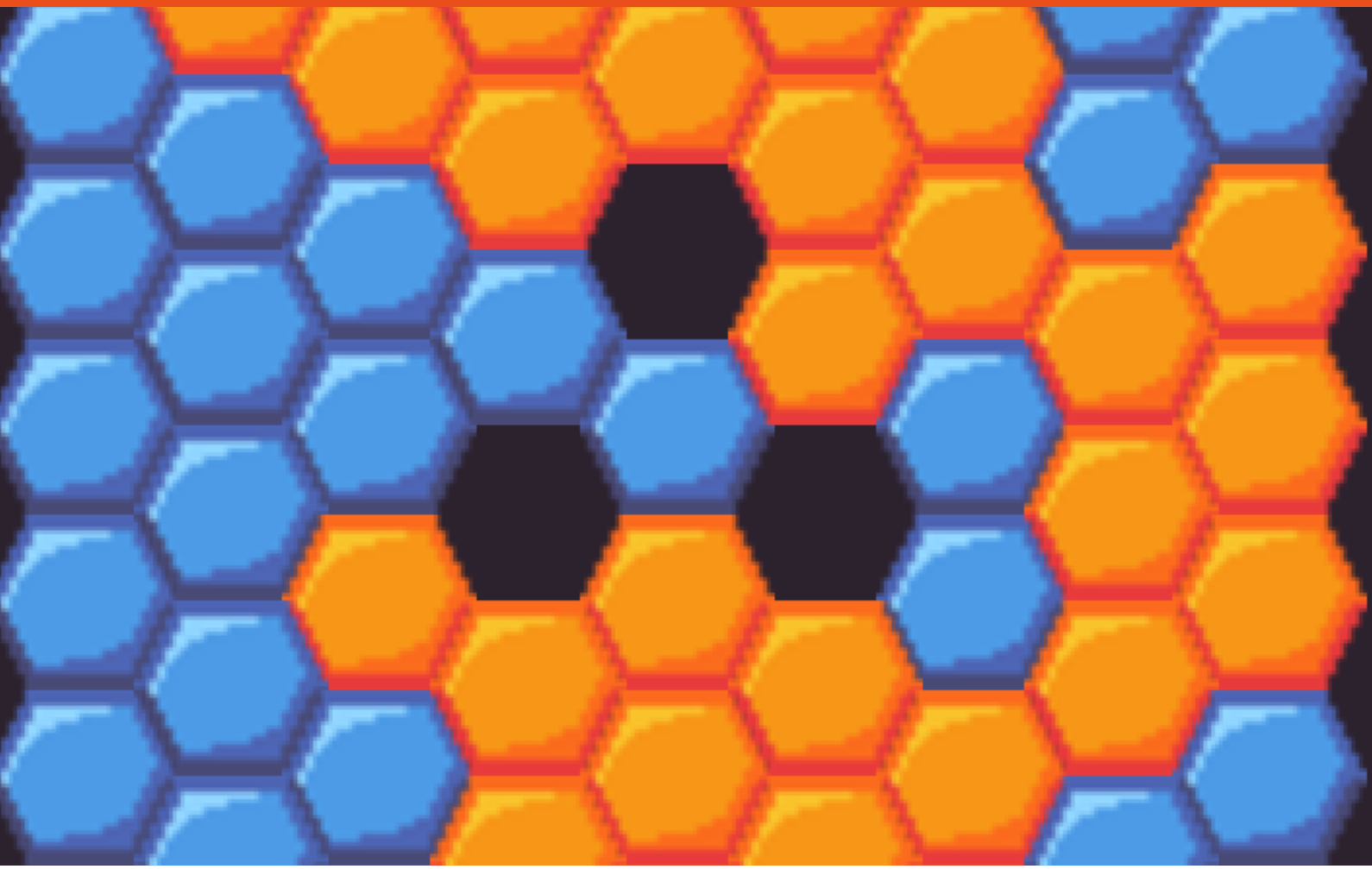
LES TROPHÉES NSI

Édition 2023

DOSSIER DE CANDIDATURE
PRÉSENTATION DU PROJET



Ce document à été rédigé en vue des Trophées NSI et contient 6 pages (4 de rapport et 2 de documentation)

Le programme détaillé dans ce document se trouve sur GitHub

Vous avez des questions sur le concours ? Vous souhaitez des informations complémentaires pour déposer un projet ? Contactez **info@trophees-nsi.fr**.

NOM DU PROJET : PyHexx

> PRÉSENTATION GÉNÉRALE :

PyHexx est un jeu de stratégie amusant et addictif basé sur le jeu Hexxagon publié par Argo Games en 1992 pour la plateforme MS-DOS (qui lui même est basé sur le jeu Ataxx, sortie en 1990). Dans ce jeu, votre objectif est de conquérir autant de cases que possible en déplaçant vos pions à travers le plateau, cependant il est hexagonal ! Le joueur qui possède le plus grand nombre de cases à la fin du jeu est déclaré vainqueur.

Avec l'envie de créer un projet original et à la fois amusant, notre équipe a décidé d'essayer de reproduire ce jeu devenu maintenant méconnu de notre génération Z. Grâce aux bases et connaissances apprises au cours de notre année scolaire et aux nombreux projets effectués, nous nous sentions capable d'utiliser correctement la bibliothèque python pygame pour ce projet.

PyHexx se déroule donc sur un plateau composé d'hexagone, et chaque joueur commence avec trois pions. Chaque tour, vous pouvez déplacer un pion de deux hexagones dans n'importe quelle direction, ou cloner un pion sur un hexagone adjacent. Chaque pion de l'adversaire adjacent au pion que vous venez de placer devient le vôtre. Le gagnant est le joueur qui possède le plus de pions une fois le plateau rempli, ou le joueur qui élimine tous les pions de son adversaire, auquel cas le jeu se remplit automatiquement de ses pions.

> ORGANISATION DU TRAVAIL :

Notre équipe se constitue de 2 membres: Adrien Mitton (chef de projet) et Cosmo Bordier

La répartition des tâches étaient la suivante:

- Adrien: Programmation de la logique du jeu, du "Board Editor" et des bots.
- Cosmo: Programmation des boutons (mute, menu, reset, etc...), de l'interface utilisateur (écran menu et info). Ainsi que la création de tous les sprites et l'implémentation de la musique et des effets sonores.
- Nous avons tout les deux travaillé sur le rapport et la vidéo

Une fois nos rôles décidés, nous avons mis en place des moyens de communication (groupe Discord) en plus de nos réunions hebdomadaires en présentiel par l'intermédiaire de notre Coding Club (club créé par Adrien dans le but de créer un espace propice à la programmation et au partage de projets informatiques dans notre collège/lycée). Nous avons aussi utilisé GitHub pour mieux collaborer sur notre code et pour garder une liste de tâches (mais nous avons réalisé un peu tardivement que nous pouvions gérer les bugs grâce aux issues de GitHub).

LES ÉTAPES DU PROJET :

Tout commence lorsque nos professeurs de NSI respectifs nous présentent les Trophées NSI en début d'années et nous décidons d'y participer

Après de nombreuses sessions de brainstorming pendant le Coding Club, nous avons (enfin) choisi un projet que nous pensions à la fois intéressant, assez complexe et respectant les règles des Trophées NSI (language et notions étudiées pendant les cours de NSI). L'idée de créer un clone de Hexxagon vient d'un ami d'Adrien qui lui-même avait découvert un port web du jeu plusieurs années auparavant et auquel ils avaient joué ensemble.

Nous nous lançons donc dans la phase de planification: nous faisons usage du tableau blanc de la salle mise à notre disposition pour le Coding Club afin d'établir les dimensions de l'espace de jeu et des pions (qui n'a au final meme pas été gardé), la liste de tâche initiale (qui ne fera que s'allonger au cours du projet), et surtout le "grid": soit la représentation en python du tableau de jeu hexagonal, ainsi que la logique des différents mouvements que le joueurs peut effectuer lors de son tour

Nous avons ensuite créé le GitHub et le groupe Discord et commencé à implémenter le "grid". Étant donné que nous travaillions à l'époque sur le chapitre Web du cours de NSI, nous pensions pouvoir faire une interface web pour le projet grâce au module Flask ou au langage JavaScript. Malheureusement, après quelques tests, nous avons réalisé que PyGame était plus adapté à nos besoins (et surtout plus simple à mettre en place).

Malheureusement, nous avons dû mettre le projet de côté afin de nous concentrer sur nos cours dans d'autres matières et ce n'est qu'au début du mois d'avril que nous réalisons à quel point nous sommes probablement en retard par rapport aux autres. Nous nous lançons donc sur l'affichage en utilisant des sprites primitifs tirés directement du jeu original ou dessiné rapidement sur photoshop et réussissons rapidement à construire la base du jeu. Puis, nous nous sommes attaqués à l'éditeur de plateau avant de réunir le tout dans un menu très simple.

Une fois le jeu construit, nous pouvons nous attaquer à l'intelligence artificielle: les "bots". Afin d'établir l'infrastructure pour les bots plus complexes qui suivent, nous implémentons d'abord un bot "random" très simple qui ne joue qu'un coup au hasard. Puis, nous construisons le bot "1 layer" (easy), et "2 layer" (normal) qui tente de prédire le meilleur coup en simulant tous les coups possibles. Malheureusement, en essayant d'implémenter un bot "3 layer", qui regarde trois coups en avance avant de prendre sa décision, nous avons rencontré un problème majeur: il était beaucoup trop lent (.

Donc, après quelques recherches, nous avons décidé d'utiliser l'algorithme de Monte Carlo tree search (MCTS) comme cerveau pour le dernier bot. Malheureusement, après plusieurs heures de lecture, de programmation et de débug, nous avons réalisé que celui-ci était aussi trop lent quand exécuté avec un nombre élevé de simulation. Hexxagon étant un jeu avec beaucoup de coups possible, nous avons réalisé qu'aucun algorithme (relativement simple) ne pouvait produire un bot qui atteindrait nos attentes sans utiliser un réseau de neurones.

Enfin, au cours de la dernière semaine, nous avons refait toutes les interfaces et les sprites ainsi qu'ajouté les effets sonores et la musique. Le code terminé, nous avons écrit le rapport et la documentation ainsi que planifié, filmé et monté la vidéo.

> FONCTIONNEMENT ET OPÉRATIONNALITÉ :

Le projet est entièrement fonctionnel et dépasse largement nos attentes initiales mais ne reste cependant pas entièrement complet (il reste toujours des choses à améliorer en informatique). Pour l'instant, il contient un menu, un écran d'information, un éditeur de plateau, plusieurs bots, et bien sûr le jeu lui-même qui fonctionne (on espère) entièrement sans bugs.

Afin de nous assurer de l'absence de bugs dans notre programme, nous avons effectué beaucoup de tests sur les fonctions individuelles lors de leurs développements ainsi que des tests entiers du programme grâce à l'interface graphique.

Nous avons rencontré des dizaines de bugs (mineurs) lors du développement du programme qui ont pu être résolu soit par persévérance ou en demandant à notre coéquipier d'y jeter un coup d'œil (l'avantage de travailler à plusieurs). Malheureusement, un problème majeur reste sans solution pour l'instant. En effet, lors du développement des bots nous avons réalisé que notre plan original (regardé tous les coups possibles jusqu'à 3 coups dans le futur) n'était pas possible dû à des contraintes de temps d'exécution. Il a donc fallu dévier du plan original afin de trouver un algorithme plus rapide (le MCTS). Malheureusement celui- ci n'était à son tour pas assez efficace

> OUVERTURE :

Voici quelques idées d'améliorations que nous avons trouvé lors d'une session de brainstorming une fois le projet terminé.

Améliorations:

- Animations lors du déplacements des pions
- "Responsive design", permettant à l'utilisateur de changer la taille de la fenêtre de jeu
- Permettre à l'utilisateur de sauvegarder sur son disque des plateau de jeu afin de pouvoirs les réutiliser lorsque le programme est relancé ou même de les partager sur le web
- Interface web avec multijoueur et un système d'upload de tableau de jeu afin de pouvoir partager ses tableaux
- Création d'un neural network pour sauvegarder les parties simulé lors de l'exécution de l'algorithme de Monte Carlo et permettant donc éventuellement une executions bien plus rapide
- Création d'un neural network similaire a AlphaGo Zero (Deepmind) permettant d'entrainer un neural network en "self play"

Stratégie de diffusion :

Nous avons retenu 3 options principales pour finaliser et publier notre jeu:

1) rendre le jeu accessible en ligne grâce à la mise en place d'un site web
2) utiliser des outils comme Cython ou PyInstaller afin de générer un fichier exécutable qui ne requiert pas l'installation de python ou autre ressources afin d'être exécuté après installation. Puis, nous pourrons distribuer le jeu sur des plateformes telles que itch.io (plateforme de jeu "indie") ou même plus tard steam ou epic games.
3) Dû à son format assez simple et ses parties courtes, le jeu pourrait être adapté pour tourner sous android ou ios comme jeu mobile et publié sur leurs "game stores" respectifs.

Afin de diffuser notre jeu et le rendre connu d'un plus grand nombre de personnes possible, nous pouvons tout d'abord créer des comptes dédiés sur des réseaux sociaux comme instagram, twitter, reddit ou youtube. Sur ces plateformes, nous pourrons publier des "mèmes" sous forme de vidéos courtes, de GIFs ou de simples images, en rapport avec notre jeu et espérer obtenir une certaine viralité et donc apporter un grand nombre de joueurs. Avant la sortie du jeu au public, nous pouvons donner un accès limité à certains influenceurs sur des réseaux sociaux afin qu'ils puissent en faire une publicité en échange d'une compensation financière.

Analyse critique:

Notre équipe comportait initialement 4 membres mais malheureusement, 2 d'entre eux n'étaient pas assez investis dans le projet et ont dû être retirés au dernier moment. Nous aurions donc pu essayer de construire une équipe plus solide avec des membres davantage investis avant de se lancer dans ce projet (chronophage). Nous aurions aussi certainement pu commencer plus tôt afin de moins avoir a faire a la dernière minute.

De plus, le code peut probablement être optimisé dans plusieurs endroits notamment dans la partie collecte du score (pour l'instant, le score est calculé en bouclant a travers chaque cellule du plateau, pas très efficace !). Nous aurions aussi pu faire utilisation de la programmation orientée objet (même si celle ci n'est pas au programme de première) et ainsi mieux séparer notre code dans plusieurs fichiers plus facile à digérer à la lecture. Finalement, nous aurions aussi pu éviter d'utiliser des variables globales puisque celles ci ne sont pas recommandées et peuvent facilement causer des problèmes inattendu quand utilisé dans des projets plus large.

DOCUMENTATION

• Spécifications fonctionnelles (guide d'utilisation, déroulé des étapes d'exécution, description des fonctionnalités et des paramètres)
• Spécifications techniques (architecture, langages et bibliothèques utilisés, matériel, choix techniques, format de stockage des données, etc)
• Illustrations, captures d'écran, etc

Prérequis

- python >= 3.6 (ref: Python)
    - (note: le programme ne fonctionnera pas sur une version de python installé depuis le Microsoft Store à cause d'un bug avec le module mixer de pygame (il manque un dll))
- pygame 2.3.0 (ref: pygame)

Guide d' utilisation

Pour télécharger le jeu, vous pouvez le récupérer sur GitHub, directement sur le site ou avec l'outil de commande git et la commande suivante:

- git clone https://github.com/lonestar125/PyHexx.git

Pygame peut être facilement installé grâce à l'outil pip avec la commande "pip install pygame" ou "pip install -r requirements.txt" une fois que vous vous êtes situé dans le répertoire grâce à la commande cd

- cd \Users\<user>\Downloads\PyHexx
- pip install -r requirements.txt

Finalement lancez le fichier "pyhexx.py" et amusez-vous !

Détails d'exécution

Le programme fonctionne essentiellement grâce à plusieurs boucles while qui s'exécutent pendant que le joueur est dans une certaine fenêtre du programme. Les boucles se trouvent dans la fonction main() du programme.
(certaines fonctions ne sont pas mentionnées dans le texte suivant)

boucle info:

- écrit continuellement le texte d'information sur l'écran
- vérifie continuellement si l'utilisateur appuie sur le bouton exit de la page ou sur le bouton menu et exécute le code nécessaire pour effectuer ces actions

boucle board editor:

- vérifie continuellement si l'utilisateur appuie sur le bouton exit de la page ou sur le bouton menu et exécute le code nécessaire pour effectuer ces actions
- si un clic gauche est effectué, la position de la souris est enregistrée et comparé aux positions de chaque cellule du tableau et de son image pour déterminer la case avec laquelle l'utilisateur souhaite interagire

- le “status” de la case est changé et mis à jour avec l’image correspondante
- si le bouton menu est appuyé, la validité du plateau est vérifiée grâce à la fonction has_valid_path (renvoie False si une case n’est pas connecté au reste du plateau) et get_score (pour vérifier que chaque joueur a au moins un pion sur le plateau) si toute les conditions sont remplies, le plateau est sauvegardé et l’utilisateur est renvoyé au menu principale ou il peut commencer une partie

boucle game:

- vérifie continuellement si l’utilisateur appuie sur le bouton exit de la page ou sur le bouton menu et exécute le code nécessaire pour effectuer ces actions
- si un clic gauche est effectué, la position de la souris est enregistrée et comparé aux positions de chaque cellule du tableau et de son image pour déterminer la case avec laquelle l’utilisateur souhaite interagire
- si la case en question contient un pions du joueur qui doit jouer, les fonctions check_cloneable et check_jumpable sont appelé pour construire une liste des cases sur lesquels il faut afficher un contour (“outline”)
- si la case en question contient un outline égale à 1, un pion de la couleur du joueur est ajouté sur la case et les case alentours sont mise a jours grâce à la fonction update_neighbours, si la case contient un outline égale à 2, le même processus est effectué mais, cette fois en enlevant le pion de la case initialement cliqué (pour effectuer un “jump”
- si une case avec un outline est cliqué, une fois le tour joué, la fonction end_turn est appelé pour recalculer le score, vérifier si le jeu est terminé et changer la variable current_player

boucle menu:

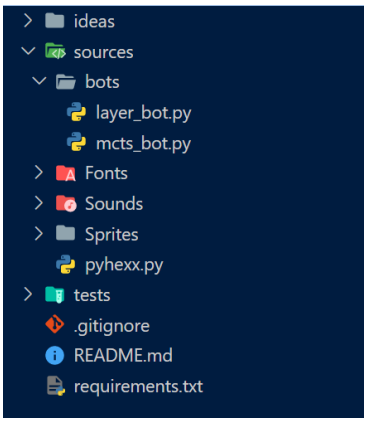
- vérifie continuellement si l’utilisateur appuie sur le bouton exit de la page ou sur le bouton menu et exécute le code nécessaire pour effectuer ces actions
- affiche les boutons et change les variables globales in_game, in_menu, in_info, in_board_editor si un boutons est cliqué ce qui engendre le départ d’une autre boucle

<u>L’algorithme de Monte Carlo</u>

L’algorithme de Monte Carlo tree search contient 4 étapes: la sélection, l’expansion, la simulation, et la “backpropagation” . Comme le nom l’indique, un arbre ou chaque noeud représente un état possible du plateau . L’algorithme consiste d’abord à sélectionner le nœud depuis lequel on veut simuler une partie, puis joué une partie au hasard puis enregistré si la partie a été gagnée ou perdue. Tout cela est répété des milliers ou même des millions de fois avant qu’un calcul soit effectué avec les donné récupéré afin de déterminer le meilleur coup

Architecture:

Notre programme finale (répertoire sources) est principalement contenu dans le fichier pyhexx.py mais fais appel à deux autres fichiers pythons: bots/layer_bot.py et bots/mcts_bot.py qui contiennent le code pour les bots “normal” et “mcts”



Tous les fichiers audios sont libre d’accès et on été obtenus sur freesound

La police utilisée a été obtenu sur fontspace