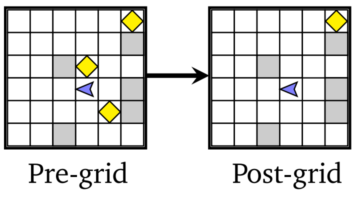
- `move`: This moves the AVATAR one grid-cell in the direction it is currently oriented in. If the AVATAR hits a WALL or the outside boundary, then the AVATAR "crashes" and the program terminates.
- `turnLeft`: This orients the AVATAR in the direction left of its current orientation.



(a) $T_1$

```
def RUN(){
  turnLeft
  move
  pickMarker
  turnLeft
  turnLeft
  move
  turnLeft
}
```

(b) $C_1$



(c) $T_2$

```
def RUN(){
  turnRight
  move
  pickMarker
  turnRight
  move
  turnRight
  move
  move
  pickMarker
  turnRight
  move
  turnRight
  move
  turnLeft
}
```

(d) $C_2$

Figure 2: Examples of generated tasks with their solution codes, $(T_1, C_1), (T_2, C_2)$. $C_1$ is the solution code of task $T_1$ and $C_2$ is the solution code of task $T_2$.

- `turnRight`: This orients the AVATAR in the direction right of its current orientation.
- `pickMarker`: This removes a MARKER from the current location (grid-cell) of the AVATAR, if present. If no MARKER is present, then the AVATAR receives an "error" and the program terminates.

## 2.2 Milestones

The following milestones serve as guidelines in achieving the project objective.

1. **Generating data:** We begin by generating data in the form of the visual tasks, for which solution codes will be synthesized. Generate $10,000$ tasks and split them into *training* ($70\%$), *validation* ($15\%$), and *test* ($15\%$) sets. The size of a grid is $6 \times 6$. We further simplify the task generation process by following the steps below for each task:

   **Pre-grid generation steps:**

   - First, randomly sample the location of WALL cells in the Pre-grid. Restrict the number of WALL cells to $5$.
   - Next, randomly sample the initial position of the AVATAR on the Pre-grid which includes its location and orientation. Sample the initial location from cells not containing WALL. The orientation must be sampled from the set {NORTH, EAST, SOUTH, WEST}.
   - Finally, randomly sample the locations of MARKER on the Pre-grid from cells not containing WALL. Restrict the number of MARKER cells to $3$.

   **Post-grid generation steps:**

   - Begin by initially setting the Post-grid to be exactly the same as the Pre-grid.
   - Sample $x \in \{1, 2\}$. Next, randomly remove $x$ number of MARKER from the Post-grid. This is the final Post-grid.

   $T_1$ and $T_2$ in Fig.2 are two examples of visual tasks generated using the above process. In the milestones below, you will train an RL agent who can synthesize solution codes for these tasks (see $C_1$ and $C_2$ in Fig.2).

2. **Setting up the RL environment:** We consider an episodic RL setting where each episode corresponds to one of the tasks. An episode for the RL agent begins by sampling a task (comprising of Pre-grid and Post-grid) from the generated data. Within an episode, at any given time step $\tau$, we refer to the current configuration of the AVATAR's grid as Curr-grid; at the beginning of the episode, Curr-grid := Pre-grid. The environment's state $s_\tau$ at time step $\tau$ is given by the tuple (Curr-grid, Post-grid). The agent takes an action $a_\tau$ from the action-set {`move`, `turnLeft`, `turnRight`, `pickMarker`}. The episode continues with actions being taken by the agent at every time step $\tau$, and modifying the Curr-grid. The episode ends when either the Curr-grid is the same as the Post-grid, or the program terminates (i.e., the AVATAR "crashes" or receives an "error") , or the number of time steps $\tau$ reaches $100$. The agent is assigned a reward $r_\tau$ of $+1$ when the Curr-grid becomes the Post-grid, $-2$ if the program terminates, and $-0.01$ at every other time step.

3. **Implementing the RL algorithm:** To train the RL agent to synthesize the solution codes of Karel tasks, employ any policy gradient RL algorithm. Note that the RL agent is learning one policy that solves the entire family of simple Karel tasks described earlier. You can use the *validation* set to fine tune the hyperparameters of the implementation.

4. **Evaluating the trained agent:** Evaluate the performance of the trained agent on the *test* set. You can report the following two performance measures as an average across tasks in the *test* set: (i) the total reward of the agent in an episode and (ii) whether the agent solved the task in an episode (i.e., 1 or 0 binary measure of success). Additionally, check these performance measures separately of two task variants: (i) tasks whose Post-grid has only 1 MARKER cell and (ii) tasks whose Post-grid has 2 MARKER cells.

## 2.3 Additional Details

Next, we provide additional details in a Q/A style format. These details are based on the points raised by students working on this project in the past.

1. **Q**: *What does "Program Termination" refer to and when is a negative reward of $-2$ awarded to the agent?*

   **A**: "Program Termination" refers to abrupt termination of the Karel code due to either of the two cases: (i) Karel AVATAR "crashes" after hitting a WALL or grid boundary; (ii) Karel AVATAR attempts to pick a MARKER from a grid cell where no MARKER is available. A negative reward of $-2$ is only awarded in case of such abrupt program terminations.

2. **Q**: *What happens if the Karel* AVATAR *hits the boundaries of the task grid?*

   **A**: The Karel AVATAR "crashes" and the program terminates if it hits the boundaries of the task grid.

3. **Q**: *Can there be more than one* MARKER *in a grid-cell?*
   **A**: For the simplified Karel tasks considered in the project, each grid-cell can have at most one MARKER only.

4. **Q**: *Some generated grids are unsolvable since the* AVATAR *is trapped by* WALL *cells and is thus unable to reach the to-be-picked* MARKER*. Should we remove such grids from our dataset or is this scenario part of the task-set that the agent should learn?*

   **A**: For simplicity, we can keep such tasks in the datasets. With the current generation process, the proportion of these tasks should be small. As pointed out in the question, the agent will not be able to solve such tasks.

5. **Q**: *What if the agent is not learning anything?*

   **A**: If you don't see any performance improvement, you can try to simplify the setting, e.g., by considering the tasks where only one MARKER needs to be picked. You can continue to make such simplifications until you see some learning. Also see responses to some of the questions below.

6. **Q**: *Can we change the reward structure?*

   **A**: For training, by default, you can use the reward function we suggested in the PDF. If the agent is not learning anything, you can also experiment with different reward functions during training (e.g., by creating intermediate rewards).

7. **Q**: *Can we speed up the learning by pre-training the policy using an imitation learning method?*

   **A**: Imitation learning is a popular technique to pre-train the policy using existing data (see Section 3.3.2 "Bootstrapping from data when available" in [7]). If your agent is unable to learn or the learning progress is very slow, you can use such a technique. To pre-train your policy using imitation learning, you will need ground-truth trajectories (a sequence of actions that solve the task). For the tasks in your project, these ground-truth trajectories can be computed using path-planning (shortest path) techniques.

8. **Q**: *Is it okay to just go through the generated Karel tasks in a defined order or is it important that each episode one of these tasks is randomly sampled (which could result in never used tasks and tasks used more than once)?*

   **A**: By default, we recommend using random sampling. You can experiment with other sampling techniques, e.g., using a specific curriculum strategy.

9. **Q**: *It is said that the agent is trained with $7,000$ Karel tasks, and each episode is $100$ steps long (maximum), does this mean that we should train the agent with exactly these numbers?*

   **A**: The number of episodes can be decided independent of #unique tasks in the training set. The number of episodes could be much larger than $7,000$. By default, you could decide the number of episodes based on the learning curve (performance on training or validation set). If you do random sampling of a task per episode, some tasks will be repeated multiple times.

10. **Q**: *Which RL method should I use?*

    **A**: By default, we suggest using one of the policy gradient methods. If you wish, you can also experiment with other methods such as DQN.

11. **Q**: *How to use the validation set in the training progress?*

    **A**: You can use the validation set to fine tune the hyperparameters of the implementation. This could include learning rate, architecture of the policy network, and for early stopping (i.e., deciding the number of training episodes).

## References

[1] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. Scratch: Programming for All. *Communications of the ACM*, 52(11):60–67, 2009.

[2] Richard E. Pattis. Karel the Robot: A Gentle Introduction to the Art of Programming. John Wiley & Sons, Inc., 1981.

[3] Umair Z. Ahmed, Maria Christakis, Aleksandr Efremov, Nigel Fernandez, Ahana Ghosh, Abhik Roychoudhury, and Adish Singla. Synthesizing Tasks for Block-based Programming. In *NeurIPS*, 2020.

[4] Adish Singla, Anna N. Rafferty, Goran Radanovic, and Neil T. Heffernan. Reinforcement Learning for Education: Opportunities and Challenges. *CoRR*, abs/2107.08828, 2020.

[5] Rudy Bunel, Matthew J. Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging Grammar and Reinforcement Learning for Neural Program Synthesis. In *ICLR*, 2018.

[6] Xinyun Chen, Chang Liu, and Dawn Song. Execution-guided Neural Program Synthesis. In *ICLR*, 2018.

[7] Aleksandr Efremov, Ahana Ghosh, and Adish Singla. Zero-shot Learning of Hint Policy via Reinforcement Learning and Program Synthesis. In *EDM*, 2020.