MARIO ET LES ZOMBIES

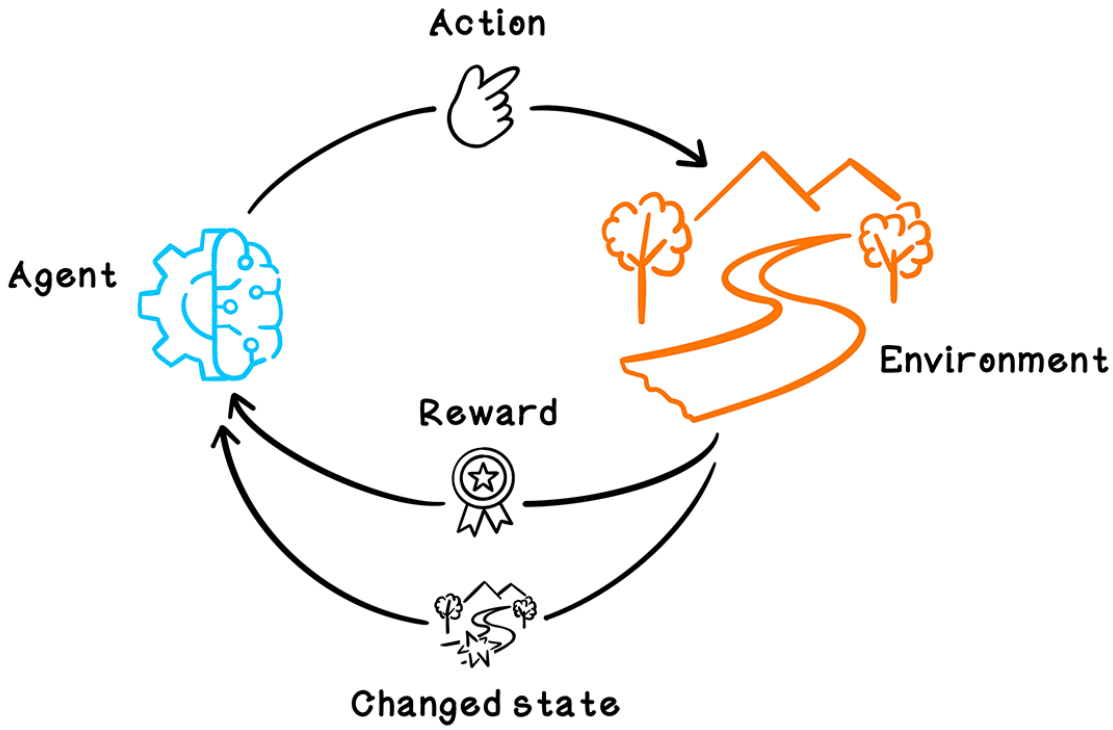
1. Analyses et choix

❖ Compréhension du jeu

Dans le jeu Mario doit esquiver des zombies qui lui tombent dessus verticalement. Il a comme possibilités aller à gauche, à droite, en haut, et en bas. Mario a donc pour chaque position quatre déplacements possibles. Tant dis que les zombies n'ont qu'un déplacement possible (de haut en bas).

❖ Choix de la méthode d'Apprentissage par Renforcement

Pour mettre en place ce jeu, nous avons choisis le **Q-learning** comme approche d'apprentissage par renforcement, car le jeu est plûtot simple, avec un nombre d'états (valeur atomique de la connaissance) limité. Il serait d'ailleurs intéressant pour nous de calculer le facteur de branchement de ce jeu (le nombre d'états atteignables depuis l'état initial), afin d'avoir une idée precise sur sa complexité.



❖ Conception de la fonction de récompense

Il nous faut trouver une fonction de récompense qui attribue des recompenses pour des actions spécifiques qui sont bénéfiques, comme esquiver un zombie, esquiver tous les zombies d'une ligne.

❖ Entraînement de l'Agent

L'agent devra explorer différentes strategies au depart, en essayant diverses actions. Une méthode comme **espsilon-greedy** sera utilisée pour équilibrer l'exploration et l'exploitation.

```
p = random()

if p < ε :
        take random action
else:
        take current-best action
```

Il nous faudra également utiliser des algorithmes d'optimisation pour améliorer ses performances au fil du temps.

❖ Évaluation et Affinage

Il nous faudra tester l'IA sur différentes parties pour voir comment elle se comporte, et en consequence améliorer son fonctionnement.

2. Aspect technique

❖ Visuel du jeu

Sachant qu'un zombie et Mario font chacun 50px de long et 50px de large pour un écran 1920*1080 pixels on a un total de 798 cases possible sur l'écran.

❖ Étapes nécessaires à l'implémentation de la solution

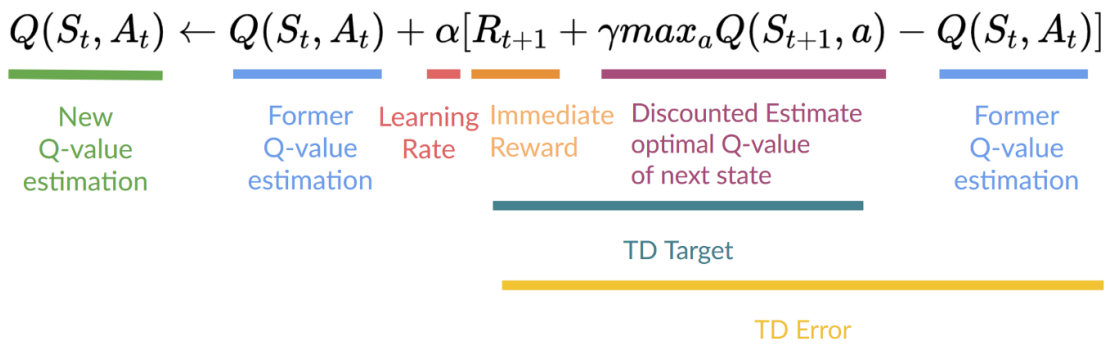
1) L'environement du jeu

Un état du jeu est une combinaison de la position de Mario et des zombies.
Les actions possibles pour Mario sont: se déplacer à gauche, à droite, en haut, et en bas.
On passe d'un état à un autre avec un déplacement de Mario, ou d'un des zombies.
On atteint la fin d'un episode si Mario est touché par un zombie, ou si il a esquivé tous les zombies d'une ligne.

2) Création et initialisation de la Q-Table

Description: La Q-table est une table de recherche où chaque entrée correspond à une paire (state, action) et stocke la valeur Q associée, qui estime la "qualité" de l'action dans cet état spécifique.
Contenu: Pour chaque combinaison d'état et d'action possible, la Q-table stocke une valeur qui représente l'espérance de la récompense future si cette action est choisie.
Initialisation: La table est initialisée avec des valeurs arbitraires (souvent 0), et elle est mise à jour au fil du temps en utilisant l'algorithme de Q-learning.

Formule:



Éléments:
α (taux d'apprentissage) : Détermine la vitesse à laquelle les valeurs de la Q-table sont mises à jour.
γ (facteur de discount) : Indique l'importance des récompenses futures par rapport aux récompenses immédiates.
St : État actuel.
At : Action actuelle.
St+1 : État suivant après avoir pris l'action *St*.
Rt+1 : Récompense obtenue en passant de *St à* St+1.

Pseudo code
Q[state][action] = Q[state][action] + alpha * (reward + gamma * max(Q[next_state]) - Q[state][action])

3) Définir la fonction de récompense

Créer une fonction qui prend en entrée l'état actuel, l'action effectuée, et l'état suivant. Elle retourne une récompense basée sur les critères suivants :

- Récompense positive pour esquiver un zombie.
- Récompense plus élevée pour esquiver tous les zombies d'une ligne.
- Pénalité pour être touché par un zombie.
- Optionnel : petite pénalité pour chaque mouvement pour encourager des actions efficaces.

```
function reward_function(state, action, next_state):
    reward = 0

    // Vérifier si Mario a esquivé un zombie
    if mario_esquive_zombie(state, action, next_state):
        reward += 10 // Récompense pour esquiver un zombie

    // Vérifier si Mario a esquivé tous les zombies d'une ligne
    if mario_esquive_tous_les_zombies_ligne(state, action, next_state):
        reward += 50 // Récompense plus élevée pour avoir esquivé toute une ligne de zombies

    // Pénaliser si Mario est touché par un zombie
    if mario_touche_par_zombie(state, action, next_state):
        reward -= 100 // Pénalité élevée pour avoir été touché par un zombie

    // Optionnel : ajouter une petite pénalité pour encourager le mouvement constant
    reward -= 1 // Pénalité mineure pour chaque mouvement, afin de limiter l'exploration excessive

    return reward
```

4) Implémenter la Politique d'Action

Mettre en œuvre la stratégie **epsilon-greedy :**

Créer une fonction pour choisir l'action à effectuer :Avec une probabilité epsilon, choisir une action aléatoire pour explorer.Avec une probabilité 1 - epsilon, choisir l'action avec la plus grande valeur Q dans la Q-table pour exploiter les connaissances actuelles.Commencer avec une valeur élevée pour favoriser l'exploration.

// Pseudo code de la fonction pour choisir une action en utilisant la stratégie epsilon-greedy

```
function choisir_action(state, Q_table, epsilon):
    // Générer un nombre aléatoire entre 0 et 1
    random_value = random()

    // Si la valeur aléatoire est inférieure à epsilon, choisir une action aléatoire (exploration)
    if random_value < epsilon:
        action = action_aleatoire()
```

```
    // Sinon, choisir l'action avec la plus grande valeur Q dans la Q-table (exploitation)
    else:
        action = argmax(Q_table[state]) // argmax retourne l'action avec la valeur Q la plus
élevée

    return action

// Fonction pour initialiser epsilon à une valeur élevée
function initialiser_epsilon(valeur_initiale):
    epsilon = valeur_initiale
    return epsilon
```

5) Mettre en place la Boucle d'Apprentissage

Initialiser les paramètres du Q-learning :

Définir le taux d'apprentissage alpha et le facteur de discount gamma.

Créer la boucle d'apprentissage principale :

Répéter pour un nombre défini d'épisodes ou jusqu'à ce que Mario atteigne un certain niveau de performance :

Initialiser l'état de départ.

Répéter jusqu'à atteindre un état terminal :

- Sélectionner une action en utilisant la politique epsilon-greedy.
- Effectuer l'action et observer la récompense et le nouvel état.
- Mettre à jour la Q-table en utilisant la formule de Q-learning.
- Passer à l'état suivant.

Réduire progressivement epsilon pour encourager l'exploitation.

```
// Exemple d'utilisation dans la boucle d'apprentissage

epsilon = initialiser_epsilon(1.0) // Commencer avec epsilon = 1.0 pour favoriser
l'exploration

for episode in range(nb_episodes):

    state = état_initial

    while not état_terminal(state):

        // Choisir une action en utilisant la stratégie epsilon-greedy

        action = choisir_action(state, Q_table, epsilon)

        // Exécuter l'action, observer la récompense et le nouvel état
```

next_state, reward = executer_action(state, action)

// Mettre à jour la Q-table avec l'équation de Q-learning

mettre_a_jour_Q_table(state, action, reward, next_state)

// Passer à l'état suivant

state = next_state

// Réduire progressivement epsilon pour moins d'exploration au fil du temps

*epsilon = max(epsilon_min, epsilon * epsilon_decay). // Réduction exponentielle d'epsilon*

6) Évaluer les Performances de Mario

Après l'apprentissage, nous devons fixer epsilon à 0 et évaluer la performance de Mario en suivant la politique apprise sur plusieurs épisodes.
Ensuite, mesurer des indicateurs comme la récompense moyenne par épisode pour évaluer l'efficacité de la stratégi.

7) Sauvegarder et Charger la Q-Table

Sauvegarder la Q-table :
Implémenter une fonctionnalité pour sauvegarder la Q-table après l'entraînement.

Charger la Q-table :
Permettre de charger une Q-table sauvegardée pour reprendre l'entraînement ou tester la politique apprise.

8) Optimisation et Réglages Finaux

Ajuster les hyperparamètres :

- Modifier les valeurs de alpha, gamma, et epsilon pour améliorer les performances de l'agent.

Refinements :

- Améliorer la fonction de récompense ou la modélisation des états si nécessaire pour des performances optimales.

9) Intégration avec l'Interface du Jeu

Tester le fonctionnement de notre algorithme avec le jeu.