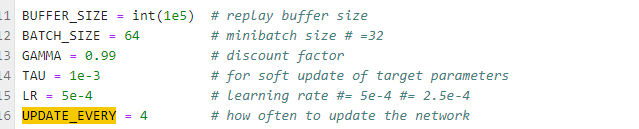
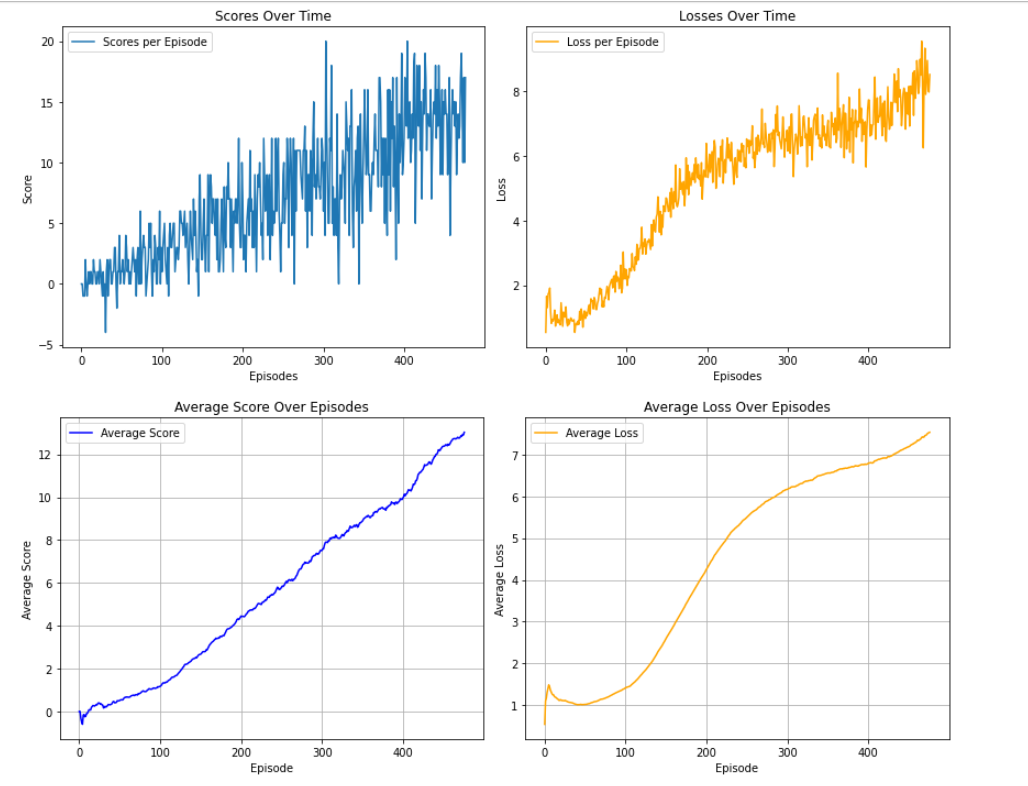
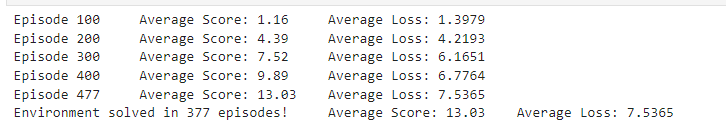
TEST 1









Hyperparametre :

en jointe d'autre résultat de l'entrainement. Au vu de résultats que je viens de te donné dans notre derniere discution et ce celle ci que conseil tu de modifier et pour quelle valeur : "BUFFER\_SIZE = int(1e5) # replay buffer size # int(1e5)

BATCH\_SIZE = 64 # minibatch size # =32

GAMMA = 0.99 # discount factor

TAU = 1e-3 # for soft update of target parameters # 1e-3

LR = 5e-4 # learning rate déja testé = 7.5e-4 # 2.5e-4

UPDATE\_EVERY = 4 # how often to update the network

optimizer = Adam (déja testé : SGD=Stochastic Gradient Descent, Adagrad, RMSprop)

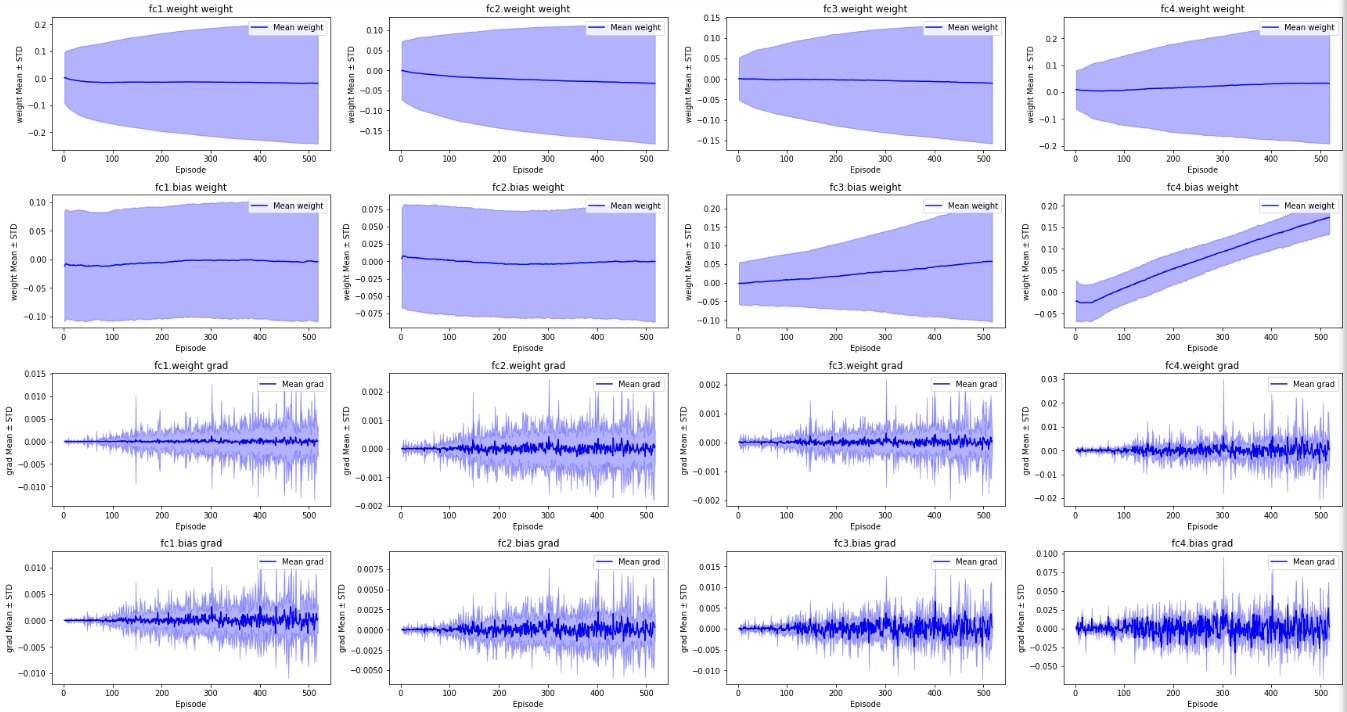
loss function = mse\_loss

layers : fc1\_units=64, fc2\_units=128, fc3\_units=64

activations function : relu

epsilone : eps\_start=1.0, eps\_end=0.01, eps\_decay=0.995

maximum number of timesteps per episode : max\_t=1000



**Observations :**

1. **Variabilité des poids et gradients** :
   * Les poids de la première couche (**fc1.weight**) semblent relativement stables au fil des épisodes, avec une légère tendance à la baisse de la moyenne, ce qui est normal au fur et à mesure de la convergence.
   * Les poids des couches suivantes (**fc2**, **fc3**, **fc4**) montrent une plus grande variabilité, avec des écarts-types qui augmentent au fil des épisodes. Cela peut indiquer que le modèle continue de s'adapter et de modifier considérablement ses paramètres, ce qui pourrait être le signe d'une exploration continue ou d'une instabilité dans l'apprentissage.
   * Les gradients sont particulièrement bruyants, ce qui est attendu dans un processus d'apprentissage par renforcement, mais les pics de certains gradients (**fc4.weight\_grad** par exemple) suggèrent des mises à jour potentiellement importantes et irrégulières des poids.

**Suggestions :**

1. **Taux d'apprentissage** :
   * Si le taux d'apprentissage est trop élevé, il pourrait causer des mises à jour trop importantes et empêcher la convergence. Envisagez de réduire le taux d'apprentissage ou d'utiliser un schéma de diminution adaptatif.
2. **Régularisation** :
   * La mise en œuvre de techniques de régularisation comme le dropout ou la normalisation par lots pourrait aider à stabiliser l'apprentissage et à réduire la variance des poids.
3. **Initialisation des poids** :
   * Envisagez d'utiliser une méthode d'initialisation différente qui pourrait être plus adaptée à votre architecture spécifique, comme l'initialisation de He pour les couches avec activations ReLU.
4. **Clipping des gradients** :
   * Pour contrôler les mises à jour importantes dues aux grands gradients, vous pourriez implémenter le clipping des gradients, qui consiste à limiter la valeur des gradients pendant la rétropropagation.
5. **Stratégies d'apprentissage avancées** :
   * Comme vous l'avez suggéré précédemment, implémenter des stratégies avancées telles que le Double DQN, le Prioritized Experience Replay ou le Dueling DQN pourrait potentiellement améliorer les performances et la stabilité de l'apprentissage.
6. **Analyse des épisodes** :
   * Examinez les trajectoires d'apprentissage et les décisions prises par l'agent dans des épisodes spécifiques où des pics de gradient sont observés pour comprendre ce qui pourrait causer ces variations.

En résumé, les graphiques suggèrent que bien que le modèle apprenne, il pourrait y avoir de la place pour améliorer la stabilité et la performance de l'apprentissage. Des ajustements sur les hyperparamètres, l'architecture et l'introduction de nouvelles techniques d'apprentissage pourraient tous être des avenues pour une amélioration supplémentaire.

Based on the results you've shared and the discussion we've had, here are some tailored suggestions for each parameter you've listed:

1. **BUFFER\_SIZE**: If you are not running into memory issues and the training time per episode is acceptable, you might not need to change this. A large buffer size allows for a more diverse range of experiences, which can help the agent learn a more general policy.
2. **BATCH\_SIZE**: You've mentioned using 64; you could experiment with a smaller batch size like 32. A smaller batch size might introduce more variance into the training updates, which can sometimes help the agent to escape local minima.
3. **GAMMA (Discount Factor)**: This is typically close to 1. If you find that the agent is not planning effectively over long time horizons, you could try increasing it slightly, but .99 is already quite high.
4. **TAU (Soft Update of Target Parameters)**: The value you're using (1e-3) is fairly standard. If your model is experiencing high variance in the loss, you might try decreasing it to 5e-4 to make the target network updates more conservative.
5. **LR (Learning Rate)**: You've mentioned testing different learning rates. If the loss is consistently increasing, it might be worth trying an even smaller learning rate, such as 1e-4, to see if that stabilizes the training.
6. **UPDATE\_EVERY**: How often to update the network is a balance between learning efficiency and computational expense. If you notice that the agent's performance is very noisy, you might want to update less frequently, perhaps every 6 steps instead of 4.
7. **Optimizer**: Since Adam is giving you the best results, sticking with Adam makes sense. You might want to experiment with its hyperparameters like **betas** and **eps** if you haven't already.
8. **Loss Function**: Mean Squared Error (MSE) is standard for regression tasks. If you're seeing unstable learning, you could consider Huber loss (smooth L1 loss), which is less sensitive to outliers than MSE.
9. **Layers and Activation Functions**: The architecture seems reasonable, but you could consider adding another layer or increasing the size of existing layers if the model isn't complex enough to capture the policy. Since you're using ReLU, make sure to monitor if many neurons are "dying". If that's the case, switch to LeakyReLU or ELU.
10. **Epsilon Decay**: The schedule for epsilon decay seems standard, but if you find the agent is not exploring effectively, you could slow down the decay or increase the minimum epsilon.
11. **Max Timesteps per Episode (max\_t)**: If the episodes are timing out before the agent reaches the terminal state, you could increase this, but only if the additional timesteps are likely to provide useful learning experiences.

Remember that hyperparameter tuning can be an iterative and sometimes unintuitive process. It's often helpful to only change one or two parameters at a time, so you can observe their effects more clearly. It might also be beneficial to implement a systematic hyperparameter optimization process like grid search or Bayesian optimization if you have the computational resources.

Based on the results you've shared and the discussion we've had, here are some tailored suggestions for each parameter you've listed:

1. **BUFFER\_SIZE**: If you are not running into memory issues and the training time per episode is acceptable, you might not need to change this. A large buffer size allows for a more diverse range of experiences, which can help the agent learn a more general policy.
2. **BATCH\_SIZE**: You've mentioned using 64; you could experiment with a smaller batch size like 32. A smaller batch size might introduce more variance into the training updates, which can sometimes help the agent to escape local minima.
3. **GAMMA (Discount Factor)**: This is typically close to 1. If you find that the agent is not planning effectively over long time horizons, you could try increasing it slightly, but .99 is already quite high.
4. **TAU (Soft Update of Target Parameters)**: The value you're using (1e-3) is fairly standard. If your model is experiencing high variance in the loss, you might try decreasing it to 5e-4 to make the target network updates more conservative.
5. **LR (Learning Rate)**: You've mentioned testing different learning rates. If the loss is consistently increasing, it might be worth trying an even smaller learning rate, such as 1e-4, to see if that stabilizes the training.
6. **UPDATE\_EVERY**: How often to update the network is a balance between learning efficiency and computational expense. If you notice that the agent's performance is very noisy, you might want to update less frequently, perhaps every 6 steps instead of 4.
7. **Optimizer**: Since Adam is giving you the best results, sticking with Adam makes sense. You might want to experiment with its hyperparameters like **betas** and **eps** if you haven't already.
8. **Loss Function**: Mean Squared Error (MSE) is standard for regression tasks. If you're seeing unstable learning, you could consider Huber loss (smooth L1 loss), which is less sensitive to outliers than MSE.
9. **Layers and Activation Functions**: The architecture seems reasonable, but you could consider adding another layer or increasing the size of existing layers if the model isn't complex enough to capture the policy. Since you're using ReLU, make sure to monitor if many neurons are "dying". If that's the case, switch to LeakyReLU or ELU.
10. **Epsilon Decay**: The schedule for epsilon decay seems standard, but if you find the agent is not exploring effectively, you could slow down the decay or increase the minimum epsilon.
11. **Max Timesteps per Episode (max\_t)**: If the episodes are timing out before the agent reaches the terminal state, you could increase this, but only if the additional timesteps are likely to provide useful learning experiences.

Remember that hyperparameter tuning can be an iterative and sometimes unintuitive process. It's often helpful to only change one or two parameters at a time, so you can observe their effects more clearly. It might also be beneficial to implement a systematic hyperparameter optimization process like grid search or Bayesian optimization if you have the computational resources.

Dans le contexte de l'apprentissage par renforcement, un "bon modèle" est souvent défini par sa capacité à maximiser la récompense cumulative dans l'environnement sur lequel il est entraîné. Cependant, il y a plusieurs facteurs qui peuvent être considérés lors de l'évaluation de la qualité d'un modèle d'apprentissage par renforcement :

1. **Performance**:
   * C'est la capacité de l'agent à atteindre ou dépasser un certain seuil de performance, comme un score moyen de 13.0 dans votre cas. La rapidité avec laquelle cet objectif est atteint est souvent un facteur important, mais ce n'est pas le seul.
2. **Stabilité**:
   * Un bon modèle devrait non seulement atteindre la performance désirée mais aussi la maintenir de manière stable au fil du temps. La stabilité de l'apprentissage est cruciale, car un modèle qui atteint rapidement une haute performance mais qui est très variable peut être moins fiable.
3. **Robustesse**:
   * La capacité de l'agent à bien performer face à des variations de l'environnement ou des conditions initiales est également un signe de qualité. Un modèle robuste peut généraliser son apprentissage à de nouvelles situations.
4. **Efficacité de l'apprentissage**:
   * La quantité de données (épisodes dans le cas de l'apprentissage par renforcement) nécessaires pour atteindre la performance cible est un autre critère. Un modèle qui apprend avec moins de données est généralement considéré comme plus efficace.
5. **Convergence**:
   * La vitesse et la fiabilité avec lesquelles un modèle converge vers une politique optimale sont importantes. Une convergence rapide est souvent souhaitable, mais pas au détriment de la stabilité ou de la robustesse.
6. **Complexité du modèle**:
   * Dans certains cas, un modèle plus simple qui atteint la performance requise est préféré à un modèle plus complexe, car il est généralement plus rapide à entraîner, plus facile à comprendre et à déboguer, et il peut être plus facile à déployer.
7. **Comportement de l'agent**:
   * Au-delà des mesures de performance quantitatives, la qualité du comportement de l'agent (par exemple, sa stratégie de prise de décision) peut également être un indicateur d'un bon modèle, en particulier dans des environnements où la sécurité et la prévisibilité sont critiques.

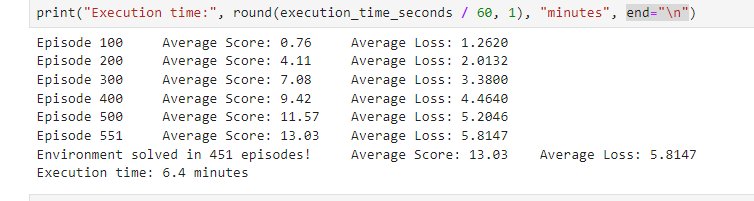
Pour votre projet, si l'objectif est d'atteindre et de maintenir un score moyen de 13.0 sur 1800 épisodes, un bon modèle sera celui qui atteint cet objectif de manière fiable et stable, sans trop de fluctuations dans la performance d'un épisode à l'autre. Il est également important de considérer la robustesse et l'efficacité de l'apprentissage, en plus de la vitesse à laquelle le score cible est atteint.

L'atteinte rapide d'un score moyen de 13.0 en 400 épisodes est une bonne performance en termes d'efficacité de l'apprentissage et de convergence. Cependant, les importants écarts types observés pour les poids des biais et autres paramètres peuvent affecter les critères suivants :

1. **Stabilité** :
   * Des écarts types élevés dans les poids et les gradients peuvent indiquer que le modèle n'est pas entièrement stable. La stabilité fait référence à la capacité du modèle à maintenir une performance cohérente sur le temps. Une grande variabilité pourrait signifier que le modèle pourrait se comporter de manière imprévisible sous certaines conditions ou dans certaines épisodes.
2. **Robustesse** :
   * Si les poids du modèle varient beaucoup, cela peut affecter sa robustesse. Un modèle robuste devrait pouvoir gérer des variations dans l'environnement ou des états initiaux sans que sa performance n'en soit trop affectée. Une grande variabilité dans les poids pourrait indiquer que le modèle pourrait ne pas bien généraliser à des variations non rencontrées pendant l'entraînement.
3. **Comportement de l'agent** :
   * Les écarts types importants dans les paramètres du modèle peuvent se traduire par des choix de stratégie inconsistants de la part de l'agent. Cela pourrait amener l'agent à prendre des décisions suboptimales dans des situations similaires, ce qui n'est pas idéal pour des comportements prévisibles et fiables.

BATCH\_SIZE = 32 # 64

TAU = 5e-4 #1e-3



LR = 1e-4 #5e-4