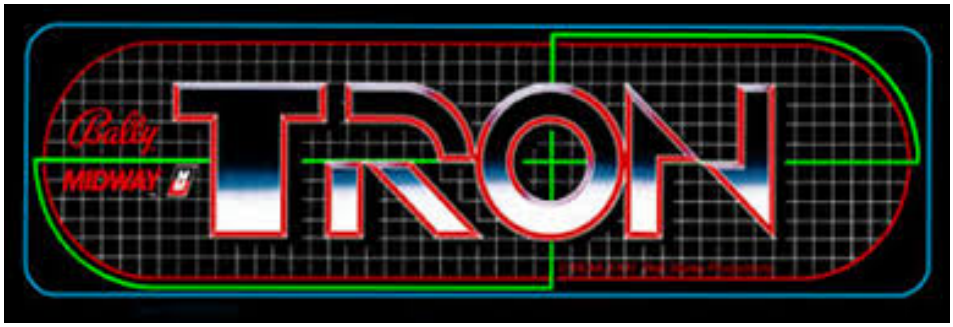
UNIVERSITÉ CAEN NORMANDIE

Université de Caen Normandie

Département Informatique de Caen



Réalisé par :

Medjdoub Karim

Rahmoun Anes

Gerouani Melissa

Moudir Mohamed

Encadré par :

Spaniol Marc

François Rioult

Jeu de Tron - Année Universitaire 2024/2025

Indice

1 Introduction 3

2 Objectifs du projet 4
2.1 Problématique 4
2.2 Étapes principales 4
2.3 Défis à affronter 5
2.4 Métriques de performance 7

3 Description des algorithmes, données et de l'architecture 9
3.1 Stratégies de jeu et IA 9
3.2 Algorithmes utilisés dans ce projet 10
3.3 Paquets du projet 10
3.4 Description des paquetages non standards utilisés 12
3.5 Description des paquetages non standards utilisés 12
3.6 Évaluation des performances 13

4 Fonctionnalités implémentées 14
4.1 Déplacement des motos 14
4.2 Stratégies d'IA 14
4.3 Affichage du jeu 15
4.4 Gestion des collisions 15
4.5 Interface de configuration 15
4.6 Fin de partie et résultats 15

5 Organisation du projet 16
5.1 Répartition des tâches 16
5.2 Collaboration et communication 16
5.3 Répartition des responsabilités supplémentaires 16

6 Expérimentations 17
6.1 Tests d'optimisation 17
6.2 Analyse des résultats 17
6.3 Pistes d'amélioration 17

7 Conclusion **18**

1 Introduction

Dans le cadre de ce projet, nous avons choisi de recréer le célèbre jeu de Tron, un jeu vidéo emblématique inspiré du film de science-fiction sorti en 1982. Ce jeu, qui se déroule dans un univers virtuel, met en scène des motos lumineuses (light cycles) qui se déplacent à grande vitesse. L'objectif principal est de faire en sorte que l'adversaire entre en collision avec la traînée laissée par votre moto ou avec les murs du terrain de jeu, tout en évitant de percuter votre propre ligne. Ce projet a pour but de revisiter ce classique en apportant des éléments modernes, tout en respectant l'esprit du jeu original, en termes de gameplay et de design. À travers ce projet, nous souhaitons non seulement recréer une expérience de jeu nostalgique mais aussi explorer des concepts de programmation, de gestion d'interactivité et d'algorithmes tout en s'inspirant d'un jeu qui a marqué l'histoire du jeu vidéo.

Figura 1: Jeu de Tron

2 Objectifs du projet

2.1 Problématique

Le projet consiste à développer une intelligence artificielle capable de jouer à Tron sans interaction humaine, en prenant des décisions stratégiques en temps réel pour éviter les collisions et maximiser sa survie. L'IA doit être compétitive et réactive dans un environnement dynamique. Le projet explore l'implémentation de l'algorithme MAXN ou Paranoid pour la prise de décision multi-agent et la création d'équipes de joueurs, inspirée de l'approche SOS, afin d'optimiser la stratégie collective. Enfin, une visualisation du jeu est intégrée pour analyser les performances de l'IA.

2.2 Étapes principales

1. Création du jeu de Tron :
    - Définir l'espace de jeu avec JavaFX (création d'une grille ou d'un plateau de jeu).
    - Implémenter la logique des mouvements des joueurs et la détection des collisions.
    - Mettre en place les règles de base du jeu (mouvement des joueurs, conditions de victoire).
2. Visualisation du jeu avec JavaFX :
    - Utiliser JavaFX pour créer une interface graphique pour afficher le jeu.
    - Dessiner les joueurs, leurs traces et mettre à jour l'affichage en temps réel.
    - Gérer l'animation des déplacements et les événements (gestion des fenêtres et des entrées utilisateur).
3. Implémentation de l'IA :

- Intégrer un algorithme de prise de décision (MAXN ou Paranoid) pour les joueurs IA.
- Construire un arbre de décision pour évaluer les actions possibles.
- Définir une fonction d'évaluation pour optimiser les choix de l'IA (espace libre, proximité des murs, etc.).

4. Extension pour gérer des équipes :
   - Ajouter un mécanisme de gestion des équipes dans le jeu.
   - Adapter l'algorithme pour permettre une stratégie collective entre les membres de l'équipe.
5. Tests et analyse des performances :
   - Lancer des simulations de parties pour observer les performances de l'IA.
   - Analyser l'efficacité de l'IA et ajuster les algorithmes pour améliorer les résultats.
6. Documentation et présentation :
   - Documenter le code et l'implémentation JavaFX.
   - Créer un rapport détaillant les choix techniques, les résultats et les conclusions.

2.3 Défis à affronter

Le développement d'une IA pour jouer à Tron implique plusieurs défis techniques et stratégiques, qui nécessitent de surmonter des obstacles pour assurer des performances optimales. Voici quelques-uns des défis majeurs :

- Gestion de la réactivité en temps réel : L'IA doit être capable de réagir rapidement à des situations dynamiques tout en prenant des décisions stratégiques sur la durée. Les algorithmes de prise

de décision, comme MAXN et Paranoid, doivent être suffisamment efficaces pour permettre à l'IA de répondre en temps réel, ce qui peut être difficile dans un environnement à grande échelle avec de multiples joueurs.

- Prévision des mouvements des adversaires : L'IA doit anticiper les actions des autres serpents, ce qui est un défi majeur dans un jeu avec plusieurs agents. Le calcul de leurs déplacements futurs tout en maximisant la propre survie du serpent demande une bonne compréhension des dynamiques du jeu et une gestion efficace de l'arbre de décision.

- Optimisation de la stratégie collective : Lorsque l'IA gère des équipes de joueurs, il devient crucial de coordonner les actions des différents serpents afin de créer une stratégie collective cohérente. Cela implique de gérer les interactions entre les joueurs et de déterminer des tactiques qui augmentent les chances de succès pour toute l'équipe.

- Équilibre entre exploration et exploitation : L'IA doit trouver un équilibre entre l'exploration de nouvelles stratégies et l'exploitation de celles qui semblent les plus efficaces. Trop d'exploration peut entraîner une inefficacité, tandis qu'une trop grande exploitation peut rendre l'IA prévisible et vulnérable.

- Gestion des ressources du terrain : Les serpents doivent naviguer dans un environnement avec des obstacles (murs et autres serpents). La gestion de l'espace libre et des positions optimales pour éviter les collisions tout en maximisant la survie est un défi qui nécessite une évaluation constante de la situation du terrain.

- Performances des algorithmes : Les algorithmes de prise de décision, notamment ceux basés sur la recherche dans les arbres (comme MAXN), peuvent devenir gourmands en temps de calcul à mesure que la complexité du jeu augmente, surtout si le nombre de joueurs

ou la taille du terrain augmente. Il est donc essentiel de rendre ces algorithmes suffisamment efficaces pour garantir une expérience fluide.

- Adaptation aux changements du jeu : Tron est un jeu à évolution rapide, et l'IA doit être capable de s'adapter à de nouvelles situations, par exemple si un adversaire adopte une nouvelle stratégie ou si l'environnement de jeu change de manière inattendue.

2.4 Métriques de performance

Les performances des équipes ont été évaluées à l'aide de plusieurs critères. À la fin de chaque jeu, un fichier 'metrics.txt' est généré, dans lequel les informations suivantes sont consignées :

- Total des mouvements : Le nombre total de déplacements réalisés par les équipes pendant la partie.
- Temps d'exécution des équipes : Le temps pris pour effectuer chaque mouvement, mesuré en nanosecondes.
- Nombre de mouvements par équipe : Le nombre total de déplacements effectués par chaque équipe.

Exemple de sortie des métriques à la fin d'une partie, tel qu'il apparaît dans le fichier 'metrics.txt' :

```
Total Moves: 39
Team Execution Times:
Team 1 (Algorithm: MAXN): 35876370 ns
Team 2 (Algorithm: MAXN): 8451822 ns
Team Move Counts:
Team 1: 20 moves
Team 2: 19 moves
```

Ces métriques permettent d'analyser les performances des algorithmes en termes de stratégie et de temps d'exécution. Elles sont particulièrement utiles pour observer l'efficacité des décisions prises par l'IA

et le temps nécessaire pour chaque action. Les informations du fichier ‘metrics.txt‘ sont ensuite utilisées pour ajuster les paramètres de l’IA et optimiser les algorithmes pour les parties suivantes.

3 Description des algorithmes, données et de l'architecture

3.1 Stratégies de jeu et IA

Dans le jeu Tron, l'intelligence artificielle (IA) repose sur des stratégies qui permettent à chaque serpent d'agir de manière autonome et de prendre des décisions optimales pour maximiser ses chances de survie et d'expansion. Les principales stratégies utilisées dans ce projet sont basées sur des algorithmes classiques de la théorie des jeux, qui prennent en compte l'anticipation des mouvements des autres joueurs et l'évaluation dynamique de l'état du jeu.

- Maxn : Une généralisation de l'algorithme Minimax, qui est couramment utilisé dans les jeux à deux joueurs. Maxn est conçu pour des jeux à plusieurs joueurs, et chaque joueur cherche à maximiser ses chances de succès en anticipant les actions de tous les autres joueurs. L'algorithme évalue plusieurs coups possibles et choisit celui qui maximise les chances de réussite pour un joueur, tout en tenant compte des décisions des autres joueurs.

- Paranoid : L'algorithme Paranoid repose sur une approche plus défensive. Plutôt que d'essayer de maximiser ses gains, il se concentre sur la minimisation des risques en anticipant les mouvements des adversaires. Cela permet de prendre des décisions prudentes, augmentant ainsi les chances de survie du serpent tout en limitant les possibilités de perte.

- SOS (Simultaneous Optimization Search) : Contrairement à Maxn et Paranoid qui se concentrent sur l'optimisation des actions d'un joueur unique, SOS prend en compte les mouvements de tous les joueurs simultanément. Il s'agit d'une approche plus globale, visant à optimiser les décisions des serpents de manière coordonnée tout en réagissant aux actions des autres joueurs. Cette stratégie

repose sur un équilibre entre la maximisation des gains pour chaque joueur et la gestion des risques.

3.2 Algorithmes utilisés dans ce projet

Les trois algorithmes implémentés dans ce projet ont des objectifs spécifiques pour l'intelligence artificielle du jeu Tron. Ils sont chacun conçus pour maximiser les chances de survie et la performance globale dans le jeu :

- Maxn : Ce système permet à chaque joueur d'évaluer plusieurs coups possibles tout en prenant en compte les actions des autres joueurs. Par exemple, si un joueur a la possibilité de bloquer un autre serpent ou de s'agrandir sans risquer une collision, il choisira l'option qui maximise ses chances de succès à long terme.
- Paranoid : L'algorithme met l'accent sur la défense et la gestion des risques. Un serpent utilisant cette stratégie tentera de s'éloigner des zones dangereuses et de prévoir les déplacements des adversaires afin de survivre le plus longtemps possible.
- SOS (Simultaneous Optimization Search) : Cette approche repose sur une gestion simultanée des mouvements de tous les serpents. L'objectif est d'optimiser les stratégies de tous les joueurs en tenant compte de leurs positions actuelles sur le terrain et des obstacles (murs et autres serpents). Le but est de maximiser les chances de survie tout en minimisant les risques pour chaque joueur.

3.3 Paquets du projet

Les différents paquets de ce projet sont organisés pour gérer les différentes fonctionnalités du jeu :

- View : Ce paquet contient les classes responsables de l'affichage du jeu et de l'interface graphique.
  - `GameCanvas`

  - `GameScene`

- com.tronai.model : Ce paquet définit les modèles de données utilisés dans le jeu, y compris les joueurs, les équipes, et les cellules du plateau.
  - entities :
    * `Bot`
    * `Bike`
  - platform :
    * `Direction`
    * `Platform`
    * `Position`
    * `Team`

- controller : Ce paquet contient la logique de contrôle du jeu, y compris le mouvement des joueurs et la gestion de l'état du jeu.
  - `Controller`
  - `GameInitializer`

- Algo : Ce paquet contient les heuristiques utilisées pour les algorithmes d'intelligence artificielle.
  - `Maxn`
  - `Minmax`
  - `Paranoid`

- util : Ce paquet contient des classes utilitaires, comme les directions et les types d'algorithmes utilisés par les équipes.
  - config :
    * `ConfigRead`
  - mvc :
    * `AbstractObservable`

    * `Observable`
    * `Observer`
  - strategie :
    * `BotStrategie`
    * `MaxnStrategie`
    * `ParanoidStrategie`

3.4 Description des paquetages non standards utilisés

Le projet utilise plusieurs bibliothèques non standards pour la gestion de l'interface graphique et des optimisations, notamment JavaFX et les bibliothèques pour l'algorithme Maxn et SOS.

3.5 Description des paquetages non standards utilisés

Le projet utilise plusieurs bibliothèques non standards pour la gestion de l'interface graphique et des optimisations, notamment JavaFX et les bibliothèques pour l'algorithme Maxn et SOS.

3.6 Évaluation des performances

Les performances des algorithmes sont mesurées sur plusieurs critères :

- Nombre de mouvements : Cela permet de déterminer l'efficacité des algorithmes en fonction de la longueur des trajets parcourus avant qu'un serpent ne soit éliminé. Des stratégies plus efficaces devraient conduire à moins de mouvements inutiles et à une gestion plus précise du terrain.
- Temps d'exécution des algorithmes : Le temps qu'un algorithme met pour prendre une décision est crucial pour le jeu en temps réel. Une IA qui met trop de temps pour calculer ses coups pourrait entraîner une expérience de jeu frustrante.

L'évaluation permet ainsi de comparer les différentes stratégies pour identifier celle qui offre le meilleur compromis entre performance et réactivité.

4 Fonctionnalités implémentées

4.1 Déplacement des motos

Les motos sont contrôlées par l'algorithme d'IA sélectionné, qui détermine leur trajectoire en temps réel. Chaque moto évalue constamment son environnement et ajuste sa direction pour éviter les collisions tout en maximisant l'occupation de l'espace libre. La prise de décision est basée sur les informations actuelles du terrain et des adversaires.

4.2 Stratégies d'IA

L'IA utilise plusieurs stratégies pour interagir avec le terrain et les autres joueurs. Ces stratégies sont dynamiques et prennent en compte les changements du jeu à chaque instant. Elles permettent aux motos de répondre aux situations en temps réel.

- Maxn : L'algorithme Maxn cherche à maximiser les chances de survie et d'expansion de la moto, tout en tenant compte des actions prévisibles des autres joueurs. Il analyse les décisions possibles de tous les joueurs et choisit celle qui offre les meilleures chances de succès.
- Paranoid : Cette stratégie met l'accent sur la défense. La moto utilisant Paranoid évite les zones risquées en anticipant les mouvements des autres motos, en cherchant à minimiser les dangers immédiats, tout en maximisant ses chances de survie.
- SOS (Simultaneous Optimization Search) : SOS coordonne les actions des motos en prenant en compte les mouvements des adversaires, cherchant à optimiser les stratégies collectives. Elle permet une gestion simultanée de plusieurs motos pour tirer parti de l'intéraction entre elles.

4.3 Affichage du jeu

Le jeu est rendu visuellement grâce à JavaFX, offrant une interface graphique fluide et réactive. Le terrain de jeu est mis à jour à chaque étape du jeu, affichant les déplacements des motos et leurs traces sur le plateau. Les animations des déplacements et des événements sont gérées en temps réel, créant une expérience visuelle immersive pour l'utilisateur.

4.4 Gestion des collisions

Les collisions avec les murs, les autres motos ou soi-même entraînent l'élimination immédiate de la moto concernée. Ce mécanisme de gestion des collisions permet de garantir une dynamique de jeu rapide et fluide, et de maintenir une expérience de jeu compétitive. La partie se termine lorsqu'un seul joueur (moto) reste en vie.

4.5 Interface de configuration

L'interface de configuration permet aux joueurs de personnaliser les paramètres du jeu avant le lancement. Les paramètres, tels que la taille du terrain, le nombre de joueurs (humains et IA), et les stratégies utilisées pour chaque moto, peuvent être définis à travers un fichier de propriétés. Cela permet de lancer une partie avec des configurations variées et d'adapter les règles à différentes préférences.

4.6 Fin de partie et résultats

La partie se termine lorsqu'une seule moto survit. Un message de victoire s'affiche à la fin de chaque partie, indiquant la moto gagnante. Après cela, les joueurs peuvent choisir de recommencer une nouvelle partie avec les mêmes ou de nouveaux paramètres. Un système de scoring ou de classement peut être intégré pour ajouter un aspect compétitif à long terme.

5 Organisation du projet

5.1 Répartition des tâches

- **Logique du jeu et algorithmes Maxn et Paranoid : Salma Benamar et Yasmine Adnane**
- **SOS et interface graphique : Meghni Mohamed Djawad et Abderezak Benyoucef**

5.2 Collaboration et communication

Le projet a été réalisé de manière collaborative, avec des réunions régulières pour assurer la bonne avancée des tâches.

5.3 Répartition des responsabilités supplémentaires

6 Expérimentations

6.1 Tests d'optimisation

Plusieurs expérimentations ont été réalisées pour tester l'efficacité des différents algorithmes (Maxn, Paranoid, et SOS) en termes de performance, de réactivité et de nombre de mouvements effectués. Ces tests ont permis d'évaluer la qualité des décisions prises par l'IA dans différentes configurations.

6.2 Analyse des résultats

Les résultats montrent une différence notable dans la performance des algorithmes, notamment en fonction de la complexité de l'environnement et des choix des adversaires. Les tests ont permis d'identifier les points forts et les faiblesses de chaque algorithme.

6.3 Pistes d'amélioration

Pour améliorer les performances de l'IA, plusieurs pistes ont été explorées :

- Optimisation de l'algorithme Maxn pour mieux anticiper les mouvements des adversaires.
- Ajout de stratégies adaptatives pour l'IA en fonction des mouvements des autres joueurs.
- Amélioration de l'interface graphique pour rendre l'expérience de jeu plus immersive.

7 Conclusion

Le projet a permis de développer une version interactive et compétitive du jeu Tron, avec une intelligence artificielle réactive et évolutive. Les différents algorithmes implémentés (Maxn, Paranoid et SOS) ont permis de tester différentes approches de la stratégie de jeu, chacune ayant ses avantages et ses inconvénients.

Des améliorations sont envisageables, notamment dans l'optimisation des algorithmes et la gestion de l'interface graphique, pour rendre l'expérience encore plus fluide et réaliste.