Reinforcement Learning Research Project:

Deep Q-Network Agent for Snake Game

Oluwaleke Umar Yusuf

M.Sc. Robotics, Control And Smart Systems

Fall 2021

OUTLINE

- Background
- Project Objective & Scope
- Perception-Action-Learning Loop
- Deep Q-Networks
- Snake Game Dynamics
- Agent & Environment
- Incentives & States
- Agent Training & Testing
- Discussion

Background

- **Reinforcement Learning (RL)** involves mapping situations to actions to maximize a numerical reward signal.

- Within the AI context, **RL** refers to goal-oriented algorithms which learn how to achieve a complex objective or maximize along a particular dimension over many steps.

- **Deep Reinforcement Learning (DRL)** combines machine learning with the framework of RL to help agents reach their goals and objectives.

- **Deep Q-Network (DQN)** is a **DRL** model based on Q-Learning – a traditional **RL** off-policy, temporal difference algorithm – which learns policies directly from (visual) high-dimensional inputs.

Project Objective & Scope

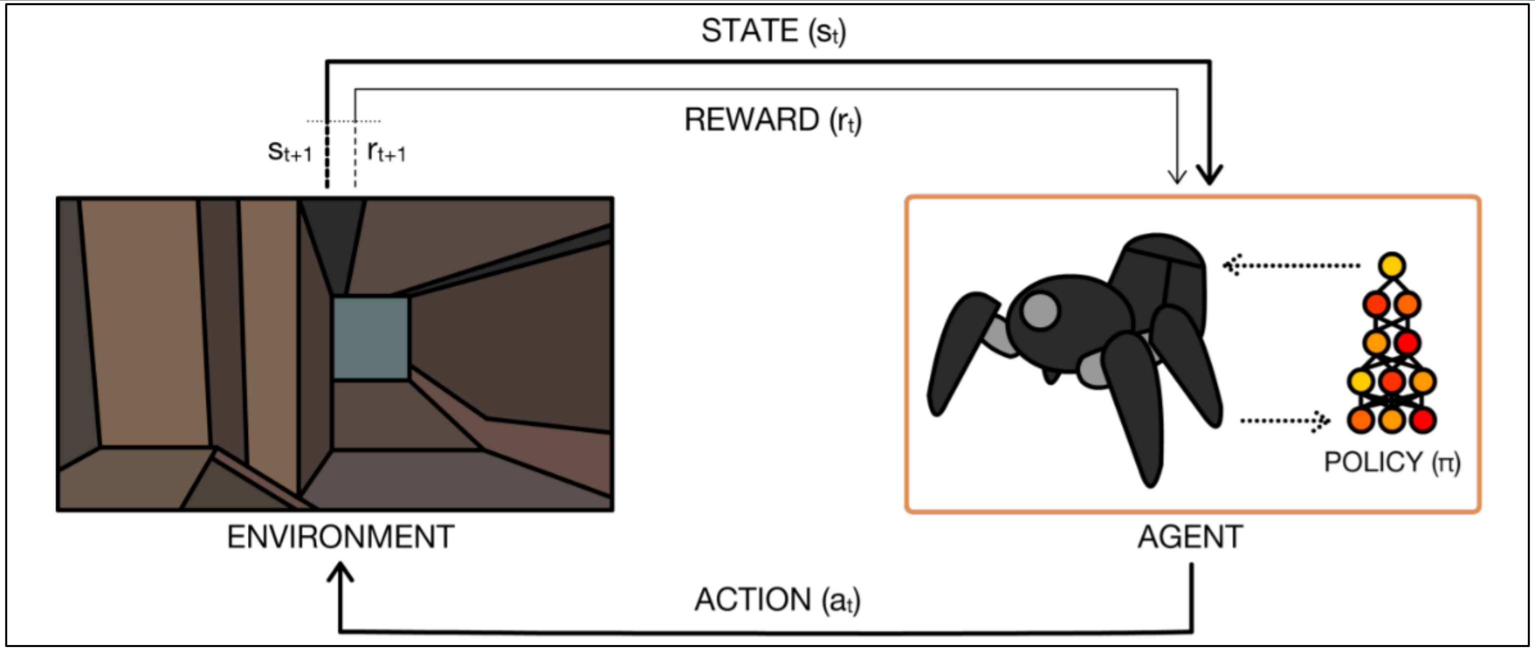
Project Objective:

- Python implementation of a simple DQN agent for the classic game of Snake.
- Using the PyTorch machine learning framework and the Pygame library.

Project Scope:

- Examine the theory behind the traditional Q-Learning algorithm.
- Study how the algorithm has been integrated with modern ML algorithms and neural network architectures to create Deep Q-Networks.

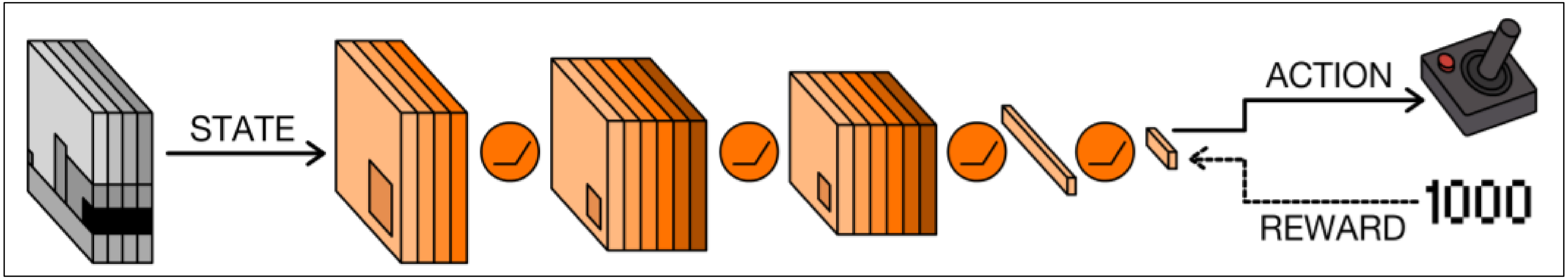
Perception-Action-Learning Loop



The Perception-Action-Learning Loop: *At time t, the agent receives state s_t from the environment. The agent uses its policy to choose an action a_t.*

Once the action is executed, the environment transitions a step, providing the next state s_{t+1} and feedback in the form of a reward r_{t+1}. The agent uses knowledge of state transitions, of the form (s_t, a_t, s_{t+1}, r_{t+1}), to learn and improve its policy.

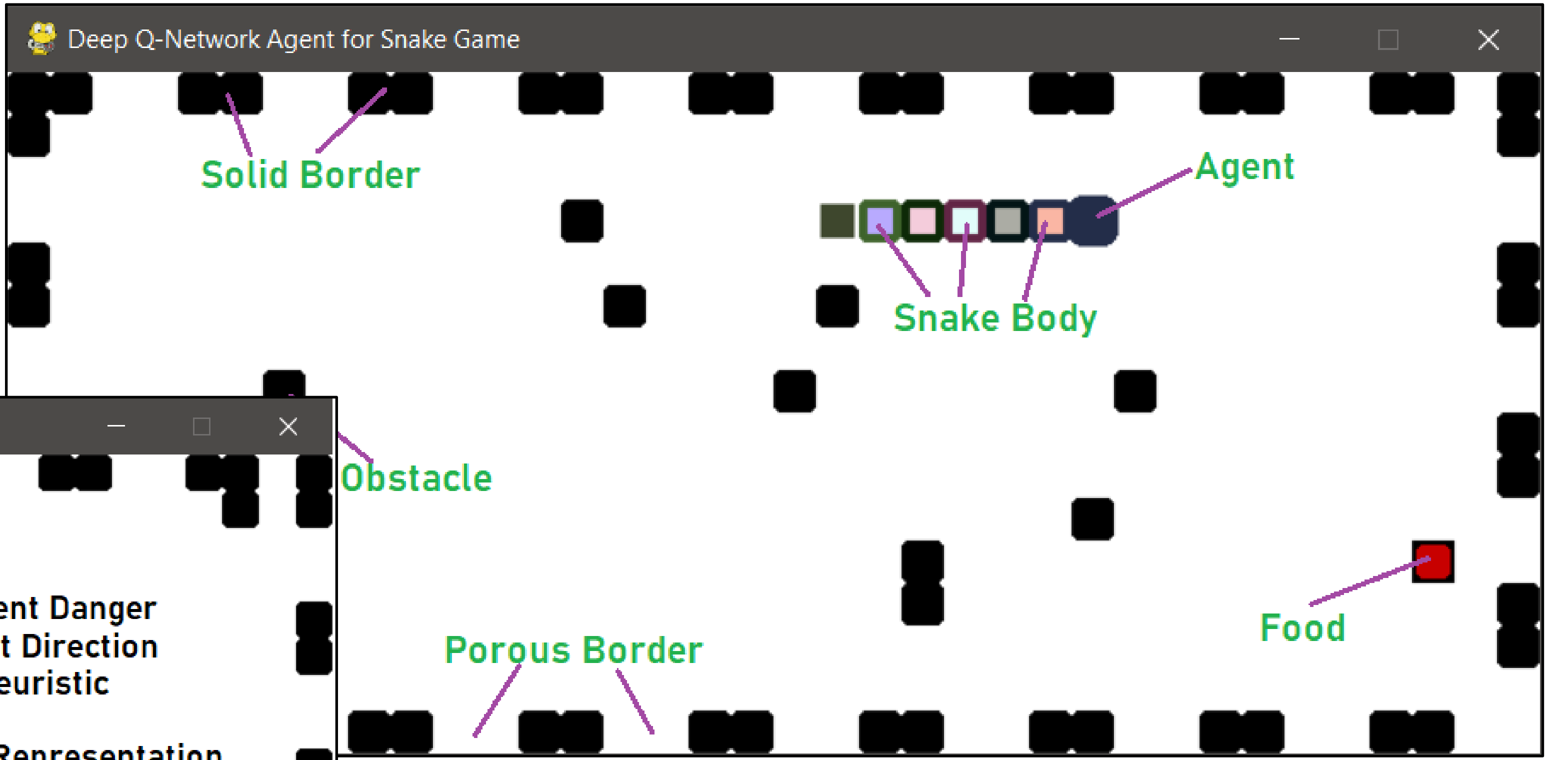
DEEP Q-NETWORKS

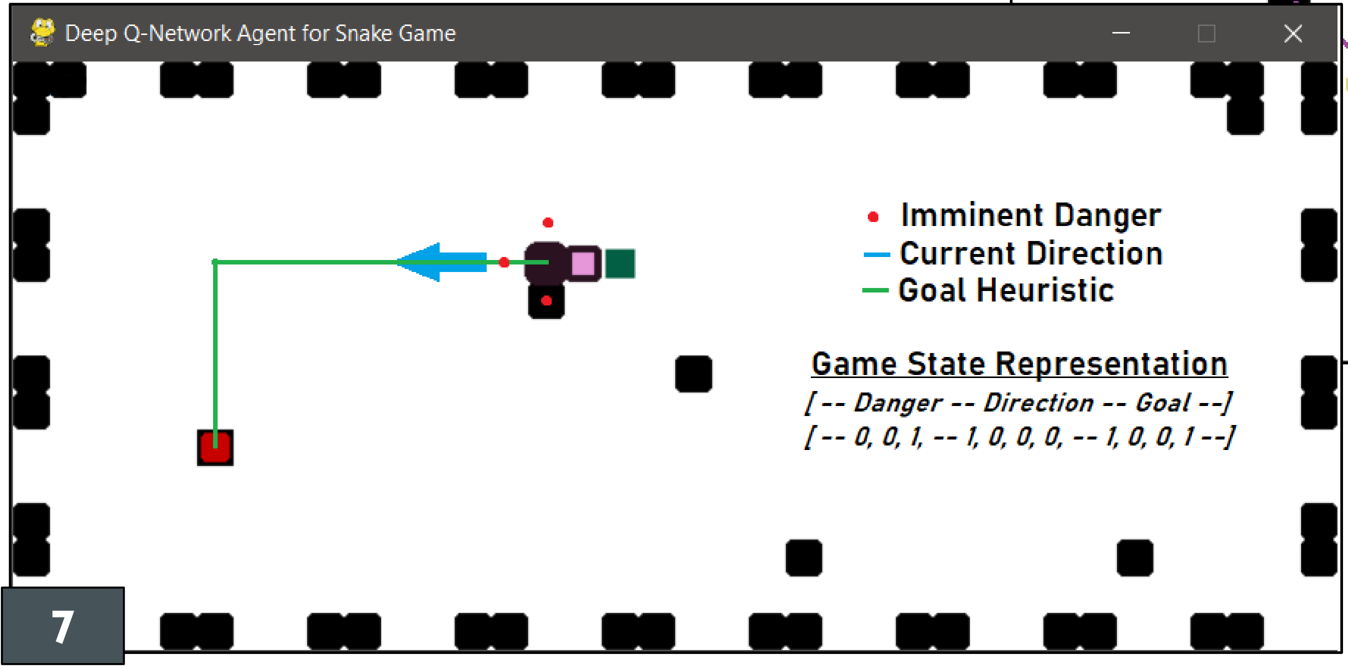


The Deep Q-Networks: *The network takes the state and processes it with (convolutional and) fully connected layers, with ReLU non-linearities in between each layer. At the final layer, the network outputs a discrete action corresponding to one of the possible control inputs for the game. Given the current state and chosen action, the game returns a new score.*

The DQN uses the reward – the difference between the new score and the previous one – to learn from its decision. More precisely, the reward is used to update its estimate of Q, and the error between its previous estimate and its new estimate is backpropagated through the network.

Agent, Environment, Incentives & States





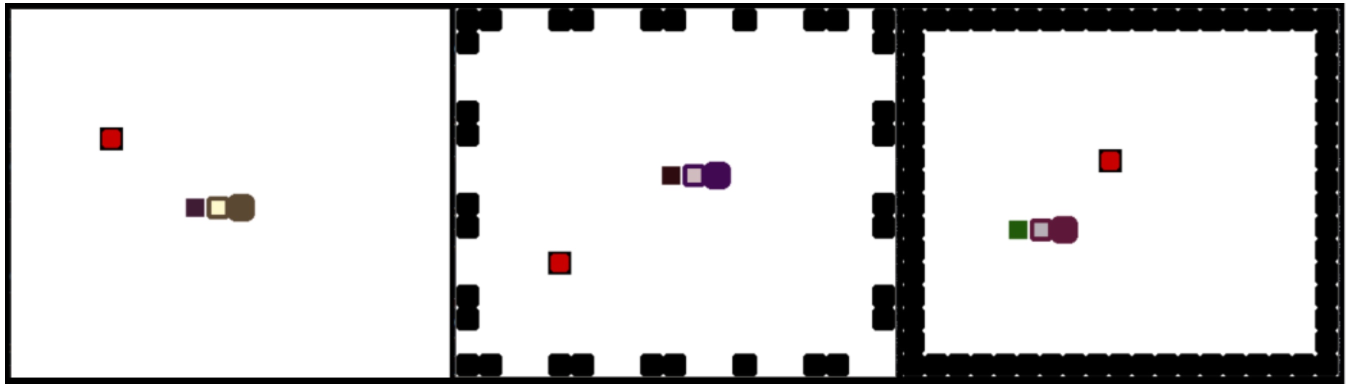
Agent & Environment

Agent:

- The snake's head is the agent.
- The interaction of the snake's head with the environment determines the progress of the game.

Environment:

- Snake’s Tail
- Walls (Borders)
  A. Open Borders (left)
  B. Hybrid Borders (middle)
  C. Closed Borders (right)
- Food
- Obstacles



Incentives & States

Incentives:

- Collisions between the agent and the food are **rewarded (+10)**.
- Collisions between the agent and the snake's tail, solid borders, and obstacles are *penalized (-10)*.
- Lack of progress – no collision with the food, snake's tail, solid borders, and obstacles after a specified period – is also *penalized (-10)*.

States:

- The game state is abstracted into 11 binary variables – representing imminent danger (collision), agent's position (direction), and goal (food) location.

***1.** Danger Straight* | ***2.** Danger Right* | ***3.** Danger Left*

***4.** Direction Left* | ***5.** Direction Right* | ***6.** Direction Up* | ***7.** Direction Down*

***8.** Food Left* | ***9.** Food Right* | ***10.** Food Up* | ***11.** Food Down*

Agent Training & Testing

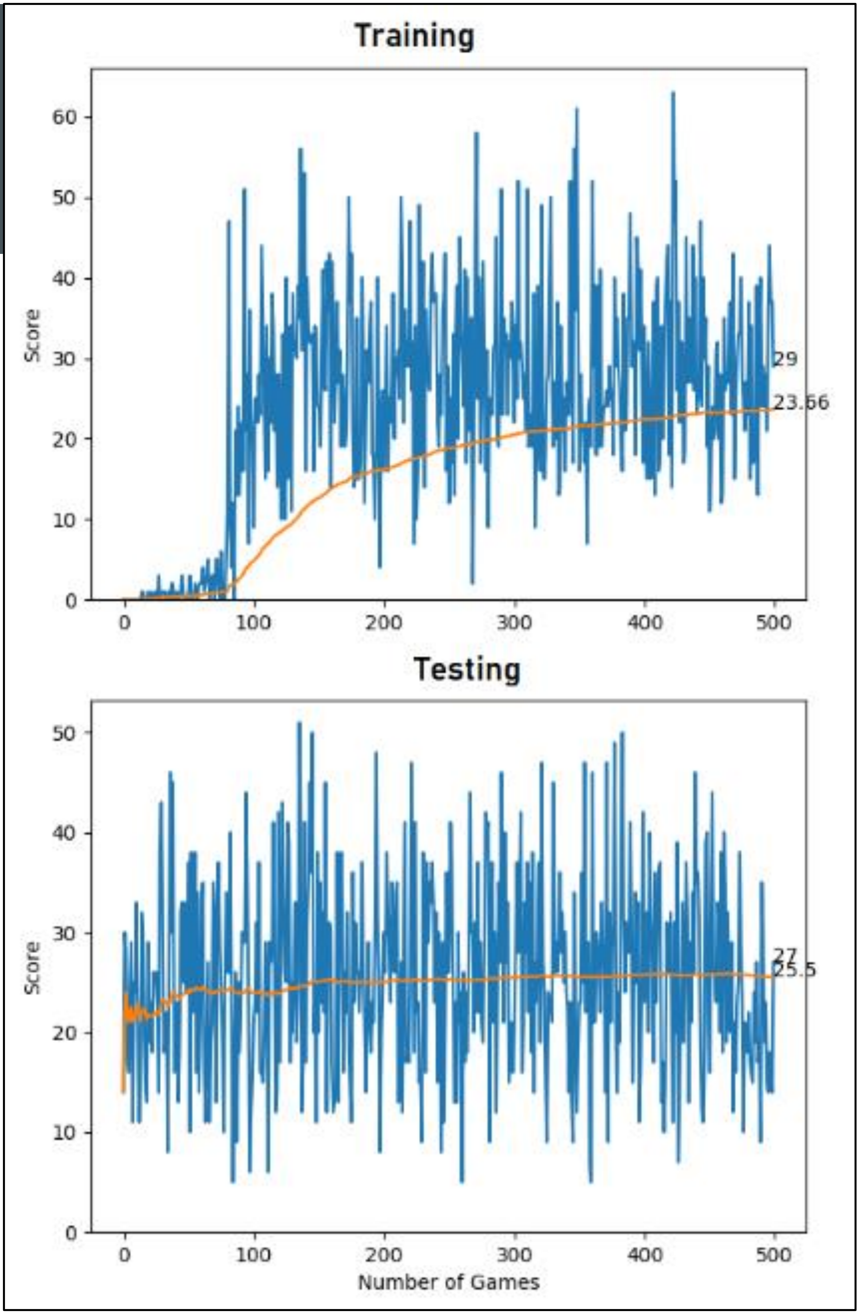
- Several deep Q-network agents were trained on different combinations of environment and incentives – borders, obstacles, rewards, and penalties.
- Agents trained on one combination of environment and incentives were tested on other combinations of environment and incentives.
- This helps to investigate how much the agent's training generalizes when faced with novel situations.
- The agents were trained for 500 games, and each test also lasted for 500 games.

	Agent, Environment & Incentives
1	Simple Neural Network Closed Borders Equal Rewards and Penalties *0 Obstacles
	Simple Neural Network Closed Borders Equal Rewards and Penalties *10 Obstacles
	Simple Neural Network Closed Borders Equal Rewards and Penalties *20 Obstacles
2	Simple Neural Network Hybrid Borders Equal Rewards and Penalties *0 Obstacles
	Simple Neural Network Hybrid Borders Equal Rewards and Penalties *10 Obstacles
	Simple Neural Network Hybrid Borders Equal Rewards and Penalties *20 Obstacles
3	Simple Neural Network Hybrid Borders Different Rewards and Penalties *[collision=-10, food=+30, no progress=-10]*
4	Testing < Closed > Borders Agent in Environment with < Hybrid > Borders
	Testing < Hybrid > Borders Agent in Environment with < Closed > Borders
5	Complex Neural Network *[11 x 256 x 512 x 3]* Closed Borders Equal Rewards and Penalties 0 Obstacles

Discussion

Training & Testing Scores:

- The **overall maximum score (78)** was achieved by an agent trained on hybrid borders with obstacles and tested in an environment with no obstacles.
- Although final (blue) scores per game tend to be noisy,. agents generally performed as well or better during testing as in training
- The mean (orange) training score gradually improves over successive games while the mean testing score stays constant and close to the training value.



DISCUSSION

Ablation Studies:

- **Effect of Borders:**

  Agents trained on hybrid borders had higher maximum scores than agents trained on closed borders. However, the hybrid-borders agents had lower mean scores than closed-borders agents.

- **Effect of Obstacles:**

  Agents trained without obstacles performed better – higher maximum and mean scores – than agents trained with obstacles during testing, regardless of the border type.

- **Effect of Neural Network Architectures:**

  An agent trained with a more complex architecture *(11x256x512x3)* showed no overall improvement in performance over the agent trained with the simple architecture *(11x256x3)* in the same environment.

DISCUSSION

Ablation Studies:

- **Effect of Incentives (Rewards & Penalties):**

  An agent trained with differentially-weighted incentives performed worse than an agent trained with equally-weighted incentives in the same environment.

- **Effect of Cross-Domain Training and Testing:**

  The agent trained on closed borders with no obstacles was tested in an environment with hybrid borders and 0/20 obstacles, and vice-versa. The hybrid-borders agent had better maximum scores while the closed-borders agent had better mean scores.

Conclusion

- The three metaheuristics of **game state representation**, **incentives scheme**, and **neural network architecture** are much more critical and harder to figure out than the game environment and its components.
- All three metaheuristics are linked, and the suboptimality of any will affect the agent's training.
  - *A game state representation without sufficient information or irrelevant information will not give good loss values when combined with the incentives.*
  - *A suboptimal incentive scheme might interact negatively with the state representation and neural network and induce undesirable behaviour in the agent.*
  - *A badly-chosen neural network architecture might either overfit or underfit and negatively impact the agent's training.*
- The DQN is a simple yet powerful RL/ML method that is easy to implement, fast to train and yields agents with outstanding performance.

References

I. R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: A Bradford Book, 2018.

II. N. Chris, ‘A Beginner’s Guide to Deep Reinforcement Learning’, Pathmind. *Available: http://wiki.pathmind.com/deep-reinforcement-learning. [Accessed: 12-Oct-2021].*

III. C. J. C. H. Watkins, ‘Learning from Delayed Rewards’, PhD Thesis, King’s College, Oxford, 1989.

IV. ‘About - Pygame’. *Available: https://www.pygame.org/wiki/about. [Accessed: 13-Oct-2021].*

V. K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, ‘A Brief Survey of Deep Reinforcement Learning’, IEEE Signal Process. Mag., vol. 34, no. 6, pp. 26–38, Nov. 2017, doi: 10.1109/MSP.2017.2743240.

VI. ‘Snake (video game)’, Encyclopedia Gamia Archive Wiki. *Available: https://gamia-archive.fandom.com/wiki/Snake_(video_game). [Accessed: 11-Nov-2021].*

VII. ‘Pixilart - Snake Game (Gif Test)’, Pixilart. *Available: https://www.pixilart.com/art/snake-game-gif-test-16c3630a9147a08. [Accessed: 11-Nov-2021].*

VIII. L. Patrick, Teach AI To Play Snake! Reinforcement Learning With PyTorch and Pygame. *https://github.com/python-engineer/snake-ai-pytorch, 2021.*

IX. L. Patrick, ‘Teach AI To Play Snake - Practical Reinforcement Learning With PyTorch And Pygame’, Python Engineer. *Available: https://python-engineer.com/posts/teach-ai-snake-reinforcement-learning. [Accessed: 12-Oct-2021].*