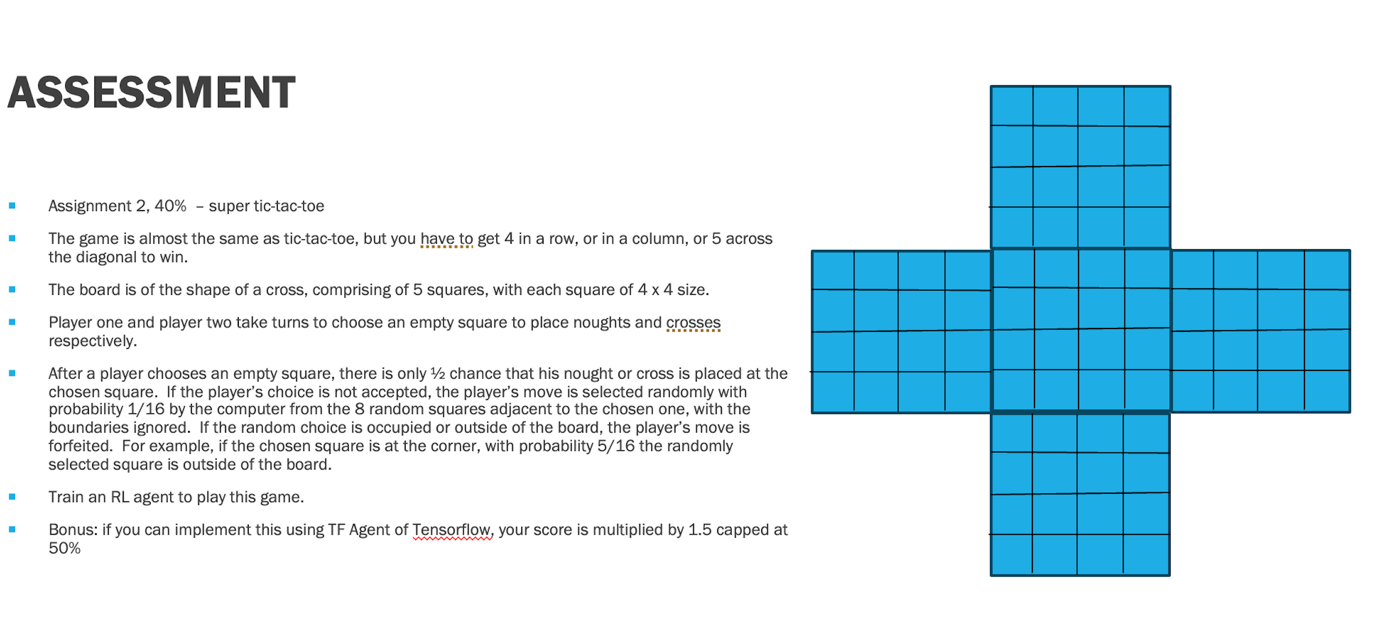
**1. Project requirement**

**2. Environment Design**

This part involved implementing a custom game environment simulating the rules of cross-shaped super tic-tac-toe. A smoke test confirmed the environment's correct behaviour. Key features include:

* 5 interconnected 4x4 grids forming a cross.
* Turn-based gameplay with randomness (50% move acceptance).
* Fallback move selection from surrounding squares with edge and occupancy constraints.

**3. Agent Design**

The RL agent was based on a neural network architecture that included:

* **Policy head**: Outputs action probabilities.
* **Value head**: Estimates the expected outcome of the game from a given state.

Monte Carlo Tree Search (MCTS) was implemented for decision-making during self-play, introducing Dirichlet noise to encourage exploration.

**4. Training Procedure**

The training process included:

* Self-play generation to populate a replay buffer.
* Training using TensorFlow, optimizing a loss function computed from cross-entropy (policy) and mean squared error (value).
* Evaluation against a random agent and previous versions to track progress. If the win rate is more than 55%, we keep the model.

**5. Training Results**

The training involved alternating between the agent playing games against itself (self-play) and learning from those games using a neural network and Monte Carlo Tree Search.

Over three rounds of training:

* The agent steadily improved in the first two rounds.
* In the third round, performance dropped slightly, so I kept the second model (checkpoint ckpt-23) as the best version.
* The best model achieved high win rate against a random agent and showed solid improvement over earlier versions.

While the results are promising, performance could likely be improved further with more computation, deeper MCTS simulations, and longer training.

**6. Limitations**

The following constraints and challenges limited performance:

* **Hardware**: Experiments were conducted on an Apple M1 MacBook, limiting computational throughput.
* **Simulation depth**: MCTS used only 50 simulations per move due to compute constraints.
* **Buffer size and training episodes**: More extensive training and larger buffers could yield improved performance.

**7. Possible Improvements**

Future work could explore:

* Increasing MCTS simulations per move (e.g., 800).
* Utilizing cloud compute to scale training.
* Fine-tuning hyperparameters such as learning rate, network architecture, and exploration parameters.
* Incorporating additional regularization or improved loss functions.