République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumédiène

Faculté d'Informatique

Master Systèmes Informatiques intelligents

Module : Vision

Rapport de projet Vision : Brick Racing

Réalisé par :
FERKOUS Sarah, 191931043867
HOUACINE Maya,191931047492
KHEMISSI Maroua, 191935007943
OUCHAR Manel,191931047502

Année universitaire : 2023 / 2024

Table des matières

Table des figures

Liste des tableaux

1.1 Introduction

Dans ce projet, nous voulons mettre en évidence le développement des filtres classiques avec des fonctions de base pour mieux comprendre le comportement de ces filtres. Ensuite, nous allons développer le jeu sous le titre "Brick Racing Game" qui se base sur la détection d'objets.

1.2 Réalisation des filtres classiques

1.2.1 Binarisation et Seuillage :

Pour simplifier ou segmenter une image en deux classes distinctes : blanc et noir (ou 0 et 1), Ces techniques sont couramment utilisées dans des applications telles que la reconnaissance d'objets, la détection de contours...
Son rôle est le suivante : Simplification de l'information, Segmentation, Prétraitement pour d'autres algorithmes, Détection de contours.

Filtre	Description	Rôle
Filtres de Lissage		
Moyen	Vise à lisser une image en remplaçant la valeur de chaque pixel par la moyenne des valeurs de ses pixels voisins. Il utilise une fenêtre de noyau pour calculer cette moyenne, ce qui atténue les variations d'intensité locales.	Lissage et réduction du bruit, Effet de flou
Médian	Remplace la valeur du pixel par la médiane des valeurs de ses pixels voisins dans une fenêtre de noyau.	Réduction du bruit, Préservation des Contours
Gaussien	Basé sur la convolution avec une fonction gaussienne, qui attribue des poids plus élevés aux pixels du centre du noyau et des poids décroissants aux pixels périphériques.	Flou Artistique, Réduction du Bruit, Prétraitement pour la Détection de Contours
Filtres de détection de contours		
Laplacien	Il est basé sur le concept de la deuxième dérivée, mettant en évidence les zones où les variations d'intensité sont importantes.	Accentuation des contours, Détection de Structures, Amélioration de la Perception Visuelle
Gradient	Identification des variations d'intensité et de la direction du changement	Détection des contours, Détection des Bords
Filtres morphologiques		
Erosion	En remplaçant chaque pixel par la valeur minimale des pixels couverts par l'élément structurant.	Réduction de la taille des objets, Séparation d'objets connectés, Élimination du bruit
Dilatation	correspond à un OU logique sur une image binaire, Elle permet d'élargir une image. On centrera le kernel sur chaque pixel et Si le pixel est égal à 1 on fera une empreinte du kernel sur l'image (centré sur le pixel).	Expansion des objets, Remplissage des trous, Connectivité des objets
Fermeture	c'est la composition de la dilatation par un gabarit suivie de l'érosion par ce même gabarit (un ET logique sur une image binaire)	Séparation d'objets connectés, Réduction du bruit
Ouverture	c'est la composition de l'érosion par un gabarit suivie de la dilatation par ce même gabarit.	Détection des contours, Détection des Bords

TABLE 1.1 – Tableau Descriptif Des Filtres.

1.3 Implémentation de filtres personnalisés

Filtre	Description	Rôle
Prewitt	Il est basé sur le concept de la convolution, où un petit noyau (matrice) est appliqué à chaque pixel de l'image pour calculer une nouvelle valeur de pixel, il se compose de deux noyaux distincts(intensité horizontales et verticales)	Détection des contours, Identification de la direction du contour
Sobel	l est basé sur la convolution d'une image avec des noyaux spécifiques conçus pour détecter les variations d'intensité le long des axes horizontal et vertical	Détection des contours, Accentuation des contours, Orientation des contours

TABLE 1.2 – Tableau Descriptif Des Filtres Proposes.

On a choisi les filtres **Prewitt** et **Sobel**, car nous avons besoin de filtres de détection de contours dans la deuxième partie, et ces deux filtres répondent à ce besoin. Nous avons effectué des tests avec ces filtres et avons choisi celui qui est le plus adapté pour notre jeu.
Prewitt est plus rapide, tandis que Sobel offre une meilleure détection.

1.4 Détection d'objet par couleur ameliorée

Le principe de notre algorithme se base sur le concept de "Less is More", c'est à dire n'utiliser que les fonctions qui contribuent considérablement à la qualité du résultat sans pour autant être coûteux en termes de temps. Atteignant ainsi un compromis entre la qualité et l'aspect temps réel de notre fonction. Les améliorations incluent :

- Le redimensionnement de l'image en entrée : technique simple et efficace pour réduire le temps d'exécution en réduisant la taille de l'image en entrée .
- L'élimination de dilatation dans la phase d'ouverture : l'ouverture a pour but d'éliminer le bruit ce qui est fait essentiellement par l'érosion, la dilatation permet de reconstituer les bords effacés par l'érosion. Mais nous avons pas besoin des bords pour calculer le centre de l'objet (centre de object avec bord == centre de object sans bord) .
- Optimisation de l'érosion et le calcul du centre.

Algorithm 1 Object Detection Algorithm

tant que *Vrai* **faire-**

Lire($frame$)
Redimensionner($frame$)
$frame \leftarrow$ Retourner verticalement($frame$)
$frame \leftarrow$ BGRtoHSV($frame$)
$masque \leftarrow$ inrange($frame$, low, $high$)
$masque \leftarrow$ erosion($masque$)
$centre \leftarrow$ center($masque$)
Dessiner cercle à la position $centre$
Afficher $frame$ et $masque$

=0

1.5 Manteau d'Invisibilité et Écran Vert

Pour ce qui est des algorithmes cape d'invisibilité et fond vert, étant donné que nous remplacions les pixels blancs de notre masque par sa partie équivalente (même position) du background enregistré auparavant (pour la cape d'invisibilité) et de l'image choisis comme fond vert dans la 2ème fonction, nous ne pouvons pas échapper à la phase de dilatation cette fois. Et cela pour reconstituer les bords qui ont eu lieu lors de l'érosion car on en a besoin cette fois.

Dans le code source l'étape de dilatation et de remplacement se fait en même temps afin d'améliorer le temps d'exécution.

Algorithm 2 Invisibilité

$background \leftarrow$ Capturer une frame de vidéo
$background \leftarrow$ Retourner verticalement($background$)
$background \leftarrow$ Redimensionner($background$)
tant que *True* **faire**
Lire($frame$)
$frame \leftarrow$ Retourner verticalement($frame$)
$frame \leftarrow$ Redimensionner($frame$)
$masque \leftarrow$ Object color detection($frame$)
Dilatation($masque$)
pour y *de* 0 *à* $longueur(frame)$ **faire**
pour x *de* 0 *à* $largeur(frame)$ **faire**
si $masque[y, x] = 255$ **alors**
$frame[y, x] = background[y, x]$
fin si
fin pour
fin pour
fin tq
=0

Algorithm 3 Fond Vert

```
Lire(image)
image ← Retourner verticalement(image)
image ← Redimensionner(image)
    tant que Vrai faire
| Lire(frame)
| frame ← Retourner verticalement(frame)
| frame ← Redimensionner(frame)
| masque ← Object color detection(frame)
| Dilatation(masque)
|     pour y de 0 à longueur(frame) faire
|     |     pour x de 0 à largeur(frame) faire
|     |     |     si masque[y, x] = 255 alors
|     |     |   frame[y, x] = image[y, x]
|     |     |     fin si
|     |     |
|     |     fin pour
|     |
|     fin pour
|
|   fin tq
```

1.6 Algorithme du jeu

Algorithm 4 Brick Racing Game

Initialisation :
Initialiser la fenêtre graphique, la largeur, la hauteur, la voiture du joueur, les ennemis, etc.
Déplacement du joueur :
Le joueur peut déplacer la voiture avec les touches 'a' (gauche) et 'd' (droite).
Déplacement des ennemis :
Les ennemis descendent verticalement à une vitesse donnée.
Collision et Score :
Vérifier les collisions entre la voiture du joueur et les ennemis. Réinitialiser en cas de collision.
Marquer des points si un ennemi atteint le bas de l'écran sans collision.
Augmentation de la Difficulté :
Augmenter la vitesse des ennemis chaque fois que le score est multiple de 10. Diminuer le nombre d'ennemis.
Affichage :
Afficher en temps réel le score, la vitesse et l'état du jeu.
Contrôle du Jeu :
Utiliser la barre d'espace pour démarrer ou redémarrer le jeu. Utiliser 'q' pour quitter.
Vision (Optionnel) :
Activer le mode vision ('v') pour le contrôle de la voiture faisant des appels au fonctions implimenter deja ObjectDetection().

1.7 Explication du code source

Nous avons créé des classes distinctes pour modéliser les différents acteurs du jeu, tels que le joueur, les ennemis et le filtre de Kalman pour la prediction d'objets. A noter d'abord que nos amelioration majeurs face au jeux sont : l'implementation de 3 modes, 1 avec touches , 1 avec detection de mouvement horizontale et enfin detection de mouvement horizontale et verticale.Aussi nous avons implementer l'algorithme de Kalman permettant la prediction de position de notre object.Enfin, la detection de colision avec les obstacles est precise et le jeu se passe en temp reel sans decalage du au temps d'execution.

1.7.1 Classes :

1.7.1.1 Player (Joueur) :

La classe Player modélise le personnage du joueur dans le jeu. Elle gère les déplacements horizontaux du joueur, son affichage à l'écran, ainsi que ses caractéristiques comme sa position et sa taille. Le tableau ci-dessous représente son contenu :

Méthodes	**Description**
move_left()	Déplace le joueur vers la gauche.
move_right(width)	Déplace le joueur vers la droite en tenant compte de la largeur de l'écran.
display(img)	Affiche le joueur à la position actuelle sur l'image.

TABLE 1.3 – Tableau descriptif des attributs et des méthodes de la classe Player (Joueur).

1.7.1.2 Enemy (Ennemi) :

La classe Enemy représente les ennemis mobiles dans le jeu. Chaque instance de cette classe est responsable de sa propre logique de déplacement, de sa collision avec le joueur, et de son affichage.

Méthodes	**Description**
collision(obj)	Vérifie la collision avec un objet donné.
out_of_bounds(height)	Vérifie si l'ennemi est hors de l'écran.
display(img)	Affiche l'ennemi à sa position actuelle sur l'image.

TABLE 1.4 – Tableau descriptif des attributs et des méthodes de la classe Enemy (Ennemi).

1.7.1.3 BorderEnemy (Ennemi de bord)

La classe BorderEnemy décrit les ennemis stationnaires positionnés sur les bords de l'écran. Comme la classe Enemy, elle gère son propre déplacement, sa collision avec le joueur et son affichage.

Méthodes	Description
collision(obj)	Vérifie la collision avec un objet donné.
out_of_bounds(height)	Vérifie si l'ennemi de bord est hors de l'écran.
display(img)	Affiche l'ennemi de bord à sa position actuelle sur l'image.

TABLE 1.5 – Tableau descriptif des attributs et des méthodes de la classe BorderEnemy.

1.7.1.4 KalmanFilter :

La classe KalmanFilter est utilisée pour appliquer un filtre de Kalman à des données de suivi, telles que la position d'un objet à l'écran. Elle prédit la position suivante de l'objet en utilisant des observations antérieures.

Méthodes	Description
predict()	Prédit la position suivante.
update(z)	Met à jour le filtre avec une nouvelle observation.

TABLE 1.6 – Tableau descriptif des attributs et des méthodes de la classe KalmanFilter.

1.7.2 Fonctions Utilitaires

En plus des classes principales, des fonctions utilitaires sont fournies pour effectuer les opérations de traitement d'image telle que :

Fonction	Description
erode(mask, kernel)	Applique l'opération érosion à la masque avec le noyau donné
inRange(img, lo, hi)	Crée un masque binaire en fonction de la plage de couleur spécifiée
center(img)	Trouve le centre d'une région blanche dans le masque
detect_inrange(image)	Applique le traitement inRange au flux vidéo pour détecter des objets
resize(img)	Redimensionne l'image en réduisant la taille

TABLE 1.7 – Tableau descriptif des fonctions utilitaires.

1.8 Conclusion Générale

Nous avons pu réaliser le travail demandé avec des améliorations très importantes. Notre jeu s'exécute avec trois modes, incluant un déplacement horizontal et verticale tout en mettant en place une prédiction de déplacement, en cas d'obstraction entre l'objet vert et la camera.

Annexe

1.9 Manuel du jeu

Ce manuel vous guidera à travers les différentes façons de jouer au jeu, que ce soit avec les touches du clavier ou en utilisant la vision par ordinateur.
Avant d'essayer d'exécuter notre projet, veuillez vous assurer d'avoir téléchargé les éléments suivants :

1. Python
2. cv2
3. math
4. random
5. time
6. subprocess
7. customtkinter
8. tkinter
9. PIL
10. threading
11. couleur choisis : vert

L'interface du jeu est comme suit :

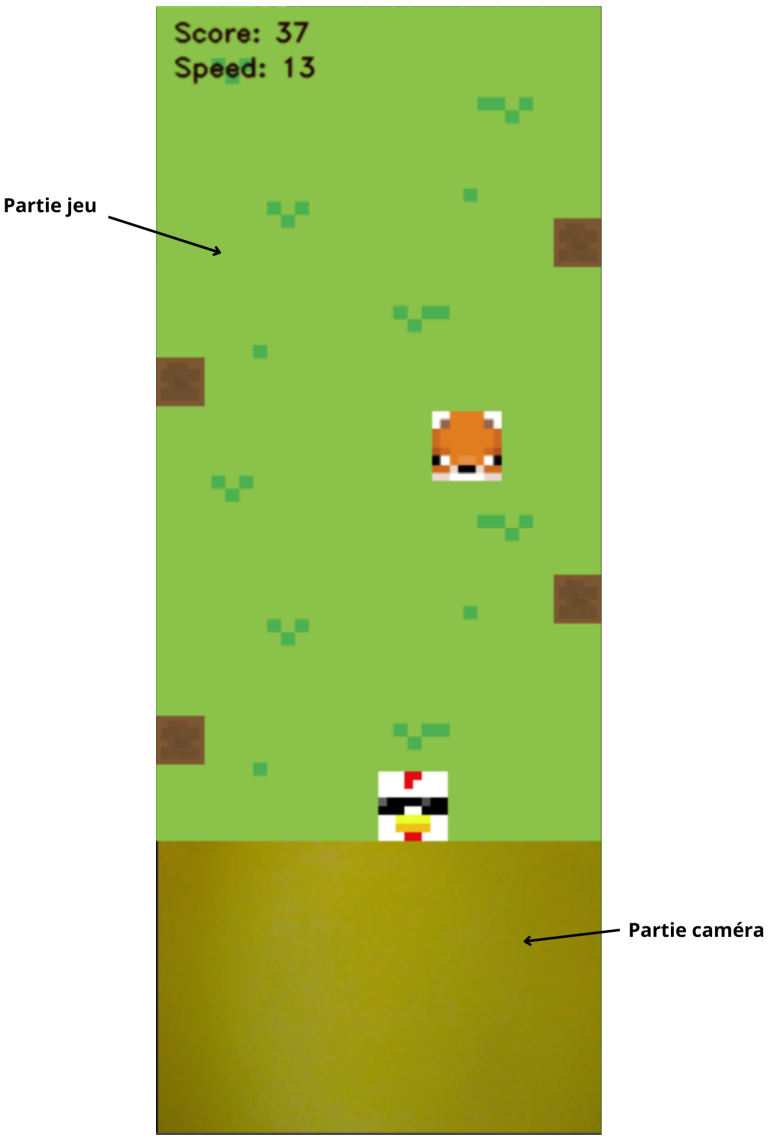

FIGURE 1.1 – Interface du jeu.

1.9.1 Instructions de base

1.9.1.1 Déplacement du joueur

— Utilisez la touche "**Q**" pour déplacer la poule vers la gauche.
— Utilisez les touches "**D**" pour déplacer la poule vers la droite.

1.9.1.2 Personnages

— **Joueur** : Une poule.
— **Ennemis** : Des renards rusés.
— **Obstacles de bordures** : Des troncs d'arbres.

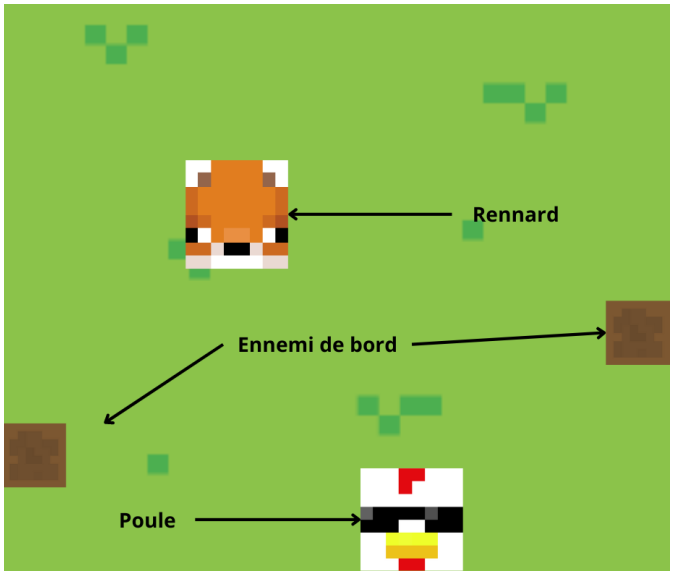

FIGURE 1.2 – Personnages du jeu sur l'interface.

1.9.1.3 Objectif du jeu

— Évitez les renards et les troncs d'arbre.
— Collectez des points en restant en vie le plus longtemps possible.

1.9.2 Mode Clavier

1. **Démarrage du Jeu** : Appuyez sur la touche "**ESPACE**" pour commencer ou redémarrer le jeu.

FIGURE 1.3 – Début du Début du jeu jeu.

2. **Pendant le Jeu** :

— Évitez les renards et les troncs d'arbre qui descendent vers le bas de l'écran.
— Gagnez des points en évitant les obstacles.
— La vitesse et le nombre de renards augmente à chaque tranche de 10 points.

FIGURE 1.4 – Le joueur (la poule) évite le renard et les troncs d'arbre en cliquant sur "**Q**" .

3. **Fin du jeu** :
   — Si vous entrez en collision avec un renard ou un tronc d'arbre, le jeu se termine.
   — Le score final est affiché.

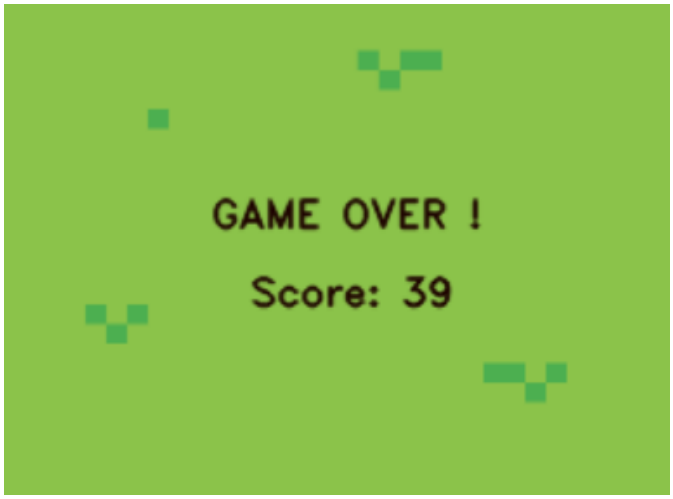

FIGURE 1.5 – Fin du jeu avec affichage du score final.

1.9.3 Mode Vision

1. **Démarrage du Jeu** : Appuyez sur la touche "**ESPACE**" pour commencer ou redémarrer le jeu.
2. **Pendant le Jeu** :
   — Utilisez la vision par ordinateur pour contrôler la poule.
   — La poule suivra le mouvement de la couleur détecté par la caméra.
   — Assurez-vous que l'objet coloré est dans le champ de vision de la caméra.

FIGURE 1.6 – Mode de jeu vision avec le suivi horizontale d une couleur verte sur une feuille blanche.

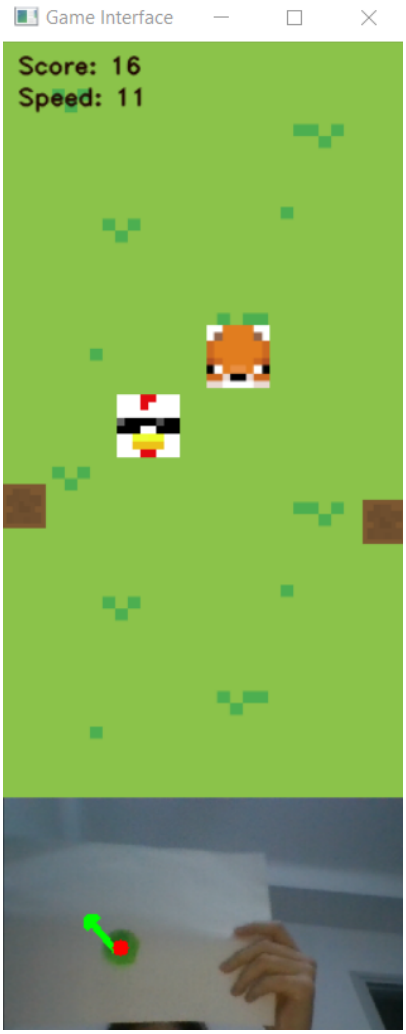

FIGURE 1.7 – Mode de jeu vision avec le suivi honrizontale et verticale d une couleur verte sur une feuille blanche.

3. **Fin du jeu** :
   — Si la poule sort du champ de vision ou entre en collision avec un renard ou un tronc d'arbre, le jeu se termine.
   — Le score final est affiché.

1.9.4 Raccourcis clavier

— Appuyez sur "**E**" pour quitter le jeu à tout moment.