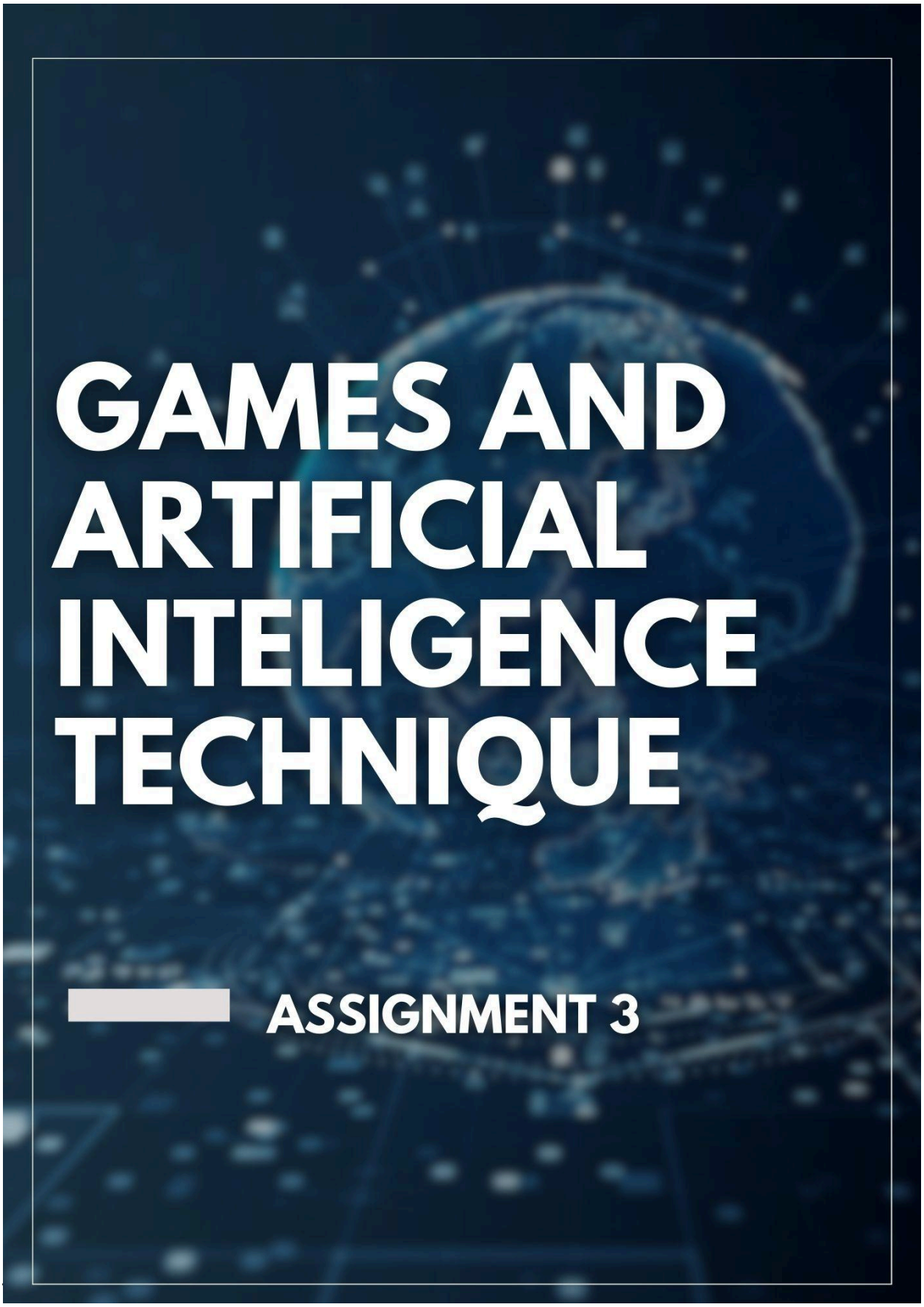
GAMES AND
ARTIFICIAL
INTELIGENCE
TECHNIQUE
ASSIGNMENT 3

1. The rewards and episode termination logic

1.1 The rewards logic

The rewards system is designed to incentivize desirable behaviors and penalize undesirable ones. According to the code that our group has implemented, the rewards logic will be explained below:

HandleHitEnemy(): The reward is -0.5 if the player has more than one life. The penalty: will end the episode with a phase penalty if the player has one life or less. Rationale: discourages the agent from being hit by enemies, with a harsher penalty when lives are low. First, we use a reward with -1.0, but because the entropy is getting consecutive, we decided to lower it to make the AI explore more. The function is used in the Enemy class. It is called when the enemy hits the player.

HandleHitBoss(): Reward is -0.5 if the player has more than three lives. Penalty will end the episode with a phase penalty if the player has three lives or less. The rationale is Similar to enemy hits, but the threshold for the penalty is higher due to the higher threat posed by the boss. First, we use a reward with -1.0, but because the entropy is getting consecutive, we decided to lower it to make the AI explore more. It is used in Boss class. It is called when the boss hits the player.

HandlePowerup(): Reward is +2.0. Rationale encourages the agent to seek and collect power-ups to improve performance. It is used in power-up classes.

HandleShootEnemy(): Reward is +0.5 for shooting an enemy. Rationale encourages the agent to engage enemies actively. However, we just use 0.5 to make the AI not only focus on shooting the enemy and forget to destroy the spawner. It is used in the Enemy class.

HandleShootSpawners(): The reward is +1.0 for shooting a spawner. Rationale encourages the agent to target spawners, which are likely critical to game progression. With 1.0, it will focus on shooting the spawners when it has changed. It is used in boss and enemy spawners.

HandleShootBoss(): Reward is +2.0 for shooting the boss. Rationale Significantly incentivizes the agent to attack the boss, a primary objective. We give it a higher reward compared to HandleShootSpawners() to make it focus on shooting the boss rather than the spawners. It is used in boss class.

HandleDestroySpawner(): Reward is +1.0 for destroying a spawner. Rationale rewards the agent for eliminating sources of new enemies. It is used in both enemy and boss classes.

HandleCloseToBoss(): Reward is -0.5f for being close to the boss. Rationale discourages the agent from being close to the boss.

HandleCloseToBorder(): Reward is -1.0 for being close to the border. Rationale discourages the agent from approaching the game area borders, likely to avoid boundary-related issues. It is used in the Player class.
HandleDestroyBoss(): Reward is +2.0 for destroying the boss. Rationale highly rewards the agent for achieving a significant game milestone. It is used in boss class.

1.2. Episode termination logic

The episode termination logic is primarily triggered when the player's status reaches a critical state.
HandleHitEnemy() and HandleHitBoss()

- **Condition**: In HandleHitEnemy(), the player has one life left and if the player hits the enemy, EndEpisode() will be called. In HandleHitBoss(), the player has three lives left, and if the player hits the boss, EndEpisode() will be called. In general, Episode termination will be called if the player dies.
- **Action**: Records the current game phase using `Academy.Instance.StatsRecorder.Add("Phase Reached", phaseReached)` and then call `EndEpisode()`.
- **Rationale**: Recording the phase reached helps track the agent's performance over episodes, while ending the episode stops further actions, forcing the agent to reset and start over, reinforcing the penalty for reaching a critical state.

2. The State Representation

CollectObservations

In the `PlayerAgent` class, the `CollectObservations` method is responsible for providing these observations to the agent.
Player's Position:

- The agent observes its own position on the x and y axes.
- `sensor.AddObservation(transform.position.x);`
- `sensor.AddObservation(transform.position.y);`

Ray Perception Sensor 2D

Ray Perception Sensors are typically used to simulate an agent's vision or sensing capabilities by casting rays (like laser beams) into the environment. These rays can detect objects, their distances, and often their tags or types. We have 4 rays:
RayPerceptionSensorEnemy: To detect all the normal ships with layer "Enemy".

RayPerceptionSensorPowerups: To detect the power up with layer “Power Up”.
RayPerceptionSensorSpawners: To detect all the spawners with layer “Spawners”.
RayPerceptionSensorBoss: To detect all the bosses with layer “Boss”.

3. The Tuning of the hyperparameters in the .yaml file.

3.1. Player.yaml file and player-new.yaml file

Due to the entropy in TensorBoard, from the player.yaml file which is the original one, we see that it is not stable when the steps reach a large number. Therefore, we created a new file which is player-new.yaml file with a higher value of batch-size and Buffer-size.

Batch-size: in the player.yaml file is 64 and in the player-new.yaml file is 256.

Buffer-size: in the player.yaml file is 256 and in the player-new.yaml file is 2048.

Learning Rate (`learning_rate: 1e-3`): `1e-3` is a common starting point; we adjust it based on observed training performance.

Training Settings (`max_steps`, `time_horizon`, `summary_freq`, `checkpoint_interval`, `keep_checkpoints`):

`max_steps: 999999999` specifies the maximum number of training steps.

`time_horizon: 64` sets the number of steps to run in each environment before updating the model.

`summary_freq: 1000` determines how often to write summary statistics.

`checkpoint_interval: 10000` specifies how often to save model checkpoints.

`keep_checkpoints: 999999` sets the maximum number of checkpoints to keep.

3.2. playerMultiEnv.yaml file

Due to the entropy in TensorBoard, we created a new file which is playerMultiEnv.yaml file with a higher value of batch-size and Buffer-size compared to the player.yaml file.

Batch-size is 512, and **Buffer-size** is 4096 in the playerMultiEnv.yaml file.

env_settings Section (Environment Settings)

- **env_path**: Path to the environment executable or directory containing environment files. This specifies where the RL agent interacts and learns.
- **num_envs: 4**: Number of parallel environments to run. Running multiple environments in parallel can speed up training by allowing the agent to gather more diverse experiences simultaneously.
- **seed: -1**: Seed for random number generation. `-1` typically means the seed is randomly chosen or not explicitly set.

We add this environment setting because reinforcement learning involves an agent interacting with an environment, learning from its actions, and receiving feedback (rewards). The `env_settings` section ensures the RL framework knows how to initialize, manage, and communicate with the environment correctly. Also, Specifying `num_envs` allows for efficient parallelization of experience collection. This is beneficial for speeding up training, especially in environments where episodes are relatively short or computationally intensive. Furthermore, Setting a specific `seed` ensures that experiments produce consistent results across different runs. This is crucial for verifying algorithm behavior, debugging, and ensuring research findings are reliable and reproducible.

4. Adjustments made for control style 2

We adjust the Actions in Behaviour Parameters. Discrete Branch is two, which is 9 for Branch zero size and 2 for Branch 1 size. Besides, we also add *sensor.AddObservation(transform.rotation)* in CollectObservation() in PlayerAgent and change to *playerScript.Movement_V2(action.strafe).*