La boucle de jeu (game loop)

La boucle de jeu, ou **game loop** en anglais, est l'un des concepts les plus fondamentaux dans le développement de jeux vidéo. Que vous utilisiez un moteur de jeu avancé ou que vous programmiez un jeu en 2D avec une bibliothèque comme PyGame, la boucle de jeu est essentielle pour réagir aux actions du joueur, gérer la logique du jeu et mettre à jour les éléments graphiques.

Dans cet article, nous allons explorer ce qu'est une boucle de jeu, pourquoi elle est indispensable, et comment elle est structurée. Nous verrons également des exemples concrets pour mieux comprendre son fonctionnement.

Qu'est-ce qu'une boucle de jeu ?

Une **boucle de jeu** est une structure répétitive qui tourne en continu pendant toute la durée d'exécution d'un jeu. Elle sert à :

1. **Traiter les entrées utilisateur** (clavier, souris, manette). **DONNÉES EN ENTRÉE**
2. **Mettre à jour l'état du jeu** (déplacement des personnages, gestion des collisions, etc.). **TRAITEMENTS**
3. **Rafraîchir l'affichage** pour que le joueur voit les changements à l'écran. **DONNÉES EN SORTIE**

En d'autres termes, la boucle de jeu est ce qui maintient le jeu "vivant" et réactif en temps réel. Contrairement à un programme classique qui s'exécute de haut en bas puis se termine, un jeu doit continuellement recevoir des entrées, réagir, et se redessiner jusqu'à ce que l'utilisateur décide de quitter.

Pourquoi la boucle de jeu est-elle indispensable ?

Sans boucle de jeu, il serait impossible de créer des jeux en temps réel. Voici quelques raisons pour lesquelles elle est indispensable :

- **Gestion des événements** : Le joueur peut interagir à tout moment avec le jeu via des contrôleurs comme le clavier ou la souris. La boucle de jeu permet de surveiller ces actions en temps réel et de réagir immédiatement.
- **Mise à jour de la logique du jeu** : Tout changement dans l'état du jeu, que ce soit le déplacement des personnages, le suivi des points de vie, ou le calcul des scores, est effectué au cours de chaque cycle de la boucle.
- **Rafraîchissement constant de l'écran** : L'image affichée à l'écran doit être constamment mise à jour pour refléter l'état actuel du jeu (par exemple, un personnage qui se déplace ou un score qui change).

Comment est structurée une boucle de jeu ?

La boucle de jeu suit généralement une structure commune dans la plupart des langages de programmation et moteurs de jeu. Voici les principales étapes d'une boucle de jeu typique :

1. **Initialisation du jeu**

2. **Boucle de jeu principale** :
    - **Traitement des entrées utilisateur**
    - **Mise à jour de l'état du jeu**
    - **Rendu graphique**
3. **Sortie de la boucle et nettoyage**

1. Initialisation du jeu

Avant même que la boucle de jeu ne commence, il est essentiel de configurer l'environnement. Cela comprend la création de la fenêtre de jeu, le chargement des ressources (images, sons, etc.), et l'initialisation des variables nécessaires (comme la position des personnages).

Exemple en PyGame :

© 2025 Mr. Lambrechts

```python
import pygame

# Initialiser PyGame
pygame.init()

# Créer une fenêtre
screen = pygame.display.set_mode((800, 600))

# Charger une image
player = pygame.image.load('player.png')

# Position initiale du joueur
player_x = 100
player_y = 100
```

2. Boucle de jeu principale

Une fois l'initialisation terminée, la boucle de jeu démarre et tourne jusqu'à ce que le jeu soit fermé. Voici les étapes détaillées de la boucle principale.

Traitement des entrées utilisateur

Le jeu doit surveiller les actions du joueur en temps réel. Que ce soit une touche de clavier pressée, un clic de souris ou un déplacement de manette, chaque entrée utilisateur doit être captée pour influencer l'état du jeu.

```python
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        running = False
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_LEFT:
            player_x -= 5
```

Dans cet exemple, lorsque la flèche gauche est pressée, la position du joueur se déplace vers la gauche de 5 pixels.

Mise à jour de l'état du jeu

C'est ici que toute la logique du jeu est exécutée. Cela inclut le déplacement des objets, la détection des collisions, l'évolution des scores, etc. Chaque élément du jeu doit être mis à jour à chaque tour de la boucle.

```python
# Déplacer le joueur
player_x += 1  # Exemple simple : le joueur se déplace à droite continuellement
```

Rendu graphique

Une fois que l'état du jeu est mis à jour, il est temps de **redessiner l'écran** pour que le joueur voie les changements. Dans PyGame, cela se fait en "effaçant" d'abord l'écran, puis en redessinant chaque élément dans son nouvel état.

```python
# Remplir l'écran avec une couleur
screen.fill((0, 0, 0))  # Noir

# Dessiner le joueur à sa nouvelle position
screen.blit(player, (player_x, player_y))
```

© 2025 Mr. Lambrechts

```
6
7   # Rafraîchir l'écran
8   pygame.display.flip()
```

Le rafraîchissement constant de l'écran donne l'illusion de mouvement et de réactivité. Sans cela, le joueur ne verrait jamais les modifications apportées dans la boucle de jeu.

Gestion du temps (Framerate)

La gestion du **framerate** (nombre d'images par seconde) est essentielle pour assurer une fluidité dans le jeu. Sans limitation de framerate, la boucle de jeu pourrait s'exécuter à une vitesse incontrôlable, rendant le jeu injouable. PyGame propose une méthode simple pour limiter le framerate.

```
1   # Limiter à 60 FPS
2   clock = pygame.time.Clock()
3   clock.tick(60)
```

En limitant le jeu à 60 images par seconde (FPS), on s'assure que le jeu fonctionne de manière fluide et prévisible, quelle que soit la puissance du matériel utilisé.

3. Sortie de la boucle et nettoyage

Lorsque le joueur décide de quitter le jeu (par exemple en fermant la fenêtre), la boucle de jeu doit se terminer proprement. Cela implique de sortir de la boucle, de fermer toutes les ressources ouvertes (fichiers, audio, etc.), et de quitter le programme correctement.

```
1   pygame.quit()
```

Exemple complet de boucle de jeu en PyGame

Voici un exemple complet qui illustre la structure d'une boucle de jeu simple en PyGame :

```
1   import pygame
2   import sys
3
4   # Initialisation
5   pygame.init()
6   screen = pygame.display.set_mode((800, 600))
7   pygame.display.set_caption("Boucle de Jeu")
8   player = pygame.image.load('player.png')
9   player_x = 100
10  player_y = 100
11
12  # Gestion du temps
13  clock = pygame.time.Clock()
14
15  # Boucle de jeu principale
16  running = True
17  while running:
18      # 1. Gestion des événements
19      for event in pygame.event.get():
20          if event.type == pygame.QUIT:
21              running = False
22          if event.type == pygame.KEYDOWN:
23              if event.key == pygame.K_LEFT:
24                  player_x -= 5
```

```
            if event.key == pygame.K_RIGHT:
                player_x += 5

    # 2. Mise à jour de l'état du jeu

    # 3. Rendu graphique
    screen.fill((0, 0, 0))  # Effacer l'écran
    screen.blit(player, (player_x, player_y))  # Redessiner le joueur
    pygame.display.flip()  # Mettre à jour l'écran

    # 4. Limiter le framerate
    clock.tick(60)

# Sortie propre du jeu
pygame.quit()
sys.exit()
```

Pratique

Pour faire foncitonner le code ci-dessous, télécharge un fichier png pour représenter ton personnage.

Ajoute maintenant les fonctionnalités suivantes (dans la partie `# 2. Mise à jour de l'état du jeu`):

- La gravité
- Quand le joueur disparait, il doit réapparaître en haut
- Quand le joueur réapparait, il doit réapparaître à une position horizontale aléatoire
- Fais en sorte que le joueur ne puisse pas sortir de l'écran à l'horizontale
- Affiche la fenêtre de jeu en plein écran
- Adapte le code pour qu'il fontionne quelle que soit la taille de la fenêtre/écran.

Les bonnes pratiques dans la gestion de la boucle de jeu

Bien que la structure de la boucle de jeu puisse sembler simple, il est important de suivre quelques **bonnes pratiques** pour assurer la performance et la lisibilité du code.

- **Ne surchargez pas la boucle de jeu** : Placez les calculs complexes (comme l'IA ou les algorithmes de physique) dans des fonctions dédiées pour garder la boucle de jeu claire et organisée.
- **Utilisez un framerate constant** : Limiter la vitesse de la boucle à un framerate constant assure que le jeu fonctionne de manière fluide sur différents types de matériel.
- **Séparez la logique du jeu du rendu graphique** : Ne mélangez pas la mise à jour de l'état du jeu avec le rendu. Il est plus facile de déboguer et de maintenir le code si chaque partie a son propre rôle clairement défini.

Limites et optimisation de la boucle de jeu

Dans des projets plus complexes, la boucle de jeu peut devenir gourmande en ressources si elle n'est pas bien optimisée. Voici quelques conseils pour améliorer les performances :

- **Minimisez les appels de rendu** : Le rendu des graphiques est souvent l'opération la plus coûteuse. Ne redessinez que ce qui a changé.

- **Utilisez des structures de données efficaces** : Pour des jeux avec beaucoup d'objets, comme des ennemis ou des projectiles, veillez à utiliser des structures de données adaptées (comme des listes ou des groupes de sprites) pour les manipuler.

- **Optimisez la gestion des collisions** : Les calculs de collision peuvent devenir complexes lorsque plusieurs objets interagissent. L'utilisation d'algorithmes de partitionnement de l'espace (comme les quadtrees) peut améliorer les performances.

Conclusion

La boucle de jeu est un élément fondamental dans tout jeu vidéo, garantissant une exécution continue et fluide des actions du jeu. Comprendre son fonctionnement est essentiel pour tout développeur de jeux, débutant ou expérimenté. En maîtrisant la boucle de jeu, vous aurez une base solide pour développer des jeux réactifs, immersifs, et performants.

© 2025 Mr. Lambrechts