Action Spaces in Deep Reinforcement Learning to Mimic Human Input Devices

Marco Pleines*, Frank Zimmer†, Vincent-Pierre Berges‡
*Technische Universität Dortmund, Germany,
marco.pleines@tu-dortmund.de
†Laboratory for Computational Intelligence and Visualization
Rhine-Waal University of Applied Sciences, Kamp-Lintfort, Germany
‡Unity Technologies, San Francisco, California, USA

***Abstract*—Enabling agents to generally play video games requires to implement a common action space that mimics human input devices like a gamepad. Such action spaces have to support concurrent discrete and continuous actions. To solve this problem, this work investigates three approaches to examine the application of concurrent discrete and continuous actions in Deep Reinforcement Learning (DRL). One approach implements a threshold to discretize a continuous action, while another one divides a continuous action into multiple discrete actions. The third approach creates a multiagent to combine both action kinds. These approaches are benchmarked by two novel environments. In the first environment (*Shooting Birds*) the goal of the agent is to accurately shoot birds by controlling a cross-hair. The second environment is a simplification of the game *Beastly Rivals Onslaught*, where the agent is in charge of its controlled character's survival. Throughout multiple experiments, the bucket approach is recommended, because it is trained faster than the multiagent and is more stable than the threshold approach. Due to the contributions of this paper, consecutive work can start training agents using visual observations.**

I. Introduction

Human players generally interact with video games by utilizing input devices like a computer mouse, a keyboard or a gamepad, while observing the screen. Based on the screen's output, decisions, on which buttons to press or locomotion tasks (i.e. moving the mouse or a thumbstick) to carry out, are sequentially to be made at high frequencies. In contrast to humans, agents play many environments using actions, that are not usually mapped to the aforementioned devices, but are easier to train for achieving more reasonable behaviors to solve their environment. For example in DotA 2 (Defense of the Ancients 2), the non-profit company OpenAI developed actions, which select positions in the game world by using a limited grid [1]. This way, the players' infinite choices of positions (by moving the mouse) are reduced and therefore simplified. Another action, which can also be seen in real-time strategy (RTS) games like StarCraft, is to choose game entities as targets. This is normally done upon a mouse click by a human player. Instead, an agent gets a direct choice of targets, which omits the necessity of moving the mouse cursor above the desired entity before clicking. Overall, these kinds of simplifications fit to the domain of their respective environment but most likely do not fit to other ones. To overcome this issue and henceforth to approach agents playing video games more generally, agents shall utilize common action spaces. In the context of video games, these action spaces shall mimic human input devices.

Two peculiarities have to be considered for developing these spaces. Actions shall be carried out concurrently to allow rich behaviors. For example, for some first-person-shooter (FPS) video games, an eligible strategy is to run, strafe, and shoot concurrently [2]. Moreover, if a robot arm had to rotate its joints one after the other, it would take much more time to accomplish its job (e.g. picking up and moving items). Such essential and rich behaviors are not possible when each action is carried out sequentially. This context can be expanded to concurrent action spaces comprising discrete and continuous actions. Mimicking human input devices cannot be done by simply using a discrete or continuous action space alone. In FPS video games, players move the mouse or the thumbstick of a gamepad to change their view. This kind of action is continuous as it allows infinite directions and velocities as input. While moving these devices, players also need to press buttons concurrently to trigger an event such as shooting a gun, which is a discrete action.

II. Contributions

To make use of concurrent discrete and continuous action spaces, this paper examines three approaches, which are applied in the context of DRL (other contexts are applicable as well):

A) **Threshold Approach**: discretizing continuous actions using a threshold
B) **Bucket Approach**: breaking down continuous actions into multiple discrete actions
C) **Multiagent Approach**: combining two cooperative agents to comprise both action spaces

In addition, a general elaboration on four possibilities on how to realize the agents' locomotion is illustrated:

1) Selecting velocities for the horizontal and vertical axis
2) Selecting a torque to rotate the agent and a velocity to move it forward
3) Determining the agent's direction by a position on a grid, while selecting a velocity to move forward

978-1-7281-1884-0/19/$31.00

4) The agent selects an angle which derives a direction from a unit circle, while selecting a velocity to move forward

The mentioned approaches are benchmarked using the herein contributed new environments *Shootings Birds* (SB) and *Beastly Rivals Onslaught* (BRO), which are based on Unity's ML-Agents toolkit [3]. Because the focus is set on action spaces, the training time of the environments shall not exceed dozens of hours (see section 5). Hence, it is not intended to use visual observations nor to face an environment's complexity as seen in DotA. After solving the issue of concurrent discrete and continuous actions, future work can start on developing agents that play multiple games generally using a common observation space (i.e. visual observation of the screen).

This paper proceeds as follows: in section 2, related work is portrayed focusing on research concerning action spaces as well as the action space composition of DotA. Section 3 details the aforementioned approaches and locomotion options. After that, the contributed environments are showcased comprising their dynamics, observation spaces, action spaces and reward signals. Section 5 introduces the run experiments and provides the results, which are put into context by the discussion in section 6. The final section draws conclusions and provides an outlook to future work.

III. Related Work

Policy gradient and actor-critic methods, like DDPG [4], A3C [5] or PPO [6], already support concurrent continuous actions. This can be done by taking a sample from the Gaussian distribution for each output of the policy's neural network. For discrete action spaces, Tavakoli et al. (2017) and Harmer et al. (2018) contributed solutions for developing concurrent discrete action spaces. Latter create an A3C policy network whose outputs are treated as a set of Bernoulli random variables [2]. Actions are selected from this policy by sampling each output individually. A different solution to support concurrent discrete actions is the idea of action branching [7]. Instead of using one output layer for all actions, the neural network is branched off into multiple sub-layers (i.e. branches). One action is sampled from each branch. Then, each action sample is concatenated to form a joint-action tuple. This functionality has been applied to the DQN algorithm [7]. Action branching has been adapted by Unity's ML-Agents toolkit to extend their PPO implementation [3]. Other related work researched parameterized action spaces, where discrete actions are accompanied with continuous parameters to allow more complex actions [8], [9]. Parameters could specify the force of a kick.

Once the desired action space kinds can be used concurrently, the next concern is to define actions fitting to human input devices. As mentioned in the introduction, OpenAI did not implement a solution to let an agent mimic a mouse cursor and a keyboard simultaneously. Instead, they created 170,000 discrete actions (per hero) to cover all actions [1]. Some of these actions, such as casting an ability, require an explicit game entity to be targeted (e.g. enemy hero or building). A human player would have to select the desired ability or attack

Fig. 1: OpenAI Five: Discretized attack actions [1]

Fig. 2: OpenAI Five: Discretized spell cast (left) and movement locations (right) [1]

move and then click on the intended entity. In OpenAI Five [1], these actions are hard coded. Given the scene in Figure 1, the agent (Sniper) has 6 potential foes to attack. So there are ultimately 6 attack actions, where each one represents a different target. Whenever a location on the ground is to be selected (Figure 2), the continuous action space is discretized to a limited grid. For selecting a position for the agent to move to, the grid is limited to the agent's (Viper) position. If a location is to be selected for an ability, the grid is originated at game entities.

IV. Approaches and Locomotion Options

The previously portrayed related works provide solutions to concurrent action spaces, but not to action spaces mixing discrete and continuous actions. In this section, the details of the approaches on enabling concurrent discrete and continuous actions are elaborated in the context of DRL. Besides that, approaches concerning the locomotion of agents are examined, where eventually one out of four locomotion implementations is selected. This work uses the PPO implementation of the ML-Agents toolkit [3]. The underlying architecture is similar to the one from Schulman et al. (2017).

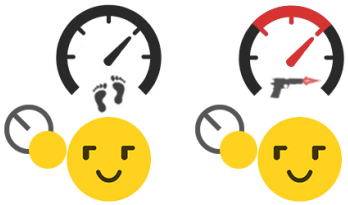


Fig. 3: The threshold agent uses a range, specified by the threshold, to decide on whether to shoot or not to shoot (right), while simultaneously selecting a movement speed (left).

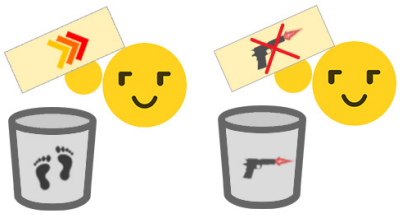


Fig. 4: The bucket agent draws a movement speed value from a limited amount of options (left), while simultaneously deciding on the discrete actions to shoot or not to shoot (right).

A. Threshold Approach

This approach considers all actions to be continuous. Continuous actions, such as movement, are represented by a value in the range $[-1,\ 1]$ in the ML-Agents toolkit. These actions are carried out without further processing. In this approach, actions, which are supposed to be discrete, are derived from a continuous one by applying a threshold. The threshold can be applied to the action's absolute value:

$$|action\ value| \begin{cases} < threshold & \rightarrow \quad \text{Shoot} \\ \geq threshold & \rightarrow \quad \text{Do nothing} \end{cases}$$

For example, if a threshold of 0.1 is applied, every value in the range $(-0.1,\ 0.1)$ will trigger the actual discrete action. The threshold can also be applied to the raw action without taking its absolute value first. This means that the discrete action is triggered if an action value is selected in the range $[-1, 0)$. Experiments will show whether it is beneficial to take the absolute value or not. Figure 3 illustrates a threshold agent.

B. Bucket Approach

The bucket approach can be interpreted as the opposite of the threshold approach. This time, actions which are meant to be continuous are now discretized by a set of discrete actions within a certain range, referred to as "bucket" from now on. This bucket also simplifies a continuous action, if its resolution (number of fine-grained or atomic actions) is not too high. Due to this simplification, precision can be lost if not enough atomic actions are available to choose from. But also increasing the number of actions amplifies the effect of the curse of dimensionality [4]. This increases the complexity of the to be trained problem, while the agent's action space is much harder to explore. Note that in this approach's action setup, regular discrete actions remain the same. Figure 4 shows an example of a bucket agent.

Buckets can be based on Tavakoli et al.'s (2017) action branching architecture, which is implemented in Unity's ML-Agents toolkit. One action branch shall represent one continuous action. Further branches can implement classical discrete actions. Also, the problem of incompatible joint-action tuples are avoided this way, because only one action per branch is selected to create the tuple. For example, shooting and reloading can be part of the same branch and therefore do not conflict.

A continuous action is bucketed by a certain number of sub-actions in a certain range. In the case of this paper, the individual actions are selected from the range $[-1,\ 1]$. The policy tells the agent which sub-action to take from the bucket.

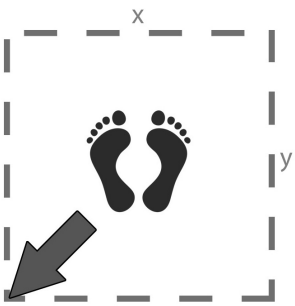
Fig. 5: Both agents make use of their natural action spaces where the continuous agent (yellow) chooses a movement speed (left), while simultaneously the discrete one (orange) decides whether to shoot or not to shoot (right).

C. Multiagent Approach

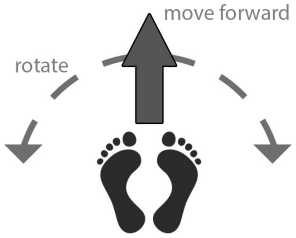
Assembling a multiagent, using agents that only use one of the two available action space types, is the last approach. Each action space operates as regularly intended without applying any buckets or thresholds. An example is given by Figure 5, where one agent decides on the continuous locomotion and the other one on shooting or not. However, using multiple agents has a few drawbacks. One deals with a redundant observation space, because each agent trains its own model, and in the worst case, both agents process the same amount of information. Depending on the environment, some information can be omitted. Same accounts for the reward function. The aforementioned continuous agent cannot be punished for reloading a fully loaded gun. Thus, the agents may vary in all RL components, which make them hard to compare to the previously mentioned approaches. As all agents have the same goals, they all have to receive a shared reward signal to aid cooperation.

D. Locomotion Options

Four possible locomotion implementations can be considered for developing the agents. The approach in Figure 6a is based on two continuous actions. One determines the velocity

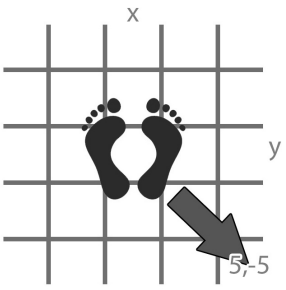


(a) The agent's locomotion is driven by the velocities on the horizontal and vertical axis.



(b) The agent rotates left and right while moving forward at a certain speed.

Fig. 6



(a) The agent moves at a certain speed towards a chosen grid cell.

(b) The agent selects an angle as moving direction and then moves at a certain speed towards this direction.

Fig. 7

on the x-axis and the other one on the y-axis. Therefore, the fastest velocity is accomplished by choosing a magnitude of 1.0 for each axis. As a consequence, the agent is biased to move diagonally most of the time. Due to this behavior, the agent is less precise or takes longer to achieve its goal, while appearing less authentic to observers. Normalizing the velocity could resolve this bias. Because normalizing a vector returns a vector of length one, a third action would have to be added to manipulate the agent's speed.

Another variant (Figure 6b) uses one continuous action to rotate the agent and a second one to select a speed to push the agent forward. Apparently, this is not a solution to the underlying use case, because a computer mouse is not intended to be rotated during its usage.

The third option considers the discretization approach, which was taken by OpenAI for DotA as seen in section 3 and Figure 7a. Following their strategy, the agent selects a grid cell to determine a position to move to. The grid cells are represented by at least one discrete action branch. An additional continuous action is in charge of selecting a movement speed.

The last approach makes use of one continuous action to derive a direction from a position on a circle's circumference.

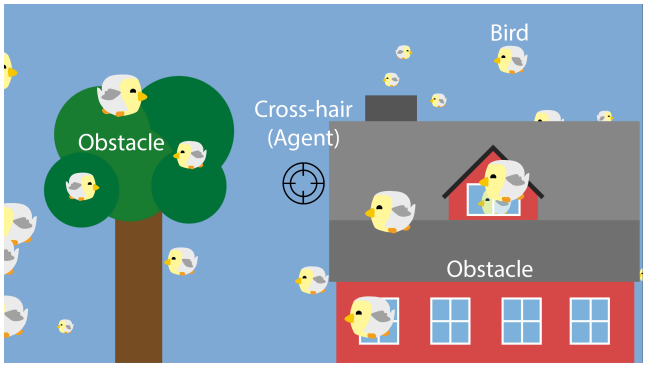


Fig. 8: The environment Shooting Birds and its entities

This way, the action determines an angle. A second one chooses the movement speed of the agent. An exemplary illustration is given by Figure 7b. This approach is used for the to be solved environments, because it closely represents the human usage of a mouse. Also, it does not introduce a bias like the first locomotion approach.

V. Environments

Some apparent restrictions have greater impact on how the environments are going to be designed. The scope of this work focuses on action spaces including various experiments for the SB and BRO environments. Training times, which take more than a dozen hours, are infeasible. This is due to many training runs, which have to be conducted to tune hyperparameters and test the implemented environments' dynamics. Therefore, too complex environments might demand too much training time, leading to a higher risk of defective results. For example, Deepmind's AlphaStar was trained for two weeks [10]. So complexity is a high concern and in general, it is advisable to start out as simple as possible when building RL environments. Once training succeeds, complexity can be raised. To keep the to-be-trained problem smaller, images are not used as observation.

Next to these restrictions, the environments shall ask the agent to act precisely to analyze the effectiveness of the introduced approaches. This has to be proven feasible for at least two use cases, in particular the SB and BRO environment, which are implemented using Unity's ML-Agents toolkit [3].

A. Shooting Birds

The Shooting Birds environment (Figure 8) is inspired by the German game *Moorhuhn*[1]. In this environment, the agent steers a cross-hair to shoot birds.

1) Dynamics of the Environment: Birds of three different sizes are randomly spawned on the left and right side of the environment. For every vanished bird, a new one is spawned. Their velocities and flying behaviors are stochastically initialized. The agent's shooting functioning is limited by ammunition (8 shots). Therefore, the agent has to manually reload to be able to shoot again. Birds, which are hidden

[1]https://www.moorhuhn.de/

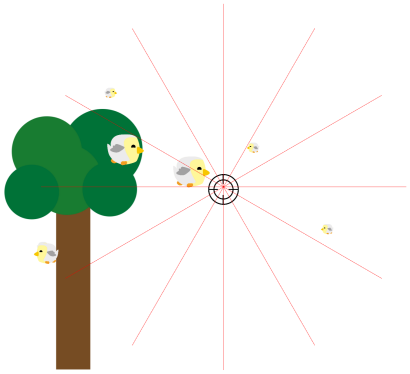


Fig. 9: Raycasts to measure distances to perceived birds

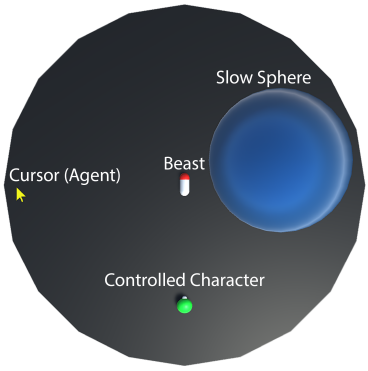


Fig. 10: The simplified BRO environment and its entities

behind obstacles (e.g. tree and barn), cannot be hit. At the beginning of each episode, one out of 7 levels is randomly loaded. Further, the agent's ammunition is randomized.

2) Observation Space: To partially observe the environment, the agent collects 44 observations. The whole observation space is stacked three times, which includes the current and the past two sets of observations. 36 out of these 44 values are perceived by using raycasts (Figure 9). Starting from the center of the cross-hair, the raycasts are sent outwards at angular steps of 30 degrees. This way, the agent detects birds and perceives their sizes and distances to them. Distances to obstacles are sensed as well. If no birds or obstacles are detected at all, the observed values are described with -1.0 for that particular raycast. The remaining 8 values comprise:

- Remaining ammunition
- Gun loaded (true or false)
- Agent's position (x and y value)
- Agent's velocity (x and y value)
- Agent's speed (magnitude of the agent's velocity)
- Hovered entity (bird, obstacle or unknown)

3) Action Space: Two continuous and two discrete actions are available to the agent. The agent's velocity is dependent on two continuous actions based on the locomotion approach, which includes a unit circle (Section IV-D). Shooting and reloading are triggered by the discrete ones. In the case of the bucket and multiagent approaches, a third discrete action is added, which does nothing at all. This is necessary for the agent to decide when not to shoot or not to reload. The agent makes a decision for every step of the environment.

B. Reward Signals

Accurately shooting down birds as fast as possible is the agent's goal. Thus, 6 different rewards are signaled throughout the environment's execution. The agent is rewarded for hitting a bird. However, the size of the bird matters. Big birds handout $+0.25$, medium ones $+0.75$ and small ones $+2.0$. Hitting nothing is punished with a negative reward of -0.1. The other reward signals are related to the ammunition. If the agent tries to shoot without ammunition, it is punished by -1.0. Reloading, while the agent's gun is still loaded, is punished proportionally to the amount of wasted shots: $remaining\ shots/max\ shots$. Therefore, reloading an empty gun is not punished, while wasting 4 shots is punished by -0.5.

C. Beastly Rivals Onslaught

This implemented environment is a simplification of the game BRO[2]. To avoid the challenge of more complex multiagent systems, the original game is broken down to just one player with the goal of surviving as long as possible.

1) Dynamics of the Environment: Figure 10 shows the four main game entities:

- A mouse cursor, which represents the agent
- A character, which is controlled by the cursor
- A beast, which chases and kills the character once reached
- A slow sphere, which wanders around and slows down the beast and the character

Over time, the beast gets more accurate and moves more quickly, and thus survival is more and more challenging. Another threat or potential support is the slow sphere. The character's and the beast's velocities are reduced while inside this sphere. The closer they are to its center, the more the agent's and beast's motion will be slowed down. To survive, the agent has to send suitable commands to its controlled character. By "clicking on the ground" the agent tells the character to move to that location. Another command, which could be implemented as a "double-click" for a human player, is the blink ability. Once commanded, the character is teleported instantly to the location of the cursor. The blink ability is then on cooldown (i.e. unavailable) for 200 steps of the environment. For visualization purposes, the character is colored orange while not being able to blink.

BRO has some stochastic elements, which affect the environment's initial state. Random locations are chosen for the agent, the controlled character and the slow sphere. Additionally, the beast's rotation is randomized.

2) Observation Space: 21 observations are collected by the agent to fully observe its environment:

- Blink cooldown

[2]Video of the game BRO https://youtu.be/OTcDd7a_R0A

- Mouse cursor position
- Mouse cursor velocity
- Character position
- Character velocity
- Relative position to the beast
- Beast position
- Beast velocity
- Beast speed (magnitude of the beast's velocity)
- Beast accuracy growth speed
- Relative position to slow sphere
- Slow sphere velocity

Position and velocity information are two dimensional features. Relative positions are related to the character. The whole observation space is stacked three times.

3) Action Space: Two continuous and two discrete actions are apparent. The agent's velocity is dependent on two continuous actions based on the locomotion approach, which includes a unit circle (Section IV-D). One discrete action triggers the movement of the character to the "clicked" location commanded by the agent. The other one is the character's blink ability, which is a teleport to the agent's position. A third discrete action is considered for the bucket and multiagent approaches, because the agent has to be capable of deciding when not to move or not to blink at all. Every six step, the agent decides which action to take. In-between these steps the same action is triggered.

D. Reward Signals

Three kinds of reward signals are sent to the agent. As long as the character is alive, the agent gets rewarded with $+0.01$. Being touched by the beast is punished with -1.0 and the last reward signal punishes the agent with -0.025 for trying to use the blink ability during its cooldown time.

VI. Experiments and Results

Seven experiments are conducted to benchmark the selected approaches on each environment. One experiment is concerned with the multiagent approach, whereas three different thresholds and three different bucket sizes are used for the other experiments:

- Thresholds: 0.0, 0.1, 0.5
- Bucket Size: 3, 11, 41

The thresholds 0.1 and 0.5 are applied on the absolute value of the action's value. 0.1 is chosen to provide a smaller range that triggers the discrete action, because the agent could receive too many punishments during exploration (e.g. frequently missing birds) and therefore stops using that action completely. Although a smaller range could be explored too infrequently. Half of the action value's range is enabled by the thresholds 0.0 and 0.5. Using 0.0 makes all negative values trigger the discrete action, whereas 0.5 utilizes the middle region of the action value. Concerning the bucket sizes, these are chosen to answer the question whether a small bucket lacks in precision or a big bucket is too complex to train. Only one multiagent experiment is run as reasoned in the previous section.

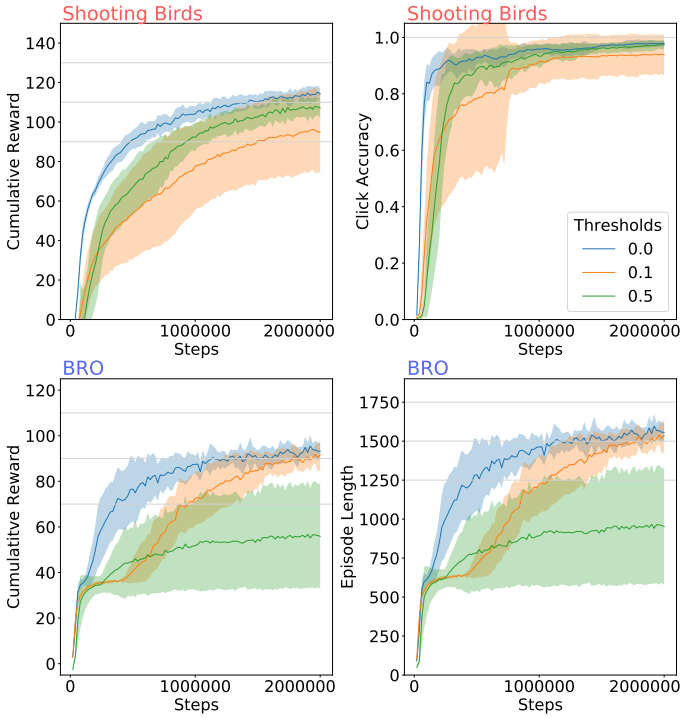


Fig. 11: Threshold Results

During the next subsections, the results for the run experiments are portrayed and described. For all measured values, the mean and standard deviation over ten training runs is shown. Each experiment measures the cumulative mean reward of episodes. The mean click accuracy is recorded for SB and the episode length is tracked for BRO. More details on the conducted runs, such as the hyperparameters, can be found in this repository[3].

A. Threshold Results

For both environments, a threshold of 0.0 outperforms the other ones in terms of stability and score as seen in Figure 11. In SB, it reaches a cumulative reward of about 114, while achieving an accuracy of about 97%. The threshold 0.5 behaves similarly, but not as good. A very unstable and much weaker performance is achieved by threshold 0.1. Concerning BRO, all thresholds have a similar behavior to the ones from SB. The best threshold agent converges to a cumulative reward of about 93, while it survives for about 1560 steps.

B. Bucket Results

The three different bucket resolutions do not provide a distinctive winner in the BRO environment (Figure 12). All of them behave stable, while reaching a cumulative reward of about 104 and an episode length of more than 1700 steps. A much greater differentiation among the performances of the buckets can be observed for SB. The best result is achieved by the bucket using a size of 11. It accomplishes a cumulative

[3]GitHub Repository https://github.com/MarcoMeter/Action-Space-Compositions-in-Deep-Reinforcement-Learning

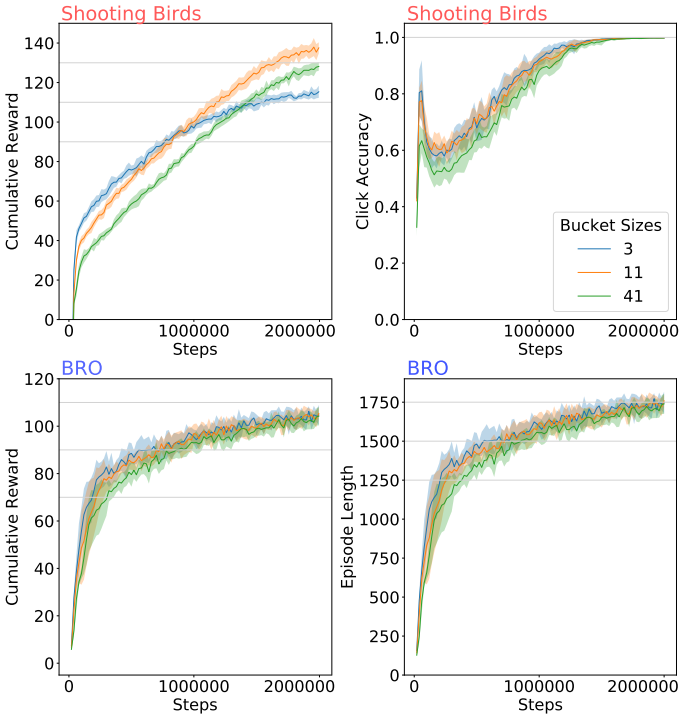


Fig. 12: Bucket Results

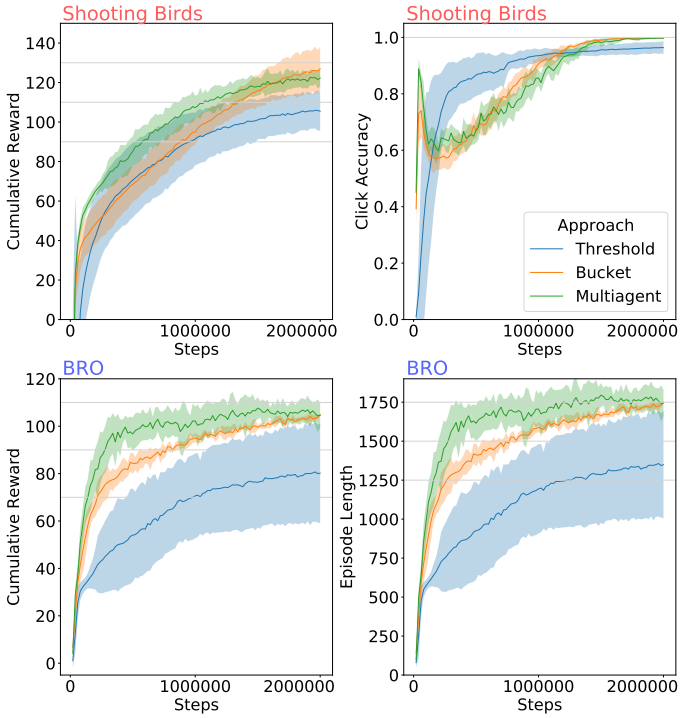


Fig. 13: Mean Results of all Approaches

reward of about 136 and an accuracy around 99.8%. The other buckets are as accurate, but do not get passed a cumulative reward of 130.

C. Approach Comparison

Figure 13 shows the mean of the threshold and bucket experiments as well as the results of the multiagent approach. In BRO, the multiagent achieves a cumulative reward of 105 and an episode length of about 1750 steps. This is slightly better than the bucket approach, but comes at a cost of 20% more training time (Figure 14). A cumulative reward of about 122 and an accuracy of about 99.7% is scored by the multiagent in SB. Comparing all approaches, the threshold approach is by far the most unstable one and is quite apart concerning its performance. The bucket approach is superior to the multiagent one in SB, but slightly inferior to the multiagent in BRO. Figure 14 shows the training durations of each experiment for both environments. In SB, the mutliagent requires more than 30% of time to be trained. The other experiments take about the same amount of time except for the biggest bucket size, which takes a little more time.

D. Learned Behaviors

The best runs of each approach can be watched in this video[4]. For each environment, the best resulted behaviors are very similar. In SB, it can be observed that the agents accurately and rapidly shoot down their targets, while not wasting any shots. Also, they are capable of hitting birds

[4]Video of the learned behaviors https://youtu.be/Pb14i3srRWc

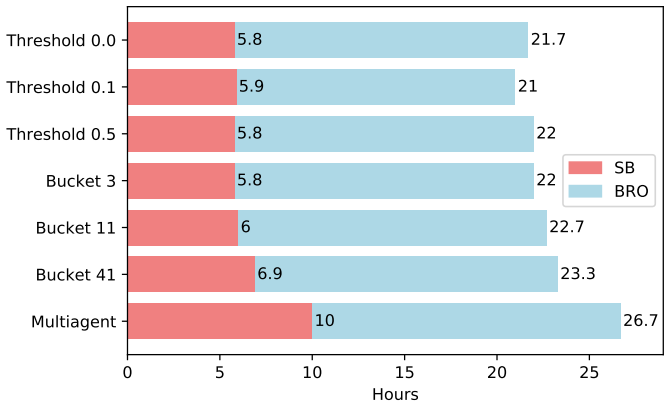


Fig. 14: Training duration of each experiment

which are partially obscured. Only the threshold agent looks less efficient by occasionally skipping denser regions of birds. Once a side of the environment is cleared, the agent decides to transition to the other one. In relation to that, the velocity is usually greater on the x-axis on average.

Concerning BRO, most trained behaviors usually survive the whole episode (2000 steps). The agent commands very frequently movement locations to its controlled character, while its blink ability is nearly never used. During the course of an episode, the agent moves in big laps. The longer the episode lasts, the smaller the laps are walked by the agent, which are usually located at the pitch's boundaries. Then, the agent moves the cursor in a triangular shape. The slow sphere is not exploited by the agent nor harms it.

VII. Discussion

The best performance is delivered by the bucket and the multiagent approach. High instability is observed for the threshold one. Due to the much longer training times of the multiagents, the bucket approach is recommended. This outcome is due to the less complex nature of discrete actions, which are easier to explore. However, a trade-off has to be made in terms of precision and complexity concerning the number of atomic actions of a bucket (curse of dimensionality). The results of SB show that 11 actions outperform its competing experiments using 3 and 41 actions. 41 actions are too complex to achieve the same performance using the same training time, while only 3 actions lack in precision (i.e. approaching targets). Concerning the threshold approach, it can be observed that the thresholds 0.1 and 0.5, which are applied to the absolute action value, achieve worse results than the 0.0 threshold. The multiagent approach comes with the drawbacks of training a redundant model as well as putting further effort into the design of the observation space and the reward signals.

Next to the benchmarked approaches, the benchmarks themselves can be discussed. SB made the differences among the approaches apparent, while BRO is too little of a challenge due to the very similar performances by the bucket and multiagent approaches. For the agents playing that environment, it is simply enough to keep constantly clicking and moving in a triangular shape. The environment needs more dynamics to demand a more precise agent behavior - for instance the complete game of BRO. Nearly no impact is contributed by the agent's blink ability and the slow sphere. Having to click on foes, which unpredictably move or blink, is much harder in comparison to the consistent behavior of the birds in SB.

VIII. Conclusion

Throughout this paper, three approaches for enabling concurrent discrete and continuous actions in DRL were examined. The threshold approach discretizes a continuous action to achieve the behavior of a discrete one. Multiple discrete actions replace a continuous one based on the bucket approach. Lastly, the multiagent combines several agents to support discrete and continuous actions concurrently. These approaches were challenged by two novel environments, where the agent is in charge of controlling a mouse cursor to play a video game. As for the outcome of the conducted experiments, the bucket approach is the most recommendable, as it achieved stable training results while not taking as long to train as the multiagent approach. Also, the multiagent draws more effort into account for designing appropriate observation spaces and reward signals. Due to high instability and low performance, the threshold approach is inferior to the other ones. In conclusion, the underlying contributions establish the first steps of solving the problem of mimicking human input devices.

A. Future Work

To let agents play video games generally, supplementary work is necessary. Based on the provided contributions, the next step is too change the observation space to visual observations. The action space would be based on the bucket approach while using the unit circle locomotion implementation. Once this setup achieves reasonable results, approaches can be developed to let an agent learn to play more than one game. As the agents already succeed in both environments, more complexity can be introduced. For example BRO could be implemented as complete game, which draws topics of competitive multiagents into account like opponent sampling [11]. Also, clicking on unpredictably behaving foes is more challenging. SB could implement more dynamics of its role-model, like moving the camera to travel the environment. Further, procedural content generation could be employed to replace the hard coded levels. This is beneficial to avoid overfitting and to measure the generalization capabilities of the trained agent [12].

References

[1] OpenAI, "Openai five," 2018, available at https://blog.openai.com/openai-five/ retrieved March 21, 2019.

[2] J. Harmer, L. Gissln, H. Holst, J. Bergdahl, T. Olsson, K. Sj, and M. Nordin, "Imitation learning with concurrent actions in 3d games," *CoRR*, vol. abs/1803.05402, 2018. [Online]. Available: https://arxiv.org/abs/1803.05402

[3] A. Juliani, V.-P. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," 2018, **arXiv preprint arXiv:1809.02627.** https://github.com/Unity-Technologies/ml-agents.

[4] 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," *CoRR*, vol. abs/1509.02971, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[5] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[7] A. Tavakoli, F. Pardo, and P. Kormushev, "Action branching architectures for deep reinforcement learning," *CoRR*, vol. abs/1711.08946, 2017. [Online]. Available: http://arxiv.org/abs/1711.08946

[8] M. J. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," *CoRR*, vol. abs/1511.04143, 2015. [Online]. Available: http://arxiv.org/abs/1511.04143

[9] J. Xiong, Q. Wang, Z. Yang, P. Sun, L. Han, Y. Zheng, H. Fu, T. Zhang, J. Liu, and H. Liu, "Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space," *CoRR*, vol. abs/1810.06394, 2018. [Online]. Available: http://arxiv.org/abs/1810.06394

[10] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, T. Ewalds, D. Horgan, M. Kroiss, I. Danihelka, J. Agapiou, J. Oh, V. Dalibard, D. Choi, L. Sifre, Y. Sulsky, S. Vezhnevets, J. Molloy, T. Cai, D. Budden, T. Paine, C. Gulcehre, Z. Wang, T. Pfaff, T. Pohlen, Y. Wu, D. Yogatama, J. Cohen, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, C. Apps, K. Kavukcuoglu, D. Hassabis, and D. Silver, "Alphastar: Mastering the real-time strategy game starcraft ii," 2019, available at https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/ retrieved March 7, 2019.

[11] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent complexity via multi-agent competition," *CoRR*, vol. abs/1710.03748, 2017. [Online]. Available: http://arxiv.org/abs/1710.03748

[12] A. Juliani, A. Khalifa, V. Berges, J. Harper, H. Henry, A. Crespi, J. Togelius, and D. Lange, "Obstacle tower: A generalization challenge in vision, control, and planning," *CoRR*, vol. abs/1902.01378, 2019. [Online]. Available: http://arxiv.org/abs/1902.01378