# Virtual Environment and Automated Physical Rolling Maze as Experimental Platform for Deep Reinforcement Learning

Marc Hensel
Information and Electrical Engineering
Hamburg University of Applied Sciences
Hamburg, Germany
marc.hensel@haw-hamburg.de

Sandra Verena Lassahn
Information and Electrical Engineering
Hamburg University of Applied Sciences
Hamburg, Germany

***Abstract*—In the context of training competent future engineers, we develop platforms that shall help students to build practical competencies by working on challenging tasks for creative and highly motivating applications. Several of these platforms use systems that autonomously learn to master control tasks. Such systems are typically based on deep reinforcement learning (DRL), and related algorithms are frequently demonstrated by agents that learn to play games. In the following, we report on first results related to a platform where AI agents learn to manoeuvre balls through virtual and physical mazes while avoiding dropping into holes.**

***Keywords—artificial intelligence, deep reinforcement learning, self-learning systems, image processing***

## I. Introduction

Our main objective in the education of engineers is to develop students into valuable future employees for industry and other organizations. This means, they should build and strengthen skills and competencies required for their future profession. It is not contradictory to do so by working on creative or even playful tasks that, strictly speaking, would not result in meaningful products. A focus on educational aspects, however, is a chance to offer students challenging tasks that are highly motivating. And experience shows that high intrinsic motivation and creative working environments typically result in a very good learning effect.

In this context, we create systems—or "platforms"—that serve students as working environments for their bachelor's and master's theses. The platforms should combine technologies and methods from different fields such as deep learning, image processing, software development, electronics, and mechatronics, to name a few. This enables us to offer tasks from various technological fields, and it requires students to take interdisciplinary aspects into account. Moreover, we build up and use the platforms in consecutive theses. As an additional advantage, students typically do not have an isolated task starting from zero but need to analyze and understand the state (e.g., software code) of the provided system and build their own work upon it.

The system subject to this paper is motivated by a famous traditional maze game. It consists of a wooden box which contains a board with start location, walls, holes, and finish. Players use two knobs to tilt the board in *x* and *y* direction, respectively, and must navigate a ball along a given path from start to finish. A player loses when the ball drops into a hole and wins when the ball reaches the finish.

We highly appreciate *Allied Vision Technologies GmbH* for providing the camera and *Kowa Optimed Deutschland GmbH* for providing the lens of the physical demonstrator.

## II. System Overview

Fig. 1 gives an overview of the system. The main purpose of use is to train and demonstrate artificial intelligence (AI)—more specifically, deep reinforcement learning agents—that manoeuvre balls through labyrinths. Hence, the central component is an AI agent running on a computer such as a laptop.

### A. Software simulation (virtual environment)

Obviously, the AI agent needs a labyrinth game to interact with. An efficient way to learn how to master the game is by training in a so-called virtual environment, i.e., a software simulation of the original game. This allows to automatically play many rounds without physical restrictions such as real-time or the need to manually place the ball at its starting position.

With respect to the interaction, the AI agent gets the environment's state (e.g., the board's tilt and the location of the ball) as input. From the state it derives which action (i.e., tilting of the board) seems most promising and passes the action to apply to the environment. Not depicted in Fig. 1, the virtual environment gives the AI agent a numerical value, or "reward", as feedback on how good or bad the respective action was. For instance, the agent gets a large positive reward when the ball reaches the finish, while it gets negative feedback when the ball drops into a hole.

### B. Physical device (physical environment)

In addition to the virtual software environment, there exists a physical environment providing comparable functionality. For automated game play we have motorized the knobs of the original wooden game by standard servo motors, timing belts, and 3D printed parts. Since laptops typically do not provide GPIO ports or similar interfaces, we integrated an Arduino Uno R3 and a PCA9685 servo driver board to control the servo motors.

The system state is determined by a camera and subsequent image processing. For this purpose, we have mounted an industrial camera Alvium 1800 U-500c by Allied Vision Technologies. The camera acquires 2592×1944 pixel at up to 68 frames per second. An industrial camera gives us the flexibility to mount lenses of different focal lengths and, hence, adapt the distance between camera and labyrinth. Our current system uses a 4.5 mm fixed focus lens LM5NCL by Kowa, mounted 40 cm above the game's board.

As depicted in Fig. 1, the AI agent need not distinguish between the virtual and the physical environment. In both

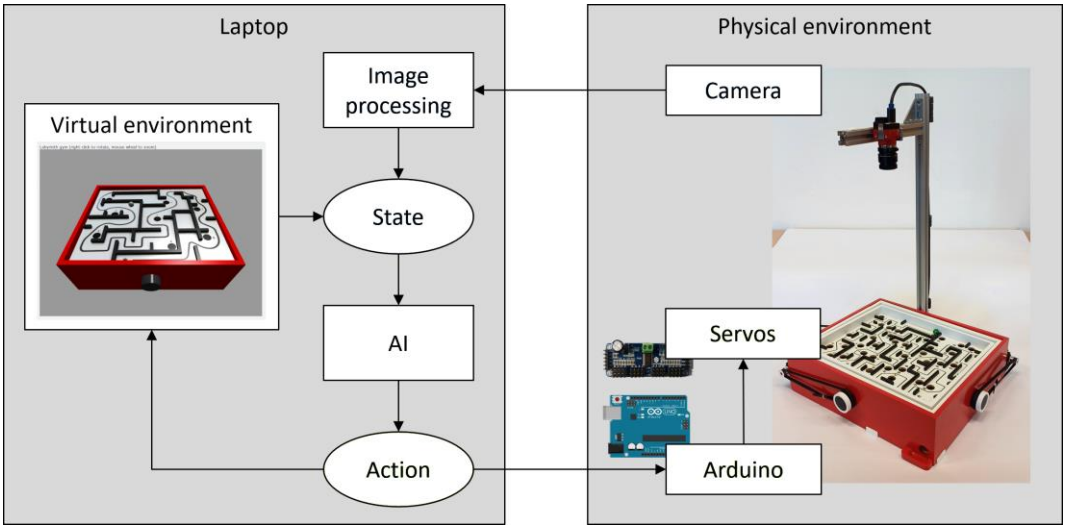


*Fig. 1: System overview*

cases it gets the same representation of the system state as input. From the agent's perspective, it does not make a difference whether these values origin from the software simulation or were derived by image processing of frames from a camera observing the physical device. Just the same, it is irrelevant for the agent whether the actions are performed by the virtual environment or the physical device.

## III. Training Procedure and Labyrinths

### A. *Solving the virtual and physical environments*

The existence of a virtual as well as a physical environment enables us to follow a step-by-step approach:

1. Train AI agents in virtual environment
2. Transfer agents to physical environment
3. Refine pre-trained agents in physical environment

At first, we use the virtual software environment, only. Once we have developed AI agents that master the virtual game, we can transfer the models to the physical device. Due to a reality gap between the physical game and its software simulation, it is likely that the models will not perform as well as they do in the virtual environment. Hence, as a third step we intend to transfer virtually pre-trained agents and continue training on the physical device to adapt them to the deviations with respect to the simulation. This shall significantly reduce the number of physical training episodes required to achieve comparable performance.

### B. *Labyrinth layouts*

As stated, the system gives us flexibility with respect to the labyrinth layouts. This enables us to adapt the complexity of the control task. However, a prerequisite for successfully transferring AI agents from the virtual to the physical environment is that both environments use the same labyrinth layouts.

Fig. 2 shows the currently used layouts, being identical in the virtual and in the physical environment—from left to right:

1. Board without holes
2. Custom labyrinth with two holes
3. Two labyrinths provided with the original game

The first board differs in the sense, that there is no path to follow and there are no holes a ball could drop into. The AI agent shall learn to bring the ball to the center location and keep it there. Apart from providing a presumably simple task, having no hole is an advantage when playing on the physical device. As the ball cannot drop into a hole the device can keep on training or playing for a long time without the need to manually place the ball on the board.

Additionally, we use three labyrinth layouts: a custom layout with only two holes and two layouts provided with the original game containing eight and 21 holes, respectively. While the custom layout defines a task of comparatively low complexity, the original layouts are obviously more challenging to solve.

## IV. Virtual Environment

We have addressed the development of a virtual environment and virtually trained AI agents in the master's thesis of Sandra Lassahn [1]. The environment is implemented in Python and written to be compatible to the OpenAI gymnasium library. Naturally, the simulation uses the same physical dimensions and labyrinth layouts as the physical

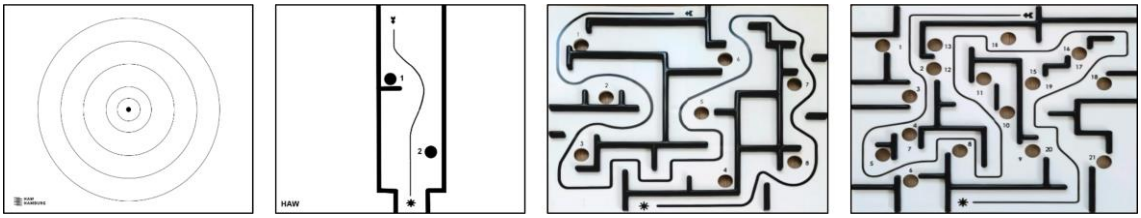

*Fig. 2: Labyrinth layouts with zero, two, eight, and 21 holes (left to right)*

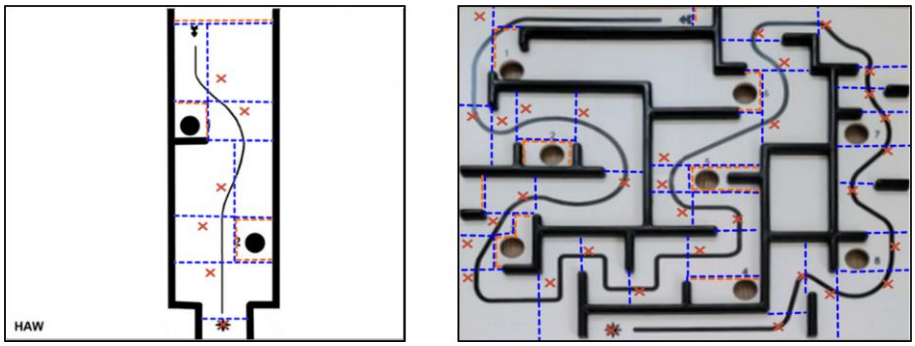


*Fig. 3: Reward system with zones (rectangles) and target coordinates (crosses)*

device. Naturally, we have modeled the physical behavior of the ball (e.g., friction and collisions with walls). Moreover, the environment offers a 3D visualization of the game. This is very helpful to demonstrate the performance of trained agents and to observe the behavior of agents in specific situations that they cannot master, yet.

## V. Deep Q-Network Agents

### A. General approach

While reinforcement learning is just a general framework to automatically master control tasks, there are numerous methods how to implement a solution. A comparatively simple and intuitive approach is Q learning, which trains a table containing the "quality" of all allowed actions given a concrete system state. Somewhat simplified, Q values are the expected future reward when taking the specific action in the given state and continue taking the best actions according to the Q table from thereon.

However, the approach is impractical when the number of actions and/or states grows too large, resulting in far too big tables to train or keep in memory. The issue can be resolved by not storing all values of the Q table but approximating them by a fully connected artificial neural network, instead. The network takes the system state as input and is trained by deep learning to output estimated Q values of all allowed actions—hence the term deep Q-network (DQN).

While alternative approaches exist, we choose DQN for two reasons. First, we want to find out if this simple approach is sufficient to solve the task. Furthermore, we have already successfully applied DQN in prior projects.

### B. Observation space and action space

As written, deep Q-networks take the system state as input and output an estimation of the quality for each action the system could take. State values are taken from the so-called observation space. We choose to feed the current tilt angles in *x* and *y* directions, the current location of the ball, and the ball's location during the previous time step into the network. Providing the current as well as the last location shall enable the agent to derive the direction and speed of the ball.

The actions, taken from the so-called action space, control the tilt of the labyrinth in *x* and *y* direction. Initially, we considered tilt angles to have values from a continuous numerical range. Moreover, actions could either represent the board's absolute inclination in degrees with respect to a level alignment, or they could represent the relative change, i.e., the tilting to apply. Finally, an action could impact both directions, *x* and *y*, simultaneously, or have an impact only on the horizontal or the vertical axis at a time, but not on both.

In our experiments we found that setting five discrete tilt angles, namely 0°, ±0.5°, and ±1°, are sufficient to navigate balls through the labyrinths. Furthermore, we do not adapt *x* and *y* at the same time. Instead, each action represents an angle in *x* or an angle in *y*, only.

### C. Reward system

During training a DQN agent receives a reward for each action taken and "learns" by iteratively adapting its parameters to maximize the overall reward. Since this it is the only feedback the agent receives, the applied reward system is one of several very critical aspects for successful training.

We conducted experiments with different reward systems, where the approach illustrated in Fig. 3 resulted in stable training processes. As a first step we divided the labyrinths into a sequence of rectangular zones leading along the path from start to finish. Hence, at any point in time it is well defined in which zone the ball is currently located and into which zone it should move to get closer to the finish. Secondly, the agent needs feedback motivating it to move the ball into the right direction. For this we have defined target coordinates located near the entry of the next zone to reach. An agent receives a positive reward when the ball gets closer to the next target location. Finally, the agent is punished by a large negative reward when the ball drops into a hole.

### D. Network architectures and training

We have trained a dedicated AI agent for each labyrinth layout depicted in Fig. 2. All agents use fully connected neural networks to approximate the actions' Q values. Naturally, all network architectures have the same input and output layers as these reflect the observed system state and the Q values. The input layers consist of six neurons receiving the current system state represented by the board's tilt angles $\alpha_x$ and $\alpha_y$ in *x* and *y*, the ball's current location $(x_t, y_t)$, and the ball's location $(x_{t-1}, y_{t-1})$ at the previous time step. The output layers contain ten neurons representing the Q values of the five discrete tilt angles in *x* and five discrete tilt angles in *y* as described above.

The number and sizes of the hidden layers depend on the labyrinth layout. For the layout without holes, we have used two hidden layers with 512 and 128 neurons, respectively. We expected the layout with two holes to be slightly more challenging—and thus require more neurons—because the ball must follow a defined path. However, two hidden layers with 128 neurons, each, were fully sufficient. Based on observations of the 3D visualization we concluded that the

task is in fact easier to learn. Instead of balancing the ball in *x* and in *y* to keep it at the board's center, the ball can keep on rolling "downward" in *y*, while the agent uses tilting in *x* to let the ball roll along the right wall and the left wall.

For the layout with eight holes, we have used three hidden layers with 2048, 1024, and 256 neurons. Naturally, it is far more complex to navigate a ball through the layout with 21 holes. As we did not want to significantly increase the network any further, we trained two agents with one agent navigating the ball through the first half, and the other one taking over for the second half of the path from start to finish. Both agents have three hidden layers with 2048, 1024, and 256 neurons.

As final remarks, all networks use leaky ReLU activations, batch normalization, and Adam as optimizer. The number of episodes range from about 1,000 for the simple layouts to about 3,500 for the layouts with eight and 21 holes. Moreover, we applied an epsilon greedy approach to balance exploitation and exploration during training.

## VI. Results

So far, we have finalized a first thesis [1] on our experimental platform for deep reinforcement learning using virtual and physical mazes. We have developed a fully functional virtual environment with optional 3D visualization, being a simulation of the original physical game, and successfully trained DQN agents to reliably navigate balls through different labyrinth layouts. Moreover, there exists an application with graphical user interface which simplifies the parametrization and training and can be used to demonstrate trained agents playing the game.

With respect to the physical environment, we have motorized the original game and implemented software to control the device. The low-level control of the servo motors is handled by an Arduino board. Applications running on a connected computer can operate the physical device by using our Python classes, which manage the connection and communication with the Arduino board and provide methods to set the servo motors. Finally, we conducted a proof of concept with a simple webcam and basic image processing.

## VII. Ongoing Work and Outlook

At the time of writing there is an ongoing thesis [2] which concentrates mainly on making the physical environment fully operational. We have integrated the industrial camera and now have full control of, for instance, the acquisition trigger and the exposure time. Most noticeably, we develop image processing methods to extract the labyrinth board and detect and locate the ball, holes, start, finish, and walls. These are key to providing AI agents with the current state of the physical environment (see Fig. 1). Finally, we intend to transfer virtually trained agents to the physical device, although this will only be a first attempt and needs to be addressed in detail in future work.

Further future work, once the platform is fully functional, might include experiments with reward systems, alternative approaches for deep reinforcement learning, custom layouts, and the development of an agent which is able to play labyrinths it has not seen during training, to name a few.

## References


[1] S. V. Lassahn, "3D-Simulation und prototypischer Aufbau eines durch Reinforcement Learning gesteuerten Labyrinths," Master's thesis, HAW Hamburg, 2024.

[2] S. Rastagar, "Deep Reinforcement Learning und Bildverarbeitung zur generalisierbaren Steuerung physischer Labyrinthe," Master's thesis, HAW Hamburg, to be published 2025.