Cardiff University

School of Computer Science and Informatics

CM3203 – Individual Project Initial Plan

Doom-playing AI via Deep Reinforcement Learning

Supervisor: Dr Frank C Langbein

Author: Hugo Huang

Initial Plan: Doom-playing AI via Deep Reinforcement Learning

1. Project Description

Training artificial intelligence (AI) agents to play games directly from high-dimensional sensory inputs (like visuals or audio) was widely considered as one of the greatest challenges of reinforcement learning (RL) [6, 7] until Deep Q-Network (DQN) [6] was proposed. Even more performant variants of DQN, training agents to play first-person shooter (FPS) games like Doom is a non-trivial task [3] and in this project I will attempt to create an AI agent that plays Doom via a Deep Q-Learning-based approach. The goal is to utilize VizDoom [2] and train my AI agents to play most of its provided scenarios (customized maps for RL) at human-level, if possible, the agent would also be aimed to play the first map of Doom: E1M1 at a close-to-human-level. Instead of relying directly on high-dimensional sensory inputs, my model would use both the pixels on the screen and depth buffer information (essentially RGB-D images) as inputs, with the latter provided as input to compensate for the lack of depth-completion ability. If possible, the agent can also incorporate a variant of Spatial Propagation Network (SPN) [5] for predicting the depth information using on-screen pixels as input.

The project would be carried out with Python as the main programming language. Python, despite its relatively poor performance in many cases, is very suitable for fast prototyping. With highly optimized libraries written in high-performance languages like C++, Fortran and CUDA, it is possible for Python programs to match the performance level of these languages and utilize specialized hardware such as GPU to accelerate batch processing jobs. After careful consideration, I decided that these libraries would be suitable for this project:

- PyTorch would be used to construct, compile, and execute the RL model I designed. Both the training and testing phase of the AI agent would be carried out using it.
- VizDoom would be used to provide the environment for the RL agent to train and test in. More specifically it would be providing the pixel and depth information required by my RL agent and also be executing the decisions of the agent.
- Matplotlib or PyEcharts would be used to plot relevant graphs visualizing how error (loss) would change over time and how that would differ between epochs or different sets of hyper parameters. Matplotlib would be used during development for its simplicity and PyEcharts would be used for visualization in the final report.
- NumPy's array data structure would be used to record data during training and provide me with valuable data, this library is also required by all of the abovementioned libraries.

I am familiar with NumPy, Matplotlib, PyEcharts and have some experience with PyTorch, however, VizDoom is new to me and may require one or two weeks of learning to use effectively. This project would use the 3.10.9 version of Python as it is the latest version at the time of initial planning that supports PyTorch on Windows machines.

PyTorch is based on Torch, a Lua-based open-source machine learning (ML) library and is widely used in the field of ML. It provides quality-of-life features such as automatic differentiation and allows users to customize ML models freely. I chose it because the actual computations are performed in high-performance languages like C++ and CUDA utilizing GPU acceleration when available, so it's mostly free of the performance bottlenecks of Python, only a small communication overhead exist and it's negligible.

VizDoom is a tool based on ZDoom, one of the most popular source-ports of the 1993 genre-defining FPS game, Doom. ZDoom supports all games running on the id tech 1 game engine but VizDoom is modified to support Doom for RL research. VizDoom has been cited in over 500 articles and has a good Python interface, it can provide content of frame buffer (on-screen pixels) and depth buffer (depth information) while being able to be rendered off-screen. I chose it as it is the best option possible if I want to train an AI agent that plays Doom.

I think this project would be suitable as my final year project since it has a relatively high difficulty, and the end goal is not impossible to achieve. It also has a lot of potential for additional features to be implemented if my progress was better than expected (such as the abovementioned depth prediction extension). I am personally interested in Doom's speed-running community as well and if the project goes well the speed-running potential of the AI agent can also be tested. ZDoom being a source-port means that popular speed-run exploits are mostly patched or close-to-impossible to put off, thus it was quite unpopular within the speed-run community. However, as shown by DeepMind's hide-and-seek agents [1], AI agents can discover exploits to the simulation environment they are in (VizDoom in this case) and it's possible that my AI agent would be able to discover ways to perform exploits for speed-runs.

2. Aims and Objectives

- Train a Deep Q-Learning-based RL model that controls the movement and rotation of the AI agent.
    - The model should learn to rotate the player such that the crosshair at the center of the screen should always be close to the nearest enemy unit in the horizontal direction (Doom ignores the vertical (y-axis) position when calculating if a bullet hits an object).
        - **Risk:** learning to rotate towards the nearest enemy may result in the model being unable to learn to move correctly towards health items/armor items/exits.
        - **Mitigation:** careful fine tuning of hyperparameters and longer training times may help.
    - The model should learn to move the player such that the player is likely to approach health items and armor items when the health is not full.
        - **Risk:** learning to approach health/armor items instead of finding covers may cause the player to get shot by enemies and die in earlier epochs of training, therefore providing negative rewards and encourages the model to not approach these items.
        - **Mitigation:** it may be helpful to only incentivize the model to approach these items in later epochs of training stage when the player is less likely to die when approaching these items.
    - **[Optional]** The model should learn to move the player to explore the map in a greedy-search style and find the exit eventually.
        - **Risk:** the model may not converge with respect to this goal as it may not have enough depth to simulate a non-linear function for maze navigation using only RGB-D input.
        - **Mitigation:** increasing the depth of the model and using a GPU with more on-board video memory to train the deeper model would mitigate this issue.
    - **[Optional]** The model could learn to open doors that open without a key if there is time to investigate this possibility.
        - **Risk:** this objective is trivial on its own, the only issue is that only relatively complex maps would have doors to open and the model may not be able to navigate through those maps while staying alive. A mitigation for this problem is the mitigations of other objectives' risks combined.
    - **[Optional]** The model could learn to speed-run through maps of relatively high complexity if there is time to investigate this possibility.
        - **Risk:** there may not be time to investigate into achieving this objective, no mitigation.
- Train a Deep Q-Learning-based RL model that controls the firing of equipped weapons.
    - The model should learn to fire the equipped weapons when an enemy approaches the crosshair in the horizontal direction.
        - **Risk:** the agent may lose bullets on purpose if the firing action has a high reward.
        - **Mitigation:** give negative rewards when bullets are wasted.
    - The model should learn to predict the movement of enemies on screen with the RGB-D input.

            - **Risk:** the model may not converge with respect to this goal as it may not have enough depth to predict the movement of enemies on screen.
            - **Mitigation:** either the mitigation mentioned for a similar risk, or instead configure VizDoom to label enemies' pixels on screen and use the RGB-D information to learn to predict the motion vector for every "enemy pixel" on screen.
        - The model should try to shoot exploding barrels when one of them is near the crosshair and is far away from the player.
            - **Risk:** in earlier epochs of the training stage the player may die multiple times to an exploding barrel close to it, and thus encouraging it to not shoot at the exploding barrels.
            - **Mitigation:** it may be helpful to only incentivize the model to approach these items in later epochs of training stage when the player is less likely to fire randomly and hitting the exploding barrels when the model does not mean to.
- Apply the two RL models to more complex scenarios where both models are tested.
    - **Risk:** the models may not work well with each other, for example if the shooting model was trained with random or no movement, it may overfit and perform poorly with the moving model controlling the movement and rotation.
    - **Mitigation:** the shooting model should be trained with the walking model making decisions for movement and rotation.
- **[Optional]** Apply the two RL models to a proportion of the first map of the actual Doom game.
    - **Risk:** the moving model may not be able to navigate through the complex map
    - **Mitigation:** improve the design of the moving model
    - **Risk:** the shooting model may not perform well in areas with little training experience in since the further into the map the less training is received.
    - **Mitigation:** train the model with play sessions that have the starting position randomized.
- **[Optional]** Train a SPN-based model that predicts depth information with only on-screen pixels as input for its inference stage, the training stage can use sparse spatial point clouds with the depth information provided by VizDoom.
    - **Risk:** this may be time-consuming, and is an ambitious project on its own, might be outside of the scope of this project.
    - **Mitigation:** it would only be attempted if I am more than 2 weeks ahead of the schedule, and I have co-authored a paper in similar area so my colleagues may be of help when I'm stuck.

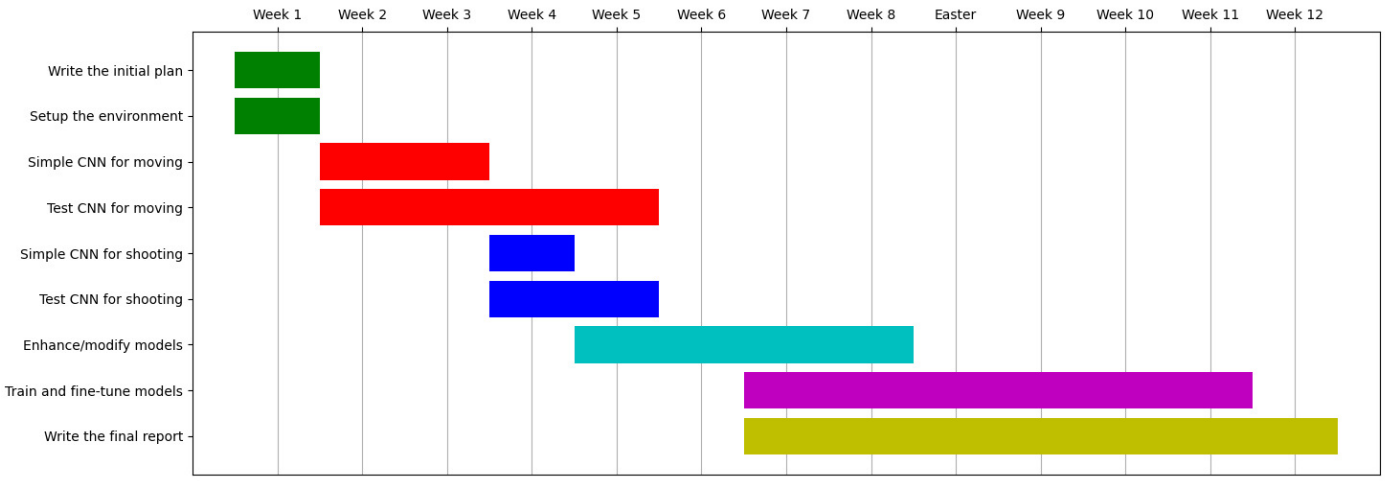
3. Feasibility

There exist two issues that may affect the feasibility of this project:

1. **[Unlikely, high severity]** A more powerful GPU with more video memory may be required as I only have an RTX 2070 mobile with 8GB of video memory, which is barely enough for this project.
2. **[Likely, low severity]** VizDoom, PyTorch or Python 3.10.9 may have compatibility issues with my Windows 11 operating system. Some components of Anaconda (the Python distribution I'm using) have shown compatibility issues already, so the risk is not non-existent.

A solution to both issues is to utilize Supercomputing Wales (SCW) or our university's Linux lab, I have already created my account for SCW successfully and is currently preparing to apply for my project. Issue 2 can also be detected by extensive testing and mitigated with a new Python 3.6 (more stable) virtual environment or a second operating system. Some basic tests have been carried out and no compatibility issue has surfaced on my machine so far, but Windows 11's automatic update feature may introduce new bugs.

4. Work Plan



I have already finished setting up the environment for this project, both PyTorch and VizDoom have been tested on my machine. For week 2 and 3 I plan to design a simple Convolutional Neural Network (CNN) [4] for the "moving model" of my RL agent, with enough depth it should be able to control the movement of player character well at least in simpler scenarios like "basic". Scenarios like "health gathering" can be used to train the moving model to collect health items and any scenario can be used to train the model to rotate toward nearby enemies. I plan to modify the moving model and train it into the "shooting model" in week 4, then it would be tested with moving model for 1-2 weeks. I would spend at least 4 weeks (5-8) modifying my models and changing their designs, I may use the easter holiday for it if the performance is poor. The training of the partially finalized models and fine-tuning of hyper parameters would start from week 7 or week 8 depending on progress and this would continue into week 11. I plan to start writing the final report in week 7, but in the first few weeks I would write only a little bit of the report due to the models still not finalized. I can edit the obsolete part of the report as more progress are made and by week 12 the fine-tuning would have ended so I have plenty of time to update the improvements. I have agreed with my supervisor to meet online every Tuesday at 12pm and I would typically work from Thursday to Sunday for this project.

5. References

1. Baker, B. et al. (2020) Emergent tool use from multi-agent Autocurricula, arXiv.org. Available at: https://arxiv.org/abs/1909.07528.
2. Kempka, M. et al. (2016) "Vizdoom: A doom-based AI research platform for Visual Reinforcement Learning," 2016 IEEE Conference on Computational Intelligence and Games (CIG) [Preprint]. Available at: https://doi.org/10.1109/cig.2016.7860433.
3. Lample, G. and Chaplot, D.S. (2017) "Playing FPS games with deep reinforcement learning," Proceedings of the AAAI Conference on Artificial Intelligence, 31(1). Available at: https://doi.org/10.1609/aaai.v31i1.10827.
4. LeCun, Y. et al. (1989) "Backpropagation applied to handwritten zip code recognition," Neural Computation, 1(4), pp. 541–551. Available at: https://doi.org/10.1162/neco.1989.1.4.541.
5. Merced, S.L.U.C. et al. (2017) Learning affinity via spatial propagation networks: Proceedings of the 31st International Conference on Neural Information Processing Systems, Guide Proceedings. Available at: https://dl.acm.org/doi/10.5555/3294771.3294916.
6. Mnih, V. et al. (2013) Playing Atari with deep reinforcement learning, arXiv.org. Available at: https://arxiv.org/abs/1312.5602.
7. Mnih, V. et al. (2015) Human-level control through deep reinforcement learning, Nature News. Nature Publishing Group. Available at: https://www.nature.com/articles/nature14236.