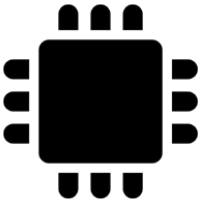
RAPPORT DE PROJET

1ère année de licence d'informatique

AKIBA

Réalisé par Christopher JACQUIOT



UFR CAEN

LICENCE INFO 2016

Table des matières

1. Introduction

Quel est ce projet ? L'objectif de ce projet est de réaliser une version personnelle du jeu d'Akiba, où le principe est d'éliminer toutes les boules de son adversaires ou d'éliminer 6 boules neutres rouges en premier. Cependant une règle de nombre de tours maximum a été ajoutée dans ce projet pour des raisons que nous verrons plus en détails plus bas.

Pourquoi l'avoir choisi ? Ce projet a été choisi parmi deux projets possibles pour sa simplicité et la quantité de temps potentielle pour pouvoir ajouter des fonctions bonus plus élaborées après implémentation des fonctions de base.

Quelles sont les ajouts attendus ? Les fonctionnalités ajoutées les plus importantes de ce projet sont, l'intelligence artificielle basée sur un algorithme génétique, la décision du meilleur mouvement par algorithme NégaMax optimisé par élagage alpha-bêta et une interface graphique propre et permettant la configuration du programme à son lancement.

Comment s'y retrouver ? Une documentation claire et expliquée a été générée avec l'aide de Sphinx et est disponible dans le dossier Docs pour des explications plus détaillées du code source.

D'autres informations : Pour plus de retours sur les différentes fonctionnalités du projet, une période de user-testing va peut-être être réalisée pour recueillir les avis de joueurs non-initiés.

Première partie

Fonctionnalités de base

2. Gestion de la partie - Structure des données du jeu

2.1. Nécessités

Gestion de parties en temps qu'instances d'objets Afin de pouvoir faire tourner plusieurs parties en simultané, nous avons décidé de faire des données de jeu un objet à part entière du nom de 'Game', contenant plateau, joueurs et autres informations afin de simplifier et généraliser la création de parties simultanées.

2.2. Problème

Représenter les boules du plateau Ce jeu as un plateau de 7 cases sur 7 cases pour un total de 49 cases, contenant initialement 13 boules rouges en son centre, 8 boules noires scindées en 2 groupes carrés sur la diagonale Sud-Ouest, Nord-Est, et 8 boules blanches sur la diagonale Nord-Ouest, Sud-Est. Les boules blanches appartiennent au joueur 1 tandis que les boules noires appartiennent au joueur 2, les boules rouges sont neutres.

Comment représenter ce plateau et les boules de chaque joueur ?

Représenter les informations des joueurs et de la partie Chaque joueur à un compteur de boules rouges éliminées et un compteur de boules restantes qui déterminent les conditions de victoire. La partie doit garder trace du joueur qui doit jouer au prochain coup, du nombre de tours joués, des 'objets joueurs' et d'une variable d'affichage.

Comment conserver ces informations de la manière la plus propre et efficace possible ?

2.3. Approches possibles

2.3.1. Représenter les boules du plateau

Point de vue de la grille : tableau à 2 dimensions Si nous représentons la totalité du plateau en mémoire par une liste de 7 listes de 7 cases, l'accès aux cases sera intuitif, en revanche parcourir l'ensemble des boules du plateau nécessite alors la vérification de toutes les cases du plateau sans exception. Soit 49 vérifications pour un maximum de 29 boules. Cette approche est particulièrement inefficace et empire a chaque boule éliminée.

Point de vue du dictionnaire : indexation balle-propriétaire Si nous représentons la totalité du plateau par un dictionnaire regroupant chaque coordonnée de plateau occupé avec son propriétaire, l'accès à une coordonnée est moins intuitive et nécessite de prendre des précautions lors de la manipulation du plateau. En revanche, nous n'avons plus besoin que de parcourir chaque boule du plateau directement pour parcourir l'ensemble des boules du plateau. Ceci dit, la vérification de l'ensemble des boules n'est utile que lors d'une vérification de propriétaire. par conséquent il reste nécessaire de parcourir un maximum de 29 boules pour un maximum de 8 à 21 boules intéressantes.

Point de vue du dictionnaire : indexation propriétaire-balle Si nous représentons la totalité du plateau par un dictionnaire regroupant chaque joueur à une liste de coordonnées de ses propres boules, alors l'accès aux boules deviens plus complexe qu'intuitif et nécessite de vérifier des coordonnées à l'intérieur de 3 listes séparées plutôt que dans une seule liste. En revanche il suffit de spécifier les groupes de boules intéressants dans une vérification pour n'avoir qu'une partie spécifique des boules à vérifier. Par conséquent le nombre de boules à parcourir pour une vérification spécifique est optimal.

2.3.2. Représenter les informations des joueurs et de la partie

Gestion par variables spécifiques Nous pouvons séparer chaque information différente dans des variables spécifiques nommées spécifiquement et faciliter l'accès à ces informations au niveau code ; il n'y a pas de véritable problème avec cette solution.

Indexation des informations dans un dictionnaire Nous pouvons regrouper des informations dans un ou plusieurs dictionnaires et permettre leur accès à travers une clé spécifique ; cela pourrait être utile pour certaines informations similaires mais compliquer l'accès a de simples valeurs telles qu'un unique booléen.

2.4. Approche utilisée

2.4.1. Représenter les boules du plateau

Dictionnaire avec indexation propriétaire-balle Nous avons fait le choix de représenter les boules du plateau par une indexation joueur-balles, afin de privilégier une optimisation du temps maximale. Ces optimisations seront d'autant plus importantes lors de l'entrainement de notre intelligence artificielle par algorithme génétique expliquée plus en détails plus bas.

2.4.2. Représenter les informations des joueurs et de la partie

Indexation des informations liées Plusieurs informations similaires ont étés regroupées dans un dictionnaire afin de permettre un accès simplifié à ces variables ; Par exemple les deux 'objets joueurs' ont été regroupés dans un dictionnaire 'joueurs' ayant deux clés : 'p1' et 'p2'

Variables séparées pour données uniques D'autres informations telles que le booléen d'autorisation d'affichage d'état du jeu 'verbose' ont été conservés dans des variables séparés afin de faciliter leur utilisation.

2.5. Remarques sur les résultats obtenus

Code clair et compréhensible Documentation aidant, le code massif de cette partie du projet reste clair et compréhensif. Les fonctions et variables sont courtes et au plus descriptif possible afin de garder une lisibilité maximale et une compréhension facile des algorithmes mis en place.

Rapidité d'exécution satisfaisante Le temps de modification du plateau a été considérablement réduit en passant d'un prototype en 'grille' à ce type

d'indexation et cela s'est ressenti sur le temps de décision de mouvement de l'intelligence artificielle.

2.6. Pistes d'amélioration

Aucune amélioration trouvée Le code me semble au plus optimisé possible tout en restant clair et suffisamment compréhensible.

3. Gestion de la partie - Règles et déroulement du jeu

3.1. Problème

Gérer le déroulement de la partie Chaque partie se déroule de manière procédurale, le premier joueur bouge s'il le peut, puis c'est au tour du second joueur, et cætera, jusqu'au moment où l'un des deux joueurs aie éliminé son adversaire du plateau ou retiré 6 boules rouge en premier. Pour éviter des parties interminables où les deux joueurs tourneraient en boucle, nous avons ajouté une limite de 50 tours, qui une fois atteinte, déclare match nul.

Ceci dit, comment intégrer cette boucle de jeu dans notre code ?

Gérer les règles du jeu Durant le tour d'un joueur, celui ci doit choisir un mouvement possible, c'est à dire un mouvement de bille qui ne peut pas éliminer une de ses propres billes et uniquement s'il a la place de poser son doigt dans la direction opposée du mouvement, autrement dit : d'une case libre ou le bord du plateau.

Comment intégrer la gestion de ces règles dans notre code ?

3.2. Approches possibles

Gestion de la partie dans une boucle externe à la partie Nous pouvons créer une boucle externe à notre objet Game, dans un module à part, mais cela interdirait ou complexifierais le déroulement simultané de plusieurs objets Game. Ce genre de fonctionnement serait plus utile sur des processus uniques.

Gestion de la partie dans une boucle interne à la partie Nous pouvons intégrer le déroulement de la partie directement dans l'objet Game, cela faciliterais amplement l'utilisation et le codage d'instances de parties. Ce genre

de fonctionnement colle assez bien avec une situation de gestion de multiples processus similaires, ce qui sera notre cas pour l'entrainement de notre IA.

Gestion des règles dans la boucle de jeu elle même Nous pouvons intégrer les règles à l'intérieur de la boucle de jeu elle même, cela permettrais d'avoir un objet Game complet et auto-suffisant, mais cela ajouterais du code à un objet déjà trop chargé.

Gestion des règles externalisée de la boucle de jeu elle même Nous pouvons gérer les règles indépendamment de l'objet en lui passant par argument la situation d'une partie, réduisant ainsi le code de l'objet Game et ajoutant la première couche d'abstraction à ce projet.

3.3. Approche utilisée

Gestion de la partie dans une boucle interne à la partie Nous avons finalement opté pour intégrer la boucle de jeu dans l'objet Game lui même, afin de faciliter l'utilisation de plusieurs parties simultanées.

Gestion des règles indépendante Enfin, nous avons externalisé la gestion des règles dans un module à part pour simplifier la lecture du code et avoir une première couche d'abstraction pour aider à coder les couches supérieures.

3.4. Pistes d'amélioration

Optimiser l'approche orienté objet Nous pourrions regrouper les règles dans les objets Game pour ne plus avoir à passer d'objet en argument, et simplifier l'utilisation du code malgré une légère complication du code.

Optimisation du code de gestion des règles (accès, détermination de l'état de ces règles) Certains algorithmes utilisées pour la vérification de certaines règles pourraient être potentiellement plus optimisés pour réduire le temps de calcul d'état d'une règle et accélérer un peu plus la décision de mouvement.

4. Mode console

4.1. Nécessités

Permettre au joueur de voir l'état de la partie en cours A chaque tour, le joueur doit pouvoir avoir une idée du plateau et de où se situe ses boules et celles adverses. Notre interface doit pouvoir donner les informations nécessaires pour se repérer à l'utilisateur.

Permettre au joueur de savoir comment interagir De même, l'utilisateur va avoir besoin de savoir comment interagir avec le jeu pour lui communiquer son prochain mouvement. Notre interface doit donc présenter le format des commandes demandées à l'utilisateur.

Informer le joueur sur les choix possibles ou non Lorsque l'utilisateur se trompe, il doit aussi être informé de la raison pour laquelle sa commande n'est pas acceptée. Notre interface doit donc prendre aussi en compte une gestion des erreurs et un affichage de messages d'erreurs.

4.2. Problème

Communiquer des choix entre l'utilisateur et le jeu Il y a plusieurs façons de permettre à un utilisateur de communiquer avec une machine mais dans notre cas, quelle type de système d'entrée serait le plus adapté ?

4.3. Approches possibles

Système de commandes Dans ce système, l'intervention de l'utilisateur est centrale, il faut gérer plusieurs types de commandes possibles pour plusieurs types d'actions possibles. Par exemple une commande 'help' pourrait rappeler les règles tandis que la commande 'move' serait mandataire pour effectuer un

mouvement. Ce type de système serait plus adapté à un outil ou un vieux RPG qu'à un jeu aussi procédural et simple.

Système procédural Dans ce système, l'utilisateur ne peut entrer qu'un type de réponse bien précis et bien défini. Une entrée non conforme sera refusée et l'utilisateur devra entrer une nouvelle réponse. En revanche ce genre de système empêche l'accès a des commandes telles qu'un rappel des règles. Ce type de système convient bien a notre projet pour son coté simple et rapide, et permet d'éviter de complexifier le code source outre-mesure.

4.4. Approche utilisée

Intervention procédurale de l'utilisateur Nous avons opté pour un système d'entrée procédural sans commandes afin de permettre a l'utilisateur de rester dans le fil du jeu sans avoir a taper de commandes. Le déplacement est constitué des coordonnées en colonne puis en ligne de la boule poussée ainsi que de la direction du mouvement.

4.5. Remarques sur les résultats obtenus

Rappel des règles souhaitable Pour des nouveaux joueurs, il faut expliquer les règles en amont, car le jeu n'as actuellement pas de rappels de règles intégrés au mode commande.

Manque d'informations cruciales Certaines informations telles que le compteur de boules rouges éliminées ont été oubliées durant la création du design de l'interface console et ne sont pas affichés, cela force les joueurs a garder un compte des boules rouges qu'ils ont pu éliminer jusqu'ici.

4.6. Pistes d'amélioration

Meilleur affichage du plateau et des informations Certaines informations cruciales sur la partie devraient être ajoutées à l'interface afin de permettre aux joueurs de se concentrer purement sur le jeu au lieu d'avoir a garder le compte des boules éliminées.

Reconnaissance d'une commande spécifique dédiée à l'affichage des règles
Une commande help de rappel des règles pourrait être un ajout simple et utile pour les nouveaux joueurs qui ne seraient pas familiers avec celles ci.

Deuxième partie

Interface graphique

5. Mode graphique

5.1. Nécessités

Améliorer la présentation des informations sur la partie Afin de rendre l'interface graphique plus utile que la version console, cette interface doit permettre un affichage clair et plus facile à lire des différentes informations du jeu. Pour cela nous allons pouvoir utiliser des images pour représenter les divers types d'informations (association logo-information) et jouer de la disposition des informations pour aider à leur compréhension.

Faciliter l'accès aux règles aux joueurs néophytes Afin de permettre aux nouveaux joueurs de pouvoir trouver les règles du jeu, un bouton ouvrant une fenêtre à part contenant les règles sera disponible afin de permettre l'accès à ses règles sans encombrer l'espace de jeu.

5.2. Problème

Simplifier les interactions de l'utilisateur Pour avoir un gameplay agréable et intuitif, l'interface graphique va avoir besoin de trouver une méthode de sélection de mouvement facile à utiliser et qui ne casse pas le rythme du jeu, tout en prenant en compte les mouvements possibles ou non.

Comment simplifier le choix de mouvement sans pour autant casser le rythme du jeu ?

5.3. Approches possibles

Système de commandes par sélection manuelle de boule puis de direction Cette approche consisterait à utiliser une quelconque méthode de sélection de boule passant par une liste ou un clic sur la case choisie, ainsi que d'un choix de direction par bouton non grisé. Ce genre de méthode serait utile dans le

cas où il n'y aurait ni restrictions de mouvements ni mouvements, ce serait plus adapté à des jeux de gestions avec divers paramètres à modifier tel que la fonction de tel ou tel objet, etc.. Dans notre cas cela donnerais une pauvre expérience de jeu par le manque de réactivité du jeu au clic de l'utilisateur.

Système de sélection de mouvements possible Cette approche consisterait à afficher à l'écran tous les mouvements possibles et ne laisser le joueur cliquer que sur l'un de ces mouvements, facilitant ainsi le choix des mouvement, améliorant l'impression de réactivité de l'interface, une bonne intégration de ce genre de système serait une intégration graphique de ce système dans notre représentation du plateau, afin de pouvoir voir les boules déplacées facilement. Ce genre de système est idéal dans notre cas où nous pouvons avoir beaucoup de mouvements possibles mais de nature limitée.

5.4. Approche utilisée

Choix du mouvement par unique clic Nous avons finalement décidé de représenter notre plateau avec une grille de cases, où chaque case contient 4 directions possibles avec 2 états possibles : possible ou non, ainsi que 3 possibles types de boules. A chaque tour les sélecteurs sont mis à jour pour permettre au joueur correspondant de savoir quels sont les mouvements qu'il peut faire et un clic sur un sélecteur non actif est ignoré.

5.5. Remarques sur les résultats obtenus

Interface trop petite La représentation du plateau a été faite pour représenter des cases de 32 pixels sur 32, et pour certains joueurs la taille de l'interface peut être problématique, il faudrait voir à pouvoir re-dimensionner ou agrandir l'interface à un niveau confortable.

5.6. Pistes d'amélioration

Retrait des textures superflues et amélioration de l'affichage de l'information Les textures de décoration et de background donnent des détails visuels inutiles pour le déroulement du jeu et encombrent l'espace visuel déjà très

restreint, limitant la clarté des informations affichées à l'écran. Le retrait des textures pourrait être bénéfique à l'affichage des informations.

Rendre le plateau de jeu de taille réglable La taille des cases de 32x32 et des sélecteurs de 8x16 est trop petite pour être cliquée confortablement, un agrandissement des dimensions ou un redimensionnement serait souhaitable pour une meilleure expérience de jeu.

Améliorer l'intuitivité de l'interaction avec un tutoriel ou un curseur en forme de doigt Il n'y a actuellement aucun guide qui n'explique la façon d'interagir avec le jeu, un tutoriel ou un curseur en forme de doigt qui pousse une boule pourrait aider à expliquer ou rendre plus intuitif les interactions avec le plateau.

Un guide du meilleur coup ? Pour aider les joueurs à se décider, il serait possible d'ajouter un nouveau type de sélecteur, le sélecteur 'guide', qui pourrait commencer à s'afficher au bout d'une durée de quelques dizaines de secondes et s'afficher sur le meilleur mouvement possible pour guider un nouveau joueur en difficultés par exemple. Cela pourrait être aussi activé ou non au lancement du jeu.

Meilleur formatage de l'affichage des règles La pop-up de règles n'est pas très sexy pour le moment, un meilleur formatage des règles serait souhaitable voire une infographie pourrait rendre leur lecture plus agréable et plus simple à retenir.

Troisième partie

Intelligence artificielle - Algorithme génétique

6. Type d'intelligence artificielle

6.1. Restrictions

Aucune information de stratégies sur ce jeu Ce jeu est particulièrement inconnu, par conséquent nous n'avons pas de guides ou de championnats de ce jeu qui nous permettrait d'avoir des informations sur les stratégies intéressantes pour gagner. Nous allons tout de même prendre en compte les conseils stratégiques de jeux similaires tels que Abalone afin d'avoir quelques pistes.

Pas de données d'entrainement disponibles Encore une fois ce jeu n'as pas de compétitions, par conséquent nous n'avons pas de guides de mauvais coups ou de m=bons coups que nous pourrions utiliser pour entrainer notre IA. Nous allons donc être restreint à de l'entrainement manuel ou de l'auto-entrainement.

6.2. Nécessités

Plusieurs niveaux de difficultés Afin de pouvoir permettre au joueur de jouer contre plusieurs niveaux d'ia possibles, il va nous falloir enregistrer plusieurs versions de difficultés différentes que l'on pourra choisir au démarrage du jeu.

6.3. Problème

Besoin de choix logiques automatisés pour jouer contre un joueur Nous avons besoin d'une IA suffisamment développée pour pouvoir jouer contre un humain, par conséquent nous n'allons pas pouvoir utiliser une détermination de meilleur mouvement aléatoire, nous allons avoir besoin d'une IA capable de prendre en compte plusieurs facteurs et de réagir selon diverses situations.

Quel type d'IA serait le plus adapté à notre situation ?

6.4. Approches possibles

Réseau neuronal artificiel de deep learning Le deep learning serait utile pour permettre à l'IA d'apprendre à reconnaitre les patterns des meilleurs coups selon diverses situations. Seulement, ce genre d'IA nécessite une grande quantité de données d'entrainements, ce que nous n'avons pas a notre disposition.

Réseau neuronal artificiel à apprentissage par renforcement positif Les IA à renforcement positif serait utile pour apprendre à l'IA quels mouvements sont les plus intéressants dans diverses situations et pourrait être entrainé manuellement, seulement nous n'avons pas d'experts de ce jeu pour donner de véritables bons jugements sur la situation d'un jeu, de plus il faudrait prendre en compte le facteur de la fatigue de l'humain et non n'avons pas de données pour pouvoir automatiser cela non plus.

Algorithme génétique Les algorithmes génétiques sont utiles pour trouver des compromis entre entraînement et manque de données. Nous n'allons pas avoir besoin de données de départ, et la sélection permettra de conserver les meilleurs IA uniquement, que nous continueront de confronter a d'autres nouvelles IA jusqu'à trouver une IA particulièrement robuste.

6.5. Approche utilisée

Algorithme génétique Nous allons finalement opter pur un algorithme génétique afin de pallier au problème de manque de données d'entrainement. Cela aura le mérite de ne pas avoir besoin de données d'entrainements mais demandera plus de temps pour parvenir a une solution optimale, et limitera l'efficacité de cette méthode selon la pertinence des divers méthodes utilisées pour cette IA (sélection, heuristique, génome, fitness..)

7. Évolution indépendante

7.1. Nécessités

Aucune supervision humaine sur l'efficacité d'un génome Cette IA n'aura aucune supervision humaine lors de son entrainement afin de pourvoir enchainer les générations au plus vite et évoluer au plus vite. Par conséquent il va être nécessaire de bien préparer les limites et la manière dont elle va évoluer par elle-même.

Mutations aléatoires pour sortir d'extremum locaux parfois non optimaux Afin d'éviter de rester bloqué avec une solution optimale localement, l'ajout de mutations aléatoire peut permettre de changer la solution en une meilleure mais non optimale et ainsi permettre d'accéder à une nouvelle et meilleure solution optimale.

7.2. Problème

Besoin de convergence vers une meilleure solution Plus notre IA sera douée, plus celle ci vaincra d'adversaires facilement. Autrement dit, notre meilleure IA devra avoir les meilleurs paramètres possibles pour faire face au plus grand nombre de types d'adversaires possibles. Il va falloir qu'une IA d'abord aléatoire aie des paramètres qui convergent vers cette meilleure solution.

Comment gérer cette convergence ?

7.3. Approches possibles

Sélection aléatoire Ce genre de sélection permet d'atteindre un maximum de solutions possibles différentes, au détriment de conserver des solutions parfois moins bonnes que certaines ayant été éliminées par l'aléatoire. Ce genre de

sélection est utile dans notre cas, où plusieurs solutions optimales peuvent exister, mais prends plus de temps pour converger.

Sélection élitiste Ce genre de sélection permet de se focaliser sur les solutions les plus performantes d'un groupe au détriment de nouvelles solutions en devenir qui s'avèreraient plus efficaces si elles évoluaient plus longtemps. Ce genre de sélection est utile dans le cas où peu de solutions optimales sont attendues comme le résultat d'un calcul, mais est aussi plus rapide a converger.

Sélection par reproduction générale Ce genre de sélection permet de mélanger les meilleures solutions avec des solutions moins optimisées pour peut être tomber sur des combinaisons plus intéressantes au détriment de potentiellement perdre une combinaison déjà optimale. Ce genre de sélection est un compromis entre les deux solutions ci dessus en temps de convergence, mais s'applique mieux a des problèmes où quelques solutions optimales différentes peuvent exister.

7.4. Approche utilisée

Sélection élitiste sur l'efficacité en temps et en score Afin de sélectionner les meilleures IA, nous allons mesurer leurs performances dans leurs matchs par rapport au nombre de tours nécessaires avant la fin de la partie, de l'issue de la partie (victoire, défaite ou temps écoulé) ainsi que des pertes subies et de celles infligées. Les meilleures IA seront les plus efficaces en temps et en élimination.

Mutations à fréquence constante Pour éviter de rester bloqué sur des solutions non optimales, nos meilleures IA vont avoir besoin de subir des modifications aléatoires, ici à fréquence constante pour simplifier l'approche des mutations dans notre algorithme, bien que ce ne soit pas la meilleure approche possible dans notre cas.

7.5. Remarques sur les résultats obtenus

Adaptation des IA générées rapide et positive Les IA partant d'un génome aléatoire ont vite commencé à comprendre diverses manœuvres inattendues

telles que de bloquer tout mouvement adverse en isolant et enfermant leurs billes, ou en faisant des boucles infinies sur des successions de plusieurs mouvements répétées pour ne pas perdre l'avantage. Une règle de limite de tours à même dû être implémentée pour empêcher ce genre de boucle infinie de perturber le fonctionnement des sessions d'entrainement.

Convergence un peu trop rapide Des solutions optimales locales sont atteintes assez rapidement mais n'évoluent pas très rapidement vers des solutions différentes, il est possible que la méthode de sélection soit trop élitiste ou qu'il faille revoir la méthode d'entrainement entre IA.

7.6. Pistes d'amélioration

Fréquence de mutation dynamique Une fréquence de mutation dynamique pourrait aider à conserver une certaine précision sur notre volonté de mutations, une fréquence plus forte au départ pourrait permettre d'avoir un plus large champs de solutions au début de l'entrainement et permettrait de réduire petit à petit le champ des nouvelles solutions explorées à mesure que l'on avance en générations en diminuant la fréquence de mutations. D'autres méthodes comme une augmentation de la fréquence à mesure que la solution optimale stagne permettrait aussi d'éviter de rester coincé sur une seule solution locale et de passer a coté d'une potentielle meilleure solution.

Sélection moins stricte Notre sélection est stricte, voire un peu trop, conserver des perdants avec des aspects plus intéressants telles qu'une meilleure efficacité en temps ou en élimination de rouges pourrait permettre de développer des solutions plus efficaces et plus spécialisées sur une stratégie que notre sélection actuelle qui favorise les stratégies équilibrées au détriment de meilleures stratégies plus lentes à apparaître.

8. Heuristique - Prise de décision

8.1. Problème

Faire agir le génome d'une IA sur son comportement Le comportement de notre IA est basée sur son génome, celui ci va lui dicter quels mouvement favoriser au dépit d'autres mouvements. Mais comment déterminer les critères des meilleurs mouvements ?

Comment relier le génome de l'IA à la décision de mouvements ?

8.2. Approches possibles

Laisser l'IA favoriser divers comportements généraux Avec cette approche, l'IA va préférer un certain set d'actions classées sous plusieurs groupes. Par exemple l'IA peut préférer jouer contre les rouges, a ce moment la l'ia pourra pondérer le groupe d'action 'offensif contre les rouges' plus fortement que les autres groupes tels que 'défensif' ou 'offensif contre l'adversaire' et laisser le hasard décider du groupe d'actions choisi. Le problème avec ce principe c'est que l'IA perd toute capacité d'adaptation, et ne sera pas vraiment très utile dans une partie contre un humain.

Laisser l'IA décider de l'importance de divers facteurs d'une situation Avec cette approche, l'IA va pondérer divers facteurs de la partie en cours pour déterminer de l'avantage de la situation pour elle même. Plus la valeur de la situation est élevée, plus l'action qui y mène est favorable. Par conséquent, l'IA ici va pouvoir réagir aux choix faits par son adversaire en cherchant à conserver un maximum l'avantage et à nuir à la situation de l'adversaire. Le problème de cette approche est aussi celui qui sera pris en charge par l'algorithme génétique, c'est la valeur de l'importance de chaque facteur dans cette pondération. Les meilleures combinaisons de facteurs seront les meilleurs génomes.

8.3. Approche utilisée

Importance d'une situation par pondération linéaire Nous avons finalement opté pour une pondération linéaires d'un maximum de facteurs possibles pour permettre à l'IA de déterminer elle même les meilleurs facteurs possibles chaque facteur aura une valeur comprise entre 0 et un maximum M, et les situations les plus bénéfiques seront les situations avec la somme de facteurs pondérés ayant le plus de valeur.

Génome de multiplicateurs Le lien avec le génome et l'heuristique sera dans la découpe de ce génome. Celui ci est composé de gênes de même longueurs, ce sont des mots d'un nombre défini de bits, qui représentent un nombre entier entre 2^n et 0. Ces nombres correspondent à la pondération de chaque facteurs définis dans le calcul de la valeur d'une situation.

8.4. Remarques sur les résultats obtenus

Certains facteurs sont ignorés au profit d'autres On peut remarquer que certains facteurs sont ignorés par rapport à d'autres, et que certains sont même mis à zéro dans certaines solutions optimales locales, certains facteurs définis dans notre heuristique seraient donc potentiellement contre productifs ?

8.5. Pistes d'amélioration

Ajout de facteurs plus pertinents, plus mathématiques Afin de permettre à l'IA plus de nuances et de pertinence dans son choix de facteurs, il serait intéressant de trouver plus de facteurs possibles que les facteurs déjà définis. Cela lui permettrait peut être aussi de découvrir des stratégies plus complexes et plus inattendues.

Augmentation de la taille des gênes pour leur permettre plus de valeurs possibles La taille des gênes influe directement sur l'ensemble de valeurs possibles par pondérations, un plus grand gène augmenterait le nombre de valeurs possibles et par conséquent la possibilité de régler plus finement les détails des pondérations en jeu, et leurs importances relatives.

9. Heuristique - Optimisation des mouvements choisis

9.1. Nécessités

Une partie dans un arbre de possibilités

9.2. Problème

Sélection des meilleurs mouvements potentiels Maintenant que nous avons déterminé comment notre IA évalue les meilleures situations, nous allons pouvoir déterminer quel mouvement sera le plus intéressant parmi tous ceux possibles. Seulement, parcourir chaque possibilités risque d'être lent et lourd en calculs.

Comment pourrait-on réduire la quantité de mouvements possibles à parcourir ?

9.3. Approches possibles

Parcours min-max sans élagage Cette approche consiste à tester toutes les possibilités possibles sans exceptions pour déterminer du meilleur choix possible. C'est une approche facile à implémenter mais couteuse en calculs dûs à de potentielles solutions non prometteuses. Un algorithme de min-max ou NégaMax peut servir à l'implémentation de ce genre d'approche pour déterminer des meilleurs parcours.

Parcours min-max avec élagage alpha-bêta Cette approche consiste à éliminer des possibilités dont la valeur est inférieure à une certaine valeur prédéterminée. L'avantage de ce genre de solution est qu'il est plutôt rapide et permet d'accélérer une recherche d'arbres en évitant de parcourir des branches considérées trop

peu prometteuses. Le désavantage est le fait d'avoir à déterminer empiriquement une valeur de bêta, et que certaines solutions avantageuses rares peuvent être ignorées d'avance par des mouvements préliminaires considérés non prometteurs. Cela peut permettre d'accélérer le résultat d'un algorithme min-max.

Méthode de Monte-Carlo Cette approche consiste à sélectionner un nombre aléatoire de possibilités à chaque état du jeu et de recommencer jusqu'à pouvoir vérifier la quantité de victoires apportées par chaque possibilités, ce genre d'algorithme est puissant pour des jeux où un mauvais mouvement n'est pas immédiatement perdant, et est plus rapide qu'un algorithme de min-max, celui ci est aussi aisément adapté à n'importe quel type de jeux car aucune heuristique n'entre en jeu lors de cet élagage. Par ailleurs le résultat de ce type d'algorithme converge vers le résultat d'un algorithme min-max.

9.4. Approche utilisée

L'algorithme NégaMax, simplification du Min-max En premier temps, nous avons implémenté un algorithme NégaMax, variante du min-max, qui à la place de comparer alternativement les maximums de nombres négatif puis les minimums de nombres positif, va constamment convertir les nombres positifs en négatifs et n'aura plus qu'à comparer les maximums à chaque fois. Ces deux algorithmes fonctionnent de la même manière, on récupère tous les mouvements possibles et pour chaque mouvement possible on recommence jusqu'à atteindre une fin de partie ou une profondeur limite définie. C'est à ce point là que l'on calcule la valeur de la situation à l'aide d'une fonction heuristique, puis l'on remonte l'arbre en conservant ici la possibilité avec le score maximum jusqu'à déterminer le meilleur mouvement à prendre à partir de la racine.

Élagage alpha-bêta L'élagage alpha-bêta nous permettra d'éliminer d'office toute branche de possibilités dont le meilleur score est inférieur à un score considéré peu prometteur. l'utilisation du NégaMax facilite l'implémentation d'un tel système car il n'y a plus besoin de transformer les bornes de positif à négatif et inversement à chaque étape.

9.5. Remarques sur les résultats obtenus

Performances grandement accélérées Les décisions de mouvements ont été grandement accélérées à plusieurs moments, d'environ 7 fois au moment de l'implémentation de l'indexation des boules sur le plateau et d'environ 2 à 3 fois au moment de l'ajout de l'élagage alpha-bêta

9.6. Pistes d'amélioration

Utilisation de la méthode de Monte-Carlo L'utilisation d'une méthode pareille pourrait s'avérer intéressante, seulement son attrait est limité dans l'état actuel du jeu du fait de la limite de 50 tours instaurées, la précision de cette méthode dépendant grandement de la quantité de possibilités explorées. Seulement le retrait de cette règles restaurerais le problème de partie infinie et l'ajout d'un plus grand nombre de tours rendrait le jeu ennuyant pour un humain bloqué par l'IA, dans l'état actuel, si l'IA le force dans une situation de boucle infinie, le nombre de tours limités force le joueur à changer de stratégie rapidement pour ne pas gâcher de précieux mouvements. Un nombre de tours plus grand empêcherais ce facteur de stress avant un trop long nombre de tours et ennuierai le joueur.

Exploration simultanée de plusieurs arbres de possibilités Le multi-threading pourrait s'avérer utile dans la recherche d'arbre de possibilités, chaque thread parcourant une branche différente ou un ensemble de branches différents. Une tentative avait été faite mais la limite de thread maximums avait été dissuasive et le manque de temps pour le reste du projet risque d'avorter cette idée pour ce projet. Mais cela pourrait accélérer le calcul considérablement.

Quatrième partie

Intelligence artificielle - Entrainement

10. Mode entrainement

10.1. Nécessités

Besoin de pouvoir modifier divers paramètres de départ Afin de pouvoir générer de nouvelles IA et les entrainer de différentes façons comme de manière plus courte ou plus intensive, nous allons avoir besoin de pouvoir configurer certaines paramètres de départ.

10.2. Problème

Besoin d'un entrainement aussi rapide que possible Afin que nos IA puisse évoluer et s'améliorer le plus efficacement possible, il va nous falloir approcher notre problème d'entrainement pour déterminer les facteurs les plus ralentissant dans une exécution python et optimiser notre code un maximum.

Comment rendre notre entrainement le plus efficace possible ?

10.3. Approche utilisée

Mode sans intervention humaine Le principal élément de temps perdu dans l'exécution d'un programme est l'attente de l'entrée utilisateur. Pour favoriser une exécution rapide de notre mode, nous avons ôté toute interactivité avec l'utilisateur durant le fonctionnement de celui ci.

Approche algorithmique la plus linéaire possible Afin d'éviter des pertes de temps à cause de notre code, nous avons tenté à tout prix de réduire la complexité de nos algorithmes le plus linéairement possible pour éviter des ralentissements à certains niveaux de notre code.

Minimum d'informations à afficher pour ne pas ralentir son exécution Enfin, un autre facteur important de ralentissement lors de l'exécution d'un pro-

gramme python est la lenteur d'un affichage console. Appeler la fonction print de multiples fois en peu de temps ralentit considérablement le code pour des raisons de vitesse maximale de terminal. Afin de ne pas dépendre de facteurs tels que la vitesse d'exécution d'un terminal, nous avons réduit l'affichage d'information au strict nécessaire et permettront d'activer ou non l'affichage de certaines informations supplémentaires en cours d'entrainement.

10.4. Pistes d'amélioration

Affichage de statistiques sur l'état de l'entrainement Des informations telles que des diagrammes sur les meilleures IA actuelles ou la progression moyenne des scores et des meilleurs IA en direct durant l'entrainement pourrait être très instructif sur l'efficacité et la progression des solutions trouvées à chaque instant de l'entrainement.

Multi-threader les parties entre bots Encore une fois le multi-threading pourrait être intéressant pour accélérer le calcul des résultats de chaque parties de manière séparée. Cela pourrait diviser le temps d'entrainement de chaque génération significativement sur du matériel approprié tel que des processeurs multi-coeurs.

11. Sauvegarde des générations d'IA

11.1. Nécessités

Besoin de conserver les meilleures IA Une fois entrainées, nous avons besoin d'avoir un moyen de conserver une trace des meilleurs IA d'une session d'entrainement spécifique, afin pouvoir les retrouver plus tard pour tenter de faire des parties avec ou pour reprendre une nouvelle session d'entrainement avec ce parent.

Besoin de classer les IA Nous avons besoin de pouvoir retrouver facilement le classement des IA entrainées par rapport aux autres, afin d'avoir une idée de la puissance relative des différentes IA sauvegardées.

11.2. Problème

Classement et sauvegarde des IA Ayant besoin de sauvegarder et de classer les IA après chaque génération, il nous faut trouver un moyen de les distinguer entre elles et de les stocker de façon à pouvoir les retrouver aisément.

Quel type de sauvegarde pourrait nous être utile ?

11.3. Approches possibles

Base de données unique Une base de données pourrait s'avérer utile pour notre cas, la sauvegarde des IA serait aisée, et le classement de diverses données telles que le score moyen, le meilleur score, la génération ou encore les paramètres de départ liés à chaque IA pourrait être implémenté dans la base de donnée directement, tout en ne conservant qu'un seul fichier à gérer.

Conserver chaque meilleur élément de chaque génération dans un fichier séparé Un nom de fichier composé des différents paramètres de départ et

actuels de l'entrainement pour chaque IA pourrait être plus simple à coder et implémenter, bien qu'après quelques entrainements il y ai de fortes chances qu'il devienne difficile de se repérer dans le dossier de sauvegardes.

11.4. Approche utilisée

Fichier indépendant par génération d'IA Nous avons finalement opté pour la solution la plus simple parmi les deux afin de pouvoir dédier plus de temps à l'heuristique. Chaque génération se voir générer un nouveau fichier dont le nom dépends de la génération actuelle, des paramètres de départ et du meilleur score parmi les éléments de la génération actuelle. chaque fichier contient alors le génome et les score des deux meilleurs IA de la génération.

11.5. Remarques sur les résultats obtenus

Classement des fichiers difficile Après quelques générations il deviens difficile de se retrouver parmi tous les fichiers générés, il serait préférable d'opter pour un autre système de classement.

11.6. Pistes d'amélioration

Base de données Après constatations une base de données pourrait être la meilleure solution pour conserver et trier les générations d'IA générées. Ce genre de base de donnée pourrait aussi contenir les valeurs de chaque gène pour faciliter des analyses statistiques ultérieures.

Cinquième partie

Lanceur principal

12. Choix des modes

12.1. Problème

Choisir quel mode lancer sans changer le code source Afin de permettre une utilisation confortable et pratique de ce projet pour un néophyte, il serait utile de lui présenter comment accéder aux différents modes intégrés dans ce projet de manière assez explicite.

De quel manière devrions nous montrer l'accès aux différents modes à l'utilisateur ?

12.2. Approches possibles

Multiples lanceurs séparés Cette approche consiste à créer un lanceur différent par mode à la racine du projet et permettrait d'avoir une certaine séparation entre chaque mode pour qu'un utilisateur lambda ne puisse pas tomber trop facilement sur le mode d'entrainement d'IA et ne s'y perdes par exemple. Ce genre d'approche est plus fool-proof mais nécessite un certain effort pour changer de mode.

Lanceur principal Cette approche consiste à créer un seul lanceur dans lequel choisir directement l'un des trois modes à lancer. Cela permet de centraliser l'accès aux différents modes et de faciliter l'utilisation du projet dans son ensemble.

12.3. Approche utilisée

Unique lanceur principal Nous avons finalement opté pour un unique lanceur afin de ne pas avoir à changer de fichier à lancer pour changer de mode. moins frustrant et permet de faire un raccourci au lieu de trois.

12.4. Pistes d'amélioration

Design plus intuitif (boutons logos) Remplacer les noms des boutons par des logos cliquables pourrait être intéressant pour proposer une expérience utilisateur de l'interface plus fluide, agréable et intuitive que les trois boutons inégaux actuels.

13. Configuration des modes

13.1. Problème

Choisir des paramètres différents pour différents modes Afin de pouvoir paramétrer à notre convenance le fonctionnement de l'application, nous allons avoir besoin de panneaux de configuration où changer des variables.

Comment gérer le paramétrage et le choix des modes ?

13.2. Approche utilisée

Création de menus de configuration séparés Nous avons fait le choix de faire apparaitre un onglet de configuration lors du choix d'un mode, et ne comprenant que les paramètres nécessaires et pertinents de chaque mode d'utilisation du projet. Il est alors nécessaire de choisir un mode pour changer des paramètres et l'œil est ainsi moins chargé niveau informations aux différentes étapes.

13.3. Pistes d'amélioration

Design plus intuitif Des logos et des représentations graphiques, plus espacées pourraient être plus utiles pour représenter facilement le sens de divers paramètres et rendre la configuration plus simple et agréable à l'utilisateur.

Sauvegarde des derniers paramètres Pouvoir conserver les derniers paramètres utilisés pourrait être utile afin de conserver des préférences d'utilisations sur un niveau d'IA par exemple.