PAF 2016-2017

Rapport du Projet d'Application Final
Test de Turing pour le générateur poïétique

LELUC Rémi

ROBIN Clément

TALLEC Gauthier

TEBOUL Raphaël

Encadrant : DESSALES Jean-Louis

Jeudi 29 Juin 2017

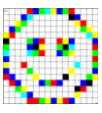
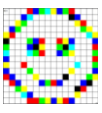


Table des matières



1. Résumé du sujet

Le Générateur poïétique est une œuvre d'art télématique, précurseur de nombreux jeux et réseaux sociaux sur Internet, imaginée par Olivier Auber en 1986 et développée en tant qu'œuvre d'art libre depuis 1987. Cette invention d'Olivier Auber est une technique de dessin collaboratif. Chaque joueur se voit attribuer une zone de dessin sur laquelle il dessine ce qu'il souhaite et il voit simultanément le dessin global.

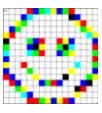
L'objectif de notre projet est de réaliser un test de Turing pour le générateur poïétique. Pour y parvenir, nous créons une interface de dessin multijoueur sous Python. Nous travaillons pour commencer avec des agents artificiels dont le comportement vise à imiter celui d'un humain. La stratégie de dessin des agents artificiels consiste à minimiser successivement la complexité globale du dessin et la complexité de sa propre sous-zone en vertu du principe de maximisation de l'inattendu.

On observe généralement des phénomènes émergents. Au départ, chacun découvre les possibilités d'action sur sa zone. Puis les utilisateurs découvrent des régularités structurelles entre leur création et une autre zone, souvent une zone adjacente. On observe alors une simplification du tableau global. Lorsque la "logique" globale devient trop simple au goût de certains, ils se plaisent à réintroduire de la diversité.

Ce projet a plusieurs objectifs liés :

- Réaliser un générateur poïétique simplifié, offrant par exemple la possibilité de faire des figures simples (traits, polygones réguliers, disques).
- Réaliser un agent capable de tenir son rôle. L'idée du test de Turing, dans ce contexte, est que les joueurs humains ne puissent pas déceler que l'un des joueurs est artificiel.
- Réaliser des parties dans lesquelles tous les joueurs sont artificiels.

Pour réaliser des agents artificiels capables d'innover, nous nous sommes inspiré de la théorie de la complexité de Kolmogorov (K-complexité). Les agents cherchent à maximiser l'inattendu structurel, c'est-à-dire qu'ils vont chercher à diminuer la K-complexité. Dans une situation complexe (proche de l'aléatoire), ils recherchent une simplification en augmentant la structure. Un moyen pour cela est d'imiter d'autres zones de l'écran. Si la structure se simplifie trop, ils recherchent la simplicité en créant une exception. On s'attend ainsi à des oscillations de la K-complexité.



2. English Summary

The Poietic Generator is a telematic work of art, a forerunner of many Internet games and social networks, conceived by Olivier Auber in 1986 and developed as a free art work since 1987. This invention of Olivier Auber is a technique of collaborative drawing. Each player is assigned a drawing area on which he draws what he wants and simultaneously sees the overall drawing.

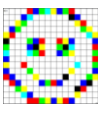
The objective of our project is to perform a Turing test for the poietic generator. To do so, we create a multiplayer drawing interface using Python. We created artificial agents whose behavior is intended to imitate that of a human. The strategy of drawing artificial agents consists in successively minimizing the overall complexity of the drawing and the complexity of its own subzone following the maximization principle of the unexpected.

Emerging phenomena are generally observed. Initially, everyone discovers the possibilities of action in their area. Then users discover structural patterns between their creation and another area, often an adjacent area. A simplification of the overall picture is then observed. When the global "logic" becomes too simple to the taste of some, they like to reintroduce diversity.

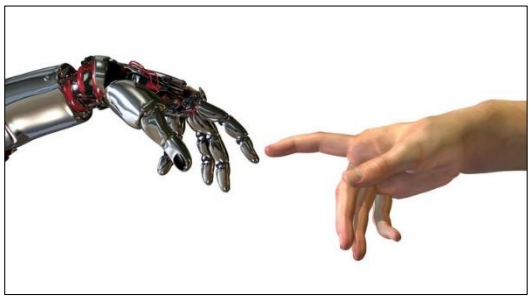
This project has several objectives:

- Create a simplified poetic generator, offering for example the possibility of making simple figures (lines, regular polygons, disks).
- Make an agent capable of holding his role. The idea of the Turing test, in this context, is that human players cannot detect that one of the players is artificial.
- Realize games in which all players are artificial.

In order to realize artificial agents capable of innovating, we were inspired by Kolmogorov's theory of complexity (K-complexity). The agents try to maximize the structural unexpected, that is to say that they will seek to diminish the K-complexity. In a complex situation (close to the random), they seek a simplification by increasing the structure. One way to do this is to mimic other areas of the screen. If the structure is simplified too much, they look for simplicity by creating an exception. We thus expect oscillations of K-complexity.

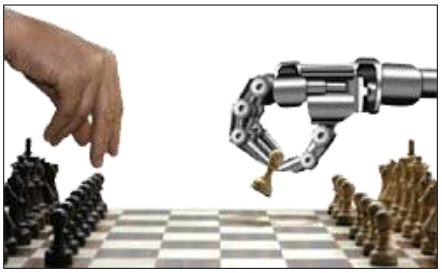


3. Introduction et problématique

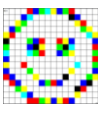


Une machine peut-elle nous surprendre ? Telle est la question de notre projet. Nous avons modélisé le générateur poïétique dans la perspective d'effectuer un test de Turing. Il s'agit de réaliser une partie de jeu entre plusieurs joueurs humains et une machine puis demander aux humains de déceler la machine.

La procédure de décision des pixels à modifier par un robot s'inspire de la théorie de la simplicité. A chaque tour de jeu, les machines choisissent un pixel en vertu du principe de maximisation de l'inattendu. Pour cela, une machine est capable d'effectuer de la reconnaissance de forme puis de prendre une décision afin de diminuer la complexité au sens de Kolmogorov. Cependant, afin de passer le test de Turing, un agent artificiel ne doit pas être en mesure de pouvoir détecter des symboles là où les humains ne les voient pas. L'implémentation de la reconnaissance de forme au sein des machines doit limiter leur *look-ahead* afin de paraître naturelle. Il est donc nécessaire de trouver un compromis lors de l'évaluation de la complexité pour décider de la forme à faire émerger et choisir le bon pixel en conséquence.



Le problème qui se pose est donc de trouver une méthode d'évaluation de la complexité d'un dessin. Pour simplifier, nous commençons par contraindre les possibilités de dessin. En effet, pour notre preuve de concept, nous avons d'abord choisi de faire émerger un seul motif simple en noir et blanc. Il s'agit d'un smiley, qui peut éventuellement être dans différentes configurations. Nous avons pour cela considéré des rotations de symboles. Puis nous avons ajouté la possibilité de jouer sur la couleur des pixels à modifier en adaptant notre reconnaissance des symboles avec des masques de convolution et de filtrage.

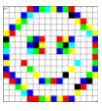


4. Etat de l’art

Le jeu défini par le Générateur poïétique se déroule à l’intérieur d’une matrice à deux dimensions et son principe s'inspire de celui du jeu de la vie et des cadavres exquis des surréalistes. Des joueurs humains contrôlent en temps réel les éléments graphiques de la matrice globale, à raison d'une unité par personne. Contrairement au cadavres exquis dans lesquels il y a toujours des parties cachées, ici toutes les actions des joueurs sont visibles en permanence par chacun d’eux. Il n’y a pas de notion de gagnant ou de perdant, le but du jeu étant simplement de faire apparaître collectivement des formes reconnaissables par tous et d’observer ensemble comment elles se créent. En l'absence de toute instruction, les joueurs commencent à créer ce qui pourrait ressembler à un motif aléatoire à distance. Mais le collectif tend à s'auto-organiser peu à peu avec des structures émergeant de temps en temps que ce soit localement ou bien globalement.

L'appellation « Générateur poïétique », qui dérive du concept d'autopoïèse en sciences du vivant, et de celui de poïétique en philosophie de l'art, traduit le processus d’auto-organisation à l’œuvre dans l'émergence continue de l'image globale. Depuis son origine, le Générateur poïétique a été conçu par son auteur comme un élément d'une recherche-action plus vaste en vue de créer un « art de la vitesse ». Le Générateur poïétique se positionne comme une invitation à un questionnement sur les processus d’interaction sociale, notamment lorsqu’ils sont médiés par des dispositifs technologiques interagissant avec les réseaux sociaux. Pour Olivier Auber, le Générateur poïétique, en tant que modèle et expérience accessibles à tous, pourrait contribuer à « une certaine connaissance conceptuelle de la manière dont la doxa se forme et s'exerce sur nous, en particulier à travers la technologie ».

Olivier Auber développe depuis le début des années 1980 des installations et expositions utilisant de multiples technologies dans la perspective de réaliser des sortes de miroirs des comportements. Il expérimente le Générateur Poïétique, le plus emblématique de ces miroirs, depuis 1986 sur différents réseaux. Il a fondé en 1997, avec l'architecte et urbaniste Bernd Hoge, le laboratoire culturel A+H (http://km2.net) qui conduit des projets interdisciplinaires entre territoires physiques et numériques.



En 1988, le Générateur Poïétique d'Olivier Auber fonctionnait dans une version centralisée, grâce à un serveur. Ce serveur peut être vu comme analogue au point de fuite spatial de la Renaissance. En effet, les informations émises par les utilisateurs convergeaient vers lui, et il les réexpédiait immédiatement vers les utilisateurs sous la forme d'une image globale réactualisée.

En 1995, Olivier Auber a modifié quelque peu sa conception de la perspective temporelle, lorsqu'il a eu accès, grâce au support de l'ENST, à une infrastructure Internet expérimentale appelée leMulticastBackbone, ou Mbone, qui relie divers centres de recherche en télécommunication de par le monde. Ce réseau qui préfigurait la nouvelle norme de l'Internet appelée IPv.6 prochainement mise en place globalement sur l'Internet, permettait théoriquement de réaliser une commutation "tous-tous" a-centrée et donc potentiellement de s'affranchir d'un serveur central pour mettre en œuvre le Générateur Poïétique. Et c'est ce qu'il a réalisé. Sur le plan de la dynamique de l'image et des règles d'interaction internes qui la sous-tendent, rien de nouveau, mais tout avait changé sur le plan de la structure réticulaire, et donc potentiellement sur celui des règles externes.

EXPERIMENTATION DU GENERATEUR POÏETIQUE DANS UNE CLASSE A BRUXELLES (2013)

Notre projet offre donc un nouveau jour au générateur poïétique. En effet, jusqu'alors ce jeu permettait de modéliser et d'expérimenter des théories sur les sciences cognitives et artistiques dans le rapport des hommes aux machines et à la simplicité. Nous avons décidé d'utiliser cet outil pour réaliser un test de Turing, l'objectif ici étant de déceler l'agent artificiel parmi les joueurs humains.

5. Organisation/Architecture du projet

Notre projet s'articule autour de sept grandes classes dont nous expliquons ici les fonctions principales et les méthodes associées :

La Classe **GameModel** : cette classe récupère les informations de jeu en provenance de la gameMatrix et les envoie à l'interface graphique, il relaie également les messages de l'interface graphique vers la gameMatrix

La Classe **GameMatrix** : cette classe est en lien entre le GameController et GameModel. Elle a pour but de représenter la matrice de jeu avec des pixels et conserver les mises à jour de cette dernière. L'un de ses rôles principaux est la conversion des coordonnées réelles envoyées par le gameModel en coordonnées virtuelles en termes de zone et de position au sein de la zone

La Classe **GameController** : cette classe a pour but d'organiser et gérer les tours des joueurs et l'organisation du jeu. Elle permet aussi d'initialiser le jeu.

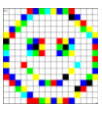
La Classe **Player** : c'est une classe abstraite qui va contenir les deux types de joueur (Human et Robot). Elle contient donc toutes les méthodes nécessaires aux deux joueurs ainsi que le constructeur qui est globalement commun entre les deux.

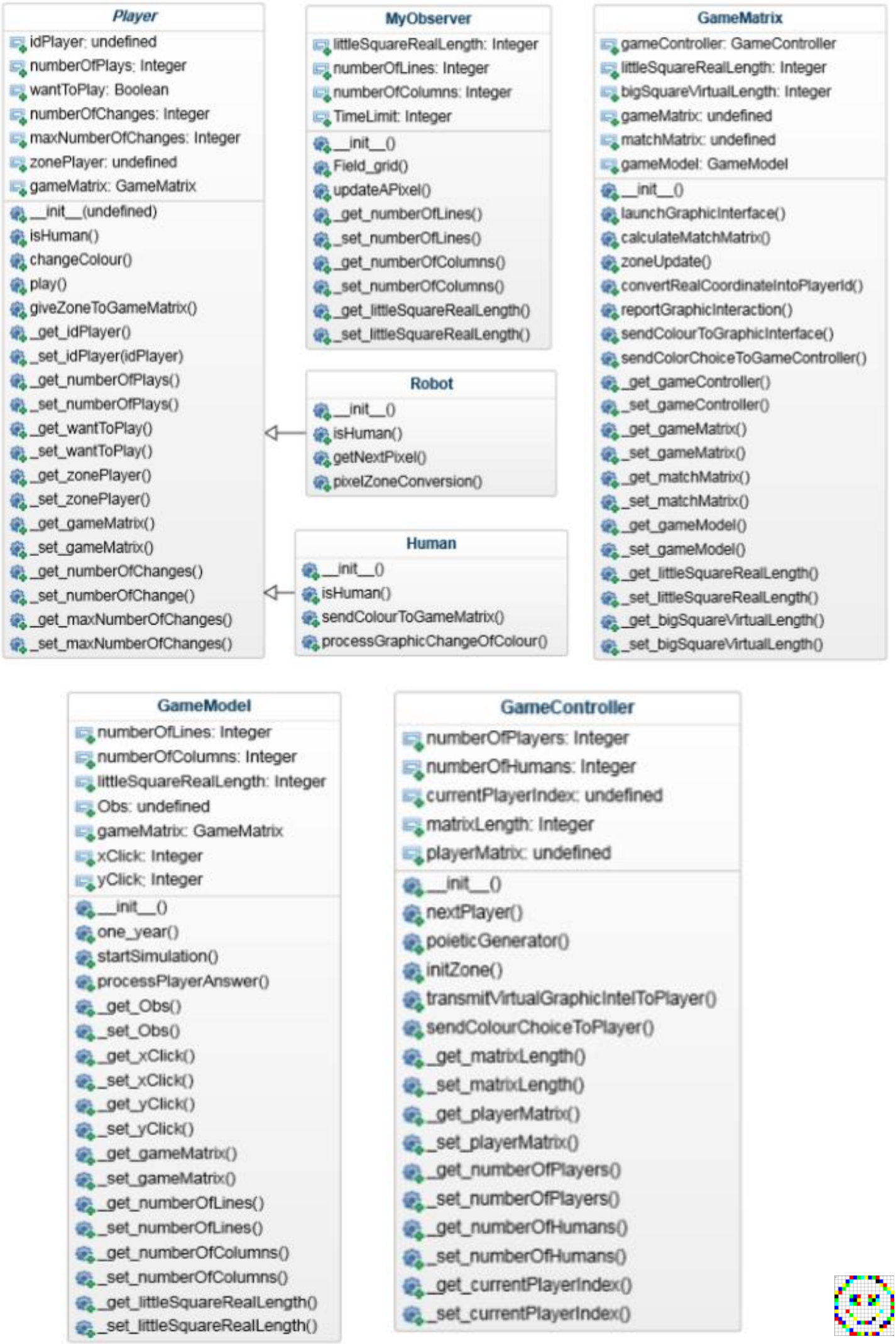
La Classe **Human** : cette classe représente les joueurs humains.

La Classe **Robot** : cette classe représente les joueurs robots, elle contient donc les méthodes de décision pour modifier la zone de pixel du joueur.

La Classe **MyObserver** : cette classe permet de détecter les changements de l'interface graphique que les joueurs humains peuvent changer.

Ci-dessous on trouve un diagramme d'architecture du projet avec les noms des attributs et méthodes de chaque classe.




Player
idPlayer: undefined
numberOfPlays: Integer
wantToPlay: Boolean
numberOfChanges: Integer
maxNumberOfChanges: Integer
zonePlayer: undefined
gameMatrix: GameMatrix
__init__(undefined)
isHuman()
changeColour()
play()
giveZoneToGameMatrix()
_get_idPlayer()
_set_idPlayer(idPlayer)
_get_numberOfPlays()
_set_numberOfPlays()
_get_wantToPlay()
_set_wantToPlay()
_get_zonePlayer()
_set_zonePlayer()
_get_gameMatrix()
_set_gameMatrix()
_get_numberOfChanges()
_set_numberOfChange()
_get_maxNumberOfChanges()
_set_maxNumberOfChanges()
MyObserver
littleSquareRealLength: Integer
numberOfLines: Integer
numberOfColumns: Integer
TimeLimit: Integer
__init__()
Field_grid()
updateAPixel()
_get_numberOfLines()
_set_numberOfLines()
_get_numberOfColumns()
_set_numberOfColumns()
_get_littleSquareRealLength()
_set_littleSquareRealLength()
Robot
__init__()
isHuman()
getNextPixel()
pixelZoneConversion()
Human
__init__()
isHuman()
sendColourToGameMatrix()
processGraphicChangeOfColour()
GameMatrix
gameController: GameController
littleSquareRealLength: Integer
bigSquareVirtualLength: Integer
gameMatrix: undefined
matchMatrix: undefined
gameModel: GameModel
__init__()
launchGraphicInterface()
calculateMatchMatrix()
zoneUpdate()
convertRealCoordinateIntoPlayerId()
reportGraphicInteraction()
sendColourToGraphicInterface()
sendColorChoiceToGameController()
_get_gameController()
_set_gameController()
_get_gameMatrix()
_set_gameMatrix()
_get_matchMatrix()
_set_matchMatrix()
_get_gameModel()
_set_gameModel()
_get_littleSquareRealLength()
_set_littleSquareRealLength()
_get_bigSquareVirtualLength()
_set_bigSquareVirtualLength()
GameModel
numberOfLines: Integer
numberOfColumns: Integer
littleSquareRealLength: Integer
Obs: undefined
gameMatrix: GameMatrix
xClick: Integer
yClick: Integer
__init__()
one_year()
startSimulation()
processPlayerAnswer()
_get_Obs()
_set_Obs()
_get_xClick()
_set_xClick()
_get_yClick()
_set_yClick()
_get_gameMatrix()
_set_gameMatrix()
_get_numberOfLines()
_set_numberOfLines()
_get_numberOfColumns()
_set_numberOfColumns()
_get_littleSquareRealLength()
_set_littleSquareRealLength()
GameController
numberOfPlayers: Integer
numberOfHumans: Integer
currentPlayerIndex: undefined
matrixLength: Integer
playerMatrix: undefined
__init__()
nextPlayer()
poieticGenerator()
initZone()
transmitVirtualGraphicIntelToPlayer()
sendColourChoiceToPlayer()
_get_matrixLength()
_set_matrixLength()
_get_playerMatrix()
_set_playerMatrix()
_get_numberOfPlayers()
_set_numberOfPlayers()
_get_numberOfHumans()
_set_numberOfHumans()
_get_currentPlayerIndex()
_set_currentPlayerIndex()

6. Répartition des modules

Pour mener à bien notre projet, nous nous sommes répartis le travail en différents modules que nous avons ensuite intégrés :

Reconnaissance de formes/Prise de décision : Raphaël et Rémi

Interface graphique (EvoLife) : Gauthier

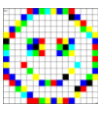
Architecture générale et intégrateur : Clément

7. Méthode et résultats

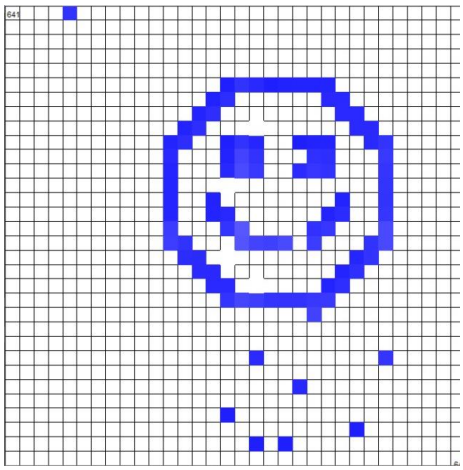
Nous avons entamé le travail par une réflexion profonde sur l'architecture du projet afin de s'assurer du plein respect du principe d'encapsulation. Par la suite nous nous sommes répartis le travail et c'est alors que les premiers vrais défis se sont posés. En effet, la prise de décision dans le dessin est complexe : avant de décider de dessiner un symbole ou une forme, il faut être capable de reconnaître un début parmi une matrice assimilable à du bruit. Comment reconnaître une forme en train d'être dessinée ? Quelle que soit sa taille, son inclinaison ou sa position dans l'espace ?

Pour ce faire, nous sommes allés demander au Dr Michel Roux du département Image de Télécom ParisTech afin de mettre au point un algorithme de reconnaissance de forme. Nous procédons ainsi : nous mesurons la corrélation pour tous les symboles différents afin de trouver le plus approchant. Cela est réaliser grâce à un filtrage par convolution, implémenté de manière efficace par transformation de Fourier. Notre hypothèse de départ est que l'inattendu repose sur la simplicité de la description du résultat donné. Nous testons donc la corrélation des symboles de manière à privilégier les symboles les plus simples à décrire. Cela se fait par une classification décroissante de la complexité au sein d'une liste de formes. Cela nous permet ensuite de choisir le pixel à allumer ou éteindre dans la zone du joueur (machine ou humain) considérée.

L'interface graphique a été réalisée sur EvoLife, une surcouche de Qt imposée par l'encadrant, dont il a fallu prendre en main l'environnement. D'une manière plus générale, nous avons travaillé sur l'ensemble du projet de manière incrémentale, en codant petit - à - petit chacune des fonctionnalités. Ainsi, dès le lundi 26 juin, nous avions une intelligence artificielle capable de dessiner à plusieurs des smileys en noir et blanc. La méthode getNextPixel() donnait les bonnes coordonnées pour dessiner un smiley codé en noir et blanc avec des 0 et des 1. Toutefois, il faut noter que l'interface graphique n'était alors pas prête. Par la suite, simultanément avec les finitions de l'interface graphique et l'intégration de la totalité du trajet, nous avons décidé de permettre aux joueurs de dessiner avec des couleurs différentes : ce qui a donné pour les robots une sur-couche leur permettant de choisir la couleur du pixel a allumé.



Et quels résultats avons-nous obtenu ? La théorie de la simplicité est efficace en ce qui concerne la reconnaissance de forme car notre projet est capable de reconnaître un smiley même avec un état de référence minimum (nous pouvons imposer un certain “horizon”, c’est-à-dire le nombre de coup d’avance que peut “voir” l’ordinateur). Nous avons de plus testé notre projet avec des humains lors de la journée de présentation, et des humains peuvent interagir avec l’ordinateur pour créer une image. Les réactions des joueurs humains ayant tenté l’expérience étaient très encourageantes. Ils ont fortement apprécié l’enjeu du projet et l’implémentation que nous avons réalisée. On trouve ci-contre le résultat obtenu pour 4 agents artificiels, chacun contrôlant 256 pixels.



Toutefois le choix de la couleur n’est pas encore pleinement satisfaisant, car nous avons mis en évidence un problème avec la théorie de la simplicité qui régit la prise de décision : c’est un point que nous développons dans le chapitre suivant.

8. Discussion

Si nous sommes parvenus à générer des smileys de tailles différentes apparaissant à des endroits différents et même générer plusieurs smileys sur une même matrice globale, il nous reste des points à améliorer sur le choix de la couleur pour les agents artificiels.

Selon la théorie de la simplicité, le plus simple à décrire - et donc le plus désirable -est un symbole d’une seule couleur. Il ne faut pas cependant remarquer qu’un joueur humain peut se plaire à utiliser une alternance régulière de plusieurs couleurs ou même des couleurs aléatoires, ce qui n’est pas admissible pour la théorie de la simplicité. Notre projet, voulant à terme pouvoir passer le test de Turing, il nous faudrait adapter l’algorithme de choix de la couleur afin de pouvoir mimer un comportement humain.

9. Conclusion et perspectives

Les résultats que nous avons obtenus sont très satisfaisants. Ils permettent de mettre en évidence et en application le principe de maximisation de l'inattendu évoqué dans la théorie de la simplicité. Dans cette perspective, le générateur poïétique que nous avons réalisé permet d'illustrer ce phénomène.

En ajoutant de nouvelles formes dans notre liste de symboles à détecter par les agents artificiels, nous pouvons sans grande difficulté adapter le jeu pour créer de nouvelles figures et faire émerger de nouveaux motifs.

Nous pouvons choisir le nombre total de joueurs ainsi que le nombre de robots et d'humains. Nous avons réalisé différentes parties avec de nombreuses configurations différentes. Nous avons pour cela joué sur les proportions du nombre de joueurs humains et robots Nous avons ainsi créé un nouveau scénario sur Evolife dans le répertoire Other puis dans le dossier PoieticGenerator. Ce module pourra être réutilisé par la suite à titre d'exemple de la théorie de la simplicité.

10. Bibliographie

- Le Générateur Poïétique, Auber, http://poietic-generator.net/
- Convolution Approach for Feature Detection in Topological Skeletons Obtained from Vascular Patterns - Martin Aastrup Olsen, https://www.fbi.h-da.de/fileadmin/gruppen/FG-IT-Sicherheit/Publikationen/2011/2011_04_Olsen_SSCI2011.pdf
- Role of Simplicity in Creative Behaviour: The Case of the Poietic Generator - Saillenfest, http://www.computationalcreativity.net/iccc2016/wp-content/uploads/2016/06/paper_39-1.pdf