

ECOLE POLYTECHNIQUE DE L'UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS

Département Informatique

64 avenue Jean Portalis

37200 Tours, France

Tél. +33 (0)2 47 36 14 14

polytech.univ-tours.fr

Projet Recherche & Développement 2016-2017

Jeux de Nim

**POLYTECH[®]**
TOURS

Tuteurs académiques

Jean-Charles BILLAUT

Carl ESSWEIN

Étudiant

Arthur CAPO (DI5)



Liste des intervenants

Nom	Email	Qualité
Arthur CAPO	arthur.capo@etu.univ-tours.fr	Étudiant DI5
Jean-Charles BILLAUT	jean-charles.billaut@univ-tours.fr	Tuteur académique, Département Informatique
Carl ESSWEIN	carl.esswein@univ-tours.fr	Tuteur académique, Département Informatique



Avertissement

Ce document a été rédigé par Arthur Capo susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Jean-Charles Billaut et Carl Esswein susnommés les tuteurs académiques.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnaît assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnaît qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable des tuteurs académiques et de l'entreprise.

L'auteur autorise l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.



Pour citer ce document

Arthur Capo, *Jeux de Nim*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2016-2017.

```
@mastersthesis{
  author={Capo, Arthur},
  title={Jeux de Nim},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2016-2017}
}
```

Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	iv
1 Introduction	1
1 Contexte	1
2 Objectifs	2
3 Bases méthodologiques	2
2 Description générale	3
1 Environnement du projet	3
2 Caractéristique des utilisateurs.....	3
3 Fonctionnalités du système	3
4 Structure générale du système	4
3 Etat de l'art - veille technologique	6
1 Introduction.....	6
2 Histoire et définition des jeux de Nim	6
2.1 Histoire	6
2.2 Définition des jeux impartiaux et des jeux de Nim	7
3 Exemples de trois de jeux de Nim : règles et stratégies gagnantes.....	7
3.1 Le jeu des allumettes	8

3.1.1	Présentation et règles	8
3.1.2	Stratégie gagnante	8
3.2	Le jeu de Marienbad	9
3.2.1	Présentation et règles	9
3.2.2	Stratégie gagnante	9
3.3	Le Tic-Tac-Nim.....	10
3.3.1	Présentation et règles	10
3.3.2	Stratégie gagnante	11
4	Généralisation de la théorie des jeux de Nim	11
4.1	Modélisation d'un jeu sous forme de graphe.....	12
4.1.1	Recherche du graphe	12
4.1.2	Propriétés de ce graphe	12
4.1.3	De la complexité à définir le graphe d'un jeu – Exemple du Tic-Tac-Nim	13
4.2	Recherche du noyau du graphe d'un jeu de Nim.....	14
4.2.1	Définition du noyau - exemple du jeu des allumettes.....	14
4.2.2	Déterminer le noyau d'un graphe orienté sans circuit	15
5	Conclusion	18
4	Analyse et conception	19
1	Premiers éléments à mettre en oeuvre.....	19
1.1	Mise en place d'un espace de projet.....	19
1.2	Mise en place de l'algorithme pour résoudre le Tic-Tac-Nim.....	19
1.3	Mise en place du plateau de jeu générique	20
2	Conception.....	20
2.1	Deuxième version du diagramme de classe	20
2.2	Respect des spécifications	21
2.3	Scènes.....	22
5	Mise en oeuvre	24
1	Outils et versions	24
2	Implémentations.....	24
2.1	Nouvelles fonctionnalités	24
2.1.1	Développement du jeu de Star-Nim	24
2.1.2	Statistiques du joueur	25
2.1.3	Comptes joueurs	25
2.2	Limites et faiblesses	26
3	Qualité et performance	26
3.1	Tests.....	26
3.2	Performances.....	27
3.3	Améliorations	27

6 Bilan et conclusion	29
Annexes	30
A Spécifications fonctionnelles et non fonctionnelles	31
1 Spécifications fonctionnelles.....	31
1.1 Choix du jeu.....	31
1.2 Personnaliser la partie	31
1.3 Jouer un coup.....	31
2 Spécifications non fonctionnelles.....	31
2.1 Contraintes de développement et conception.....	31
2.2 Contraintes de fonctionnement et d'exploitation : performances et capacités	32
B Présentation et résolution de plusieurs jeux de Nim	33
1 Jeu de Star Nim.....	33
1.0.1 Présentation et règles	33
1.0.2 Stratégie gagnante	34
2 Coin-Strip	35
2.0.1 Présentation et règles	35
2.0.2 Stratégie gagnante	36
3 Jeu de Hex.....	36
3.0.1 Présentation et règles	36
3.0.2 Stratégie gagnante	37
4 Jeu de Cram	37
4.0.1 Présentation et règles	37
4.0.2 Stratégie gagnante	38
C Comment ajouter son propre jeu de Nim dans l'application	39
D Gestion de projet	41
1 Plannings	41
2 Analyse de faisabilité.....	43
2.1 Spécifications de base	43
2.2 Evolution des objectifs.....	43
3 Analyse de risque	43
4 Suivi de projet.....	44
5 Outils utilisés pour la gestion de projet.....	44
Comptes rendus hebdomadaires	45

Table des figures

2 Description générale

- 1 Diagramme de classe de l'application..... 4
- 2 Diagramme de classe de l'application..... 5

3 Etat de l'art - veille technologique

- 1 Exemple d'une partie de jeu des allumettes 8
- 2 Exemple d'une partie de Tic-Tac-Nim 11
- 3 Quatre grilles équivalentes au Tic-Tac-Nim 11
- 4 Graphe du jeu des allumettes..... 12
- 5 Rotations et symétries d'une grille de Tic-Tac-Nim 13
- 6 Les 27 coups possibles sur une grille du Tic-Tac-Nim..... 14
- 7 Graphe du jeu des allumettes, avec mise en avant du noyau 15
- 8 Première étape de l'algorithme du recherche du noyau 16
- 9 Deuxième étape de l'algorithme du recherche du noyau 16
- 10 Dernière étape de l'algorithme du recherche du noyau 17

4 Analyse et conception

- 1 Diagramme UML final 20
- 2 Storyboard de l'application 22

5 Mise en oeuvre

- 1 Statistiques complètes d'un joueur 25
- 2 Profiler 27

B Présentation et résolution de plusieurs jeux de Nim

1	Plateau du jeu de Star Nim.....	33
2	Les 23 états du Star Nim	34
3	Le graphe du jeu de Star Nim.....	35
4	Exemple d'une partie de Coin-Strip	36
5	Ecart entre les pièces.....	36
6	Plateau d'un jeu de Hex 11×11	37
7	Exemple d'une partie de Cram sur un plateau 2×6	38
8	Symétrie des cases du jeu de Cram sur un plateau 2×6	38

D Gestion de projet

1	Diagramme de Gantt réel du S9.....	41
2	Diagramme de Gantt prévisionnel du S10	42
3	Diagramme de Gantt réel du S10.....	42

1

Introduction

Le Projet de Recherche & Développement (PRD), réalisé en 5^{ème} année, s'inscrit dans la formation dispensée à Polytech Tours dans le cadre du cursus d'ingénieur en informatique. Le PRD se déroule de la fin septembre à fin mars à raison de deux journées par semaine. Il donne lieu à la rédaction d'un cahier de spécifications, d'un état de l'art, d'un rapport et de deux soutenances.

Plusieurs sujets de PRD ont été proposés et mon choix s'est porté sur les jeux de Nim, où la MOA était représentée par Jean-Charles Billaut et Carl Esswein, enseignants chercheurs à Polytech Tours.

Le but de ce document est de formuler les spécifications fonctionnelles et techniques nécessaires à la réalisation du PRD. Il inclut également un état de l'art, ainsi qu'une analyse et conception du projet.

Le PRD est orienté vers le développement d'une application smartphone implémentant des jeux de Nim. Les acteurs du projet sont :

- La maîtrise d'oeuvre (MOE) : Arthur Capo, étudiant en 5^{me} année de l'Ecole d'Ingénieur Polytech Tours au sein du département Informatique.
- La maîtrise d'ouvrage (MOA) : Les encadrants du projet, Jean-Charles Billaut et Carl Esswein, de l'équipe pédagogique de Polytech Tours.

Ce document a été rédigé par Arthur Capo et il sera validé par les différents acteurs du projet.

1 Contexte

Les jeux de Nim sont une catégorie de jeux qui opposent deux joueurs. A tour de rôle, chaque joueur effectue un coup : retirer des allumettes, placer un ou plusieurs pions... La partie se termine lorsqu'il n'y a plus de coup possible, et il y a forcément un gagnant et un perdant (pas de match nul possible). Les différentes caractéristiques des jeux de Nim sont décrites à la section [2.2](#) (Chapitre 3) .

La résolution et la compréhension de la théorie de ces jeux fait appel à plusieurs outils de la recherche opérationnelle, notamment en ce qui concerne la manipulation de graphes. Ce projet s'inscrit dans une vulgarisation de certaines techniques et outils de ce domaine, sous la forme d'une application smartphone. Ce projet est proposé en complément de la fabrication d'un kit de jeu physique, contenant des plateaux et des pions, proposant plusieurs jeux de Nim, un

travail mené par Jean-Charles Billaut (MOA), dont la réalisation est en cours. L'idée est d'offrir la possibilité de jouer à des jeux de Nim sur deux supports : physique et smartphone.

2 Objectifs

L'objectif est donc de développer une application smartphone proposant de jouer à plusieurs jeux de Nim. Il offrira la possibilité de jouer contre une intelligence artificielle, et potentiellement contre un autre joueur (sur le même appareil).

Pour le moment, deux jeux sont à développer : le jeu de Coin-Strip (description à la section 2.0.1 (Annexe B)) et le Tic-Tac-Nim (description à la section 3.3.1 (Chapitre 3)).

Il est donc nécessaire de développer les algorithmes suffisants pour implémenter une intelligence artificielle capable de jouer le meilleur coup possible, l'IA pouvant être réglé selon 3 difficultés : facile (coup aléatoire), moyen (une chance sur deux entre un coup aléatoire et un coup parfait) et difficile (coup parfait).

On doit également être en mesure de fournir une interface cohérente permettant à l'utilisateur de jouer ces jeux de Nim.

3 Bases méthodologiques

Concernant la modélisation du système, on utilisera le langage UML, qui permet de définir les classes et composants du programme, un langage largement utilisé donc adapté ici.

Nous partons sur un rythme d'un rapport tous les jeudi soirs, dans lequel est décrit ce qui a été fait durant la semaine et ce qui est prévu pour la semaine à suivre.

2

Description générale

1 Environnement du projet

Ce projet ne se base pas sur une application existante. Cependant, nous allons nous inspirer et utiliser les différents travaux effectués sur les jeux de Nim. Les principales méthodes de résolution des jeux de Nim, quant à elles, existent déjà. On s'inscrit toujours dans un contexte de vulgarisation des algorithmes et outils faisant appels à la recherche opérationnelle.

2 Caractéristique des utilisateurs

Sur ce projet, il n'y a pas de type d'utilisateur défini, car on cherche à toucher le grand public. Cette application ne se destine pas à un utilisateur particulier et ne nécessite pas de compétences en informatique pour le manipuler.

L'application doit être donc simple et claire. On ne doit pas étouffer le joueur avec trop d'informations, mais il faut quand même afficher le strict minimum : on doit donc trouver un juste milieu pour que l'application soit intuitive à utiliser.

3 Fonctionnalités du système

L'application consistera en plusieurs écrans principaux : le menu d'accueil, le choix du jeu, la personnalisation du jeu, ainsi que le jeu en lui-même.

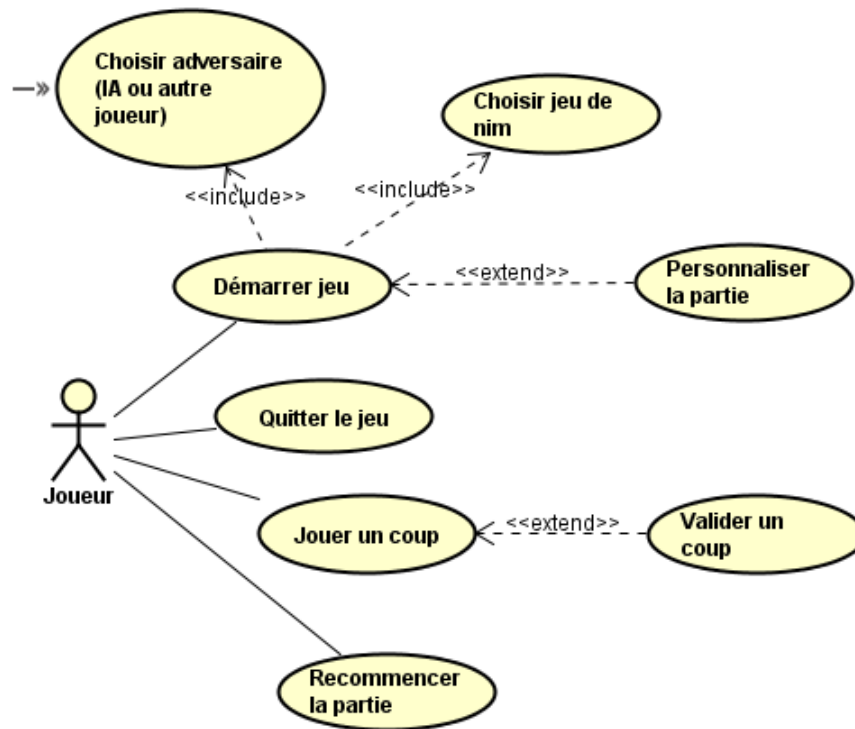


Figure 1 – Diagramme de classe de l'application

4 Structure générale du système

Le but est de modéliser une structure qui permet de s'adapter à la plupart des jeux de Nim. Nous allons partir d'une base la plus générique possible, qu'il faudra ensuite spécifier pour différents cas. Les jeux plus « complexes » nécessiteront sans doute des ajouts de fonctions ou de classe, mais elles utiliseront toutes l'architecture de base proposée.

On relève des éléments communs à tous ces jeux : il y a 2 joueurs (dont un peut être une intelligence artificielle), une partie (qui contiendra les éléments et informations du jeu en cours – le plateau du jeu notamment), dans laquelle on vérifie, à chaque coup joué, si l'un des joueurs a gagné... On vérifie également si le coup joué est autorisé.

Notre plateau est représenté par la classe abstraite `BaseGame`, dont hériteront nos plateaux réels, en fonction du jeu proposé. Chaque jeu de Nim peut être représenté par un graphe, celui-ci possédant un noyau (ensemble d'états clés, utiles pour la victoire du jeu).

3

Etat de l'art - veille technologique

1 Introduction

Les jeux de Nim sont considérés comme les premiers jeux existants au monde. Leur apparente simplicité cache en réalité des mathématiques et des théories qui ont été prouvées il y a une centaine d'années. Sans les progrès de la technologie et de l'informatique, il aurait d'ailleurs été long et compliqué d'en résoudre un certain nombre.

Nous allons d'abord présenter les jeux de Nim, puis en sélectionner trois dont les règles et la stratégie à adopter pour gagner sont diverses. Enfin, dans la dernière partie, nous prouverons que ces stratégies peuvent se résumer en la modélisation du jeu sous la forme d'un graphe, en abordant notamment la notion de noyau.

2 Histoire et définition des jeux de Nim

2.1 Histoire

Les origines ne sont pas vérifiées, mais les premières traces des jeux de Nim remonteraient en Chine Ancienne, ce qui en fait sans doute l'un des plus vieux jeu du monde. On trouverait également quelques origines en Europe, au XVIème siècle.

On doit le nom « Nim » à Charles L. Bouton (1869 – 1922), un mathématicien américain, professeur à l'université de Harvard. A nouveau, l'origine est incertaine : Nim peut être "win" à l'envers et nimm signifie "prendre" en allemand...

En 1940, lors de l'exposition universelle de New York, le physicien Edward Condon présente une machine, le Nimatron, émulant le jeu de Nim. En 1951, John Bennett propose, avec l'aide de l'entreprise Ferranti, une machine semblable, qui deviendra célèbre et est considérée comme l'un des premiers jeux informatisés existant.

Les jeux de Nim ont été également popularisé par un film français, L'année Dernière à Marienbad, sorti en 1961, où l'on voit deux protagonistes jouer à une variante du jeu de Nim classique, qu'on appelle désormais le jeu de Marienbad.

2.2 Définition des jeux impartiaux et des jeux de Nim

Les jeux de Nim font partie d'une catégorie de jeu qu'on appelle les jeux impartiaux. Ces jeux réunissent les critères suivants :

- Une partie réunit 2 joueurs, qui s'opposent à tour de rôle.
- Les 2 joueurs ont connaissance de l'état de la partie et toutes ses composantes et ce, à tout moment du jeu (rien n'est dissimulé).
- Le hasard n'intervient pas pendant le jeu
- Les mouvements disponibles à partir d'une position donnée sont les mêmes pour les deux joueurs. La seule différence fondamentale entre le joueur 1 et le joueur 2 est que le joueur 1 commence à jouer en premier.
- La partie se termine lorsqu'un joueur ne peut plus jouer et le joueur qui joue le dernier coup gagne la partie. Il n'y a donc pas de match nul possible. Il existe également une variante des jeux de Nim, qu'on nomme « misère » : dans ce cas-là, c'est le joueur qui effectue le dernier coup possible qui perd la partie.

Voici une liste non exhaustive de jeux connus qui n'appartiennent pas à la catégorie des jeux impartiaux :

Jeu	Pourquoi ce jeu n'est pas impartial ?
Bataille navale	Les 2 joueurs n'ont pas connaissance de toutes les composantes du jeu (placement des bateaux de l'adversaire)
Echecs	Les mouvements disponibles ne sont pas les mêmes pour les deux joueurs, notion de pions blancs et noirs
Tic-Tac-Toe	Possibilité de match nul
Jeux de carte (poker, tarot)	Intervention du hasard (lors de la distribution), pas de connaissance des cartes de l'adversaire, coups disponibles différents selon le joueur, implique souvent plus de 2 joueurs...

On considère donc qu'à partir du moment où un jeu respecte les conditions énoncées ci-dessus, c'est un jeu de Nim. Par conséquent, au vu de la relative simplicité de ces règles, il en existe un grand nombre et il est aisé d'en inventer des originaux. Nous allons désormais en présenter trois, en détaillant les règles de chacun et en expliquant quelle est la stratégie à adopter pour gagner une partie à coup sûr.

3 Exemples de trois de jeux de Nim : règles et stratégies gagnantes

Trouver une stratégie gagnante, c'est déterminer mathématiquement et/ou visuellement une suite de coups amenant vers une victoire assurée. Nous allons voir qu'il existe plusieurs méthodes pour la trouver et que la forme et l'essence de ces stratégies gagnantes peuvent être différentes d'un jeu à l'autre.

3.1 Le jeu des allumettes

3.1.1 Présentation et règles

Le jeu des allumettes, sans doute le plus commun et le plus ancien, consiste en un tas de N allumettes (ou des pierres - tout objet capable d'être retiré). Chaque joueur retire tour à tour de 1 à M allumette(s). Le joueur qui retire la dernière a gagné.

Pour illustrer simplement ce jeu, nous allons choisir un tas de $N = 12$ allumettes et le joueur peut en retirer jusqu'à $M = 3$.

Voici l'exemple d'une partie :

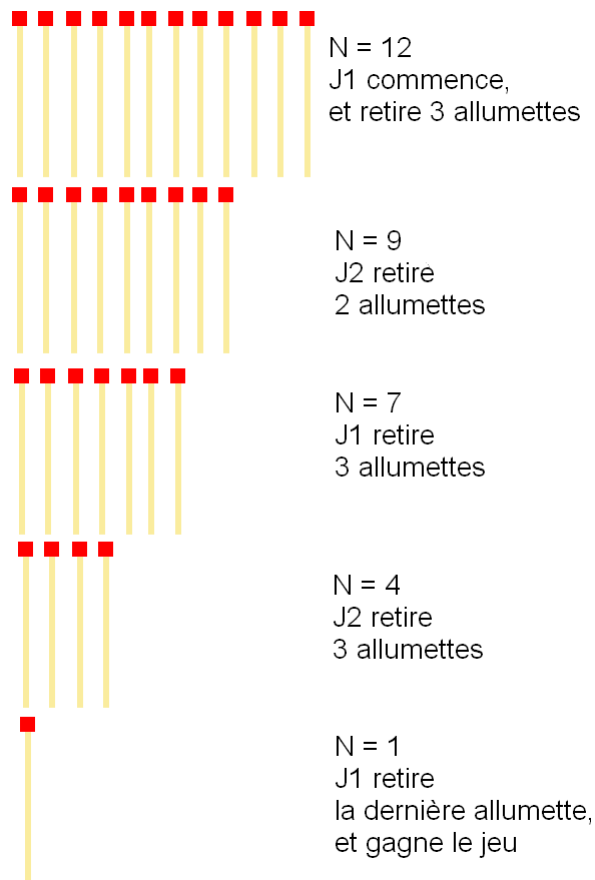


Figure 1 – Exemple d'une partie de jeu des allumettes

3.1.2 Stratégie gagnante

On peut définir la stratégie à adopter de manière naïve, en partant de la situation suivante. Supposons que ce soit le tour de l'adversaire de jouer. S'il lui reste 1, 2, ou 3 allumettes, alors on est sûr qu'il va gagner puisqu'il est dans la capacité de toutes les retirer. En revanche, s'il lui reste 4 allumettes, on est sûr qu'il nous restera 3, 2, ou 1 allumette(s) après son coup, qu'on pourra ensuite toutes retirer.

S'il reste 5 allumettes à l'adversaire, alors il a tout intérêt à ne retirer qu'une seule allumette : il nous en restera 4, et on se situera dans la situation précédente, qui ne nous est pas favorable. On

peut étendre la situation jusqu'aux 12 allumettes et on remarque très rapidement qu'on peut toujours gagner lorsqu'on laisse 4, 8 ou 12 allumettes à l'adversaire.

Avec les valeurs $N = 12$ et de $M = 3$, il est préférable pour un joueur de laisser son adversaire commencer, car ce premier peut toujours se ramener à l'une des situations que l'on a énoncées.

Finalement, on peut résoudre le problème pour toutes valeurs de N et M . On va chercher à laisser à l'adversaire $A \times (M + 1)$ allumettes, avec $A > 0$. Concernant le choix de commencer la partie ou non, on regarde si $N \div (M + 1) = 0$. Si c'est le cas, alors on a tout intérêt à laisser l'adversaire commencer, sinon il serait judicieux de commencer à jouer.

3.2 Le jeu de Marienbad

3.2.1 Présentation et règles

Le jeu de Marienbad consiste en 4 tas d'allumettes, avec respectivement 1, 3, 5 et 7 allumettes par tas. Chaque joueur peut retirer tour à tour autant d'allumettes qu'il veut (au moins une), mais dans un seul tas à la fois. Le joueur qui est forcé de retirer la dernière allumette perd : son adversaire a gagné la partie.

Ce jeu est une spécification du jeu de Nim classique. Dans celui-ci, la situation initiale peut être diverse : on peut avoir un nombre de tas et d'allumettes aléatoires. De plus, dans le jeu classique, c'est le joueur qui retire la dernière allumette qui gagne, tandis que c'est l'inverse pour le jeu de Marienbad : celui-ci est donc une version misère (cf Section 2.2)

3.2.2 Stratégie gagnante

Pour la résolution de ce jeu, nous allons travailler sur la situation initiale de Marienbad, qui propose 4 tas de 1, 3, 5 et 7 allumettes. Cependant, nous allons utiliser la version classique et non pas misère : le joueur qui prend la dernière allumette a donc gagné.

La stratégie gagnante de ce jeu a été énoncée par Charles Bouton, en 1901, dans son article Nim, A Game with a complete mathematical Theory [1]. Il a notamment prouvé le théorème suivant :

- Le joueur qui effectue le premier coup possède une stratégie gagnante si et seulement si la somme de Nim des tas est différente de zéro. Sinon, le second joueur possède une stratégie gagnante.

Bouton définit la somme de Nim de manière suivante :

- Soient a_1 et $a_2 \in \mathbb{N}$. La somme de Nim de 2 entiers, notée $a_1 \oplus a_2$, correspond à l'opération binaire Ou Exclusif ($a_1 \text{ XOR } a_2$), qui suit la table de vérité suivante :

XOR	0	1
0	0	1
1	1	0

Par exemple, avec $A = 3 = (011)$ et $B = 5 = (101)$, on a :

$$A \oplus B = 5 \oplus 3 = ((0 \text{ XOR } 1)(1 \text{ XOR } 0)(1 \text{ XOR } 1)) = (110) = 6 \quad (1)$$

L'opération XOR est commutative. Appliquons la somme de Nim à la situation initiale du jeu :

7	1	1	1
5	1	0	1
3	0	1	1
1	0	0	1
<hr/>			
	0	0	0

Le joueur qui commence la partie ne possède pas de stratégie gagnante, puisque la somme de Nim est égale à 0. C'est donc le cas de celui qui jouera en deuxième.

Maintenant, on va déterminer quel nombre d'allumettes retirer à chaque tour pour gagner à coup sûr. Dans son article, Bouton énonce les lemmes suivants.

Soit S la somme de Nim des tas restants :

- Si $S = 0$, alors, après n'importe quel mouvement d'un joueur, $S \neq 0$ [1]
- Si $S \neq 0$, alors il existe un mouvement tel que, après ce mouvement, on ait $S = 0$ [2]

Supposons que l'on soit dans le cas initial, avec des tas de 1, 3, 5 et 7 allumettes ($S = 0$) et laissons l'adversaire commencer. Après son coup, la somme de Nim des tas sera forcément différente de 0 (d'après [1]). On est ensuite dans la possibilité de retirer un certain nombre d'allumettes pour obtenir une somme de Nim égale à 0 (d'après [2]).

On répète chacune de ces étapes jusqu'à arriver au cas final où il n'y a plus d'allumettes dans chacun des tas ($S = 0$).

Contrairement au jeu des allumettes, on ne peut pas visuellement déterminer la stratégie gagnante du jeu de Nim classique. Cependant, on peut se souvenir du comportement à adopter dans certains cas spécifiques. On suppose les situations suivantes avec différents tas de sorte à ce que la somme de Nim soit différente de 0 :

- Si l'on a 2 tas, avec respectivement N_1 et N_2 allumettes et $N_1 > N_2$, alors on retire $N_1 - N_2$ allumettes au tas 1. On a ainsi 2 tas de N_2 allumettes.
- Si l'on a 3 tas, dont 2 de même taille, alors on retire toutes les allumettes du tas de la taille différente des 2 autres.
- Si on a 4 tas, dont 2 de même taille, alors on va faire en sorte qu'il y ait autant d'allumettes dans le 3ème et le 4ème tas.

Pour les autres cas, on doit passer par le calcul de la somme de Nim et essayer de modifier un tas pour obtenir une somme égale à 0.

3.3 Le Tic-Tac-Nim

3.3.1 Présentation et règles

Le jeu du Tic-Tac-Nim, dont le principe est largement inspiré du Tic-Tac-Toe, consiste en une grille de 3x3 dont le but est de placer des croix de sorte à ce que l'on soit le dernier joueur à jouer : la grille sera donc remplie et plus aucun coup n'est possible.

Cependant, contrairement au Tic-Tac-Toe classique, il est possible de placer de 1 à 3 croix sur une grille, à partir du moment où elles se suivent horizontalement ou verticalement. Voici l'exemple d'une partie :

Ici, c'est le joueur 1 qui gagne, puisqu'il a effectué le dernier coup possible.

On peut bien entendu étendre le jeu à une grille de taille $N \times M$ (avec, de préférence, $N \geq 3$ et $M \geq 3$, sinon le jeu aurait un intérêt limité).

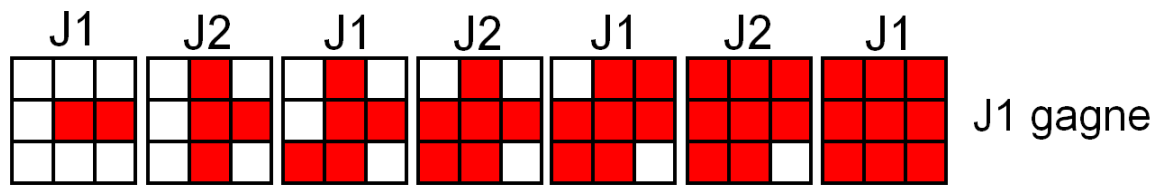


Figure 2 – Exemple d'une partie de Tic-Tac-Nim

3.3.2 Stratégie gagnante

Contrairement au jeu des allumettes et au jeu de Nim classique, on ne peut pas définir une stratégie gagnante de manière naïve, ni de manière calculatoire.

En réalité, il existe un ensemble de grilles appartenant à une catégorie de positions sûres. Ces positions sont soumises à une propriété : si, à n'importe quel instant du jeu, on trouve un mouvement pour obtenir une des grilles appartenant à cet ensemble, alors il existe une série de mouvements permettant d'arriver jusqu'à la grille finale et ce quel que soit la réplique de l'adversaire. Cependant, si la grille actuelle est déjà une position sûre, alors c'est l'adversaire qui est dans la capacité d'effectuer cette série de coups.

Il existe 28 grilles dites "sûres". Cependant, comme le jeu de Tic-Tac-Nim est un jeu de plateau, une grille peut être équivalente à une autre, car il existe des rotations et symétries.

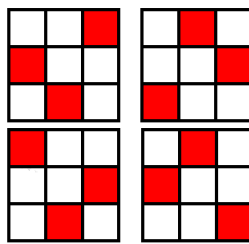


Figure 3 – Quatre grilles équivalentes au Tic-Tac-Nim

Du coup, il existe visuellement beaucoup plus de 28 grilles dans cet ensemble de positions sûres.

Bien entendu, la grille finale appartient à cet ensemble, puisqu'une fois qu'on l'a atteinte, on a gagné la partie. On sait que la grille initiale (vide) appartient également à cet ensemble : il est donc idéal de laisser l'adversaire commencer, d'après la propriété énoncée précédemment.

Trouver cet ensemble de grilles n'a été possible qu'en énumérant toutes les combinaisons possibles de la grille du Tic-Tac-Nim. Effectuer cette liste à la main est fastidieux, puisqu'il en existe un grand nombre (512). On a donc eu recours à un programme capable de générer toutes les grilles possibles (cf. section 1.2 (Chapitre 4)).

Pour expliquer comment nous avons trouvé cet ensemble de grilles et de positions sûres, nous allons désormais généraliser la théorie sur les jeux de Nim, aborder la notion de graphe de jeu, ainsi que le noyau d'un graphe.

4 Généralisation de la théorie des jeux de Nim

Nous avons vu qu'il existe un nombre infini de jeux de Nim et sur les 3 présentés dans la partie précédente, on a toujours été en mesure de trouver une suite de coups permettant à un joueur de gagner à chaque fois, peu importe la réplique de son adversaire. En plus, cela dépend du choix du joueur de commencer la partie ou non.

4.1 Modélisation d'un jeu sous forme de graphe

4.1.1 Recherche du graphe

Dans *Algorithms and Networking for Computer Games* [3], Jouni Smed et Harri Hakonen affirment que tout jeu à information parfaite (rien n'est caché à aucun des joueurs) peut être représenté sous la forme d'un graphe. On peut donc le faire pour tous les jeux de Nim.

En effet, on est dans la capacité d'énumérer chaque situation d'un jeu. Pour le jeu des allumettes, chaque situation peut représenter le nombre d'allumettes restantes. Pour le jeu de Nim classique, ce serait la liste de tous les tas d'allumettes possibles, en démarrant de la situation initiale. Ces situations sont les états d'un jeu et ce nombre d'états est fini.

Pour passer d'un état à un autre, on effectue un coup, comme retirer une allumette, ajouter deux croix, déplacer un pion... C'est ce qu'on appelle une transition d'un état à un autre.

Ces notions d'états et de transitions peuvent être représentées par un graphe, où les états sont les sommets et les transitions sont les arcs. Par exemple, pour le jeu des allumettes, on compte 13 états différents et la règle de retrait de 1 à 3 allumettes modélise les transition entre ces états :

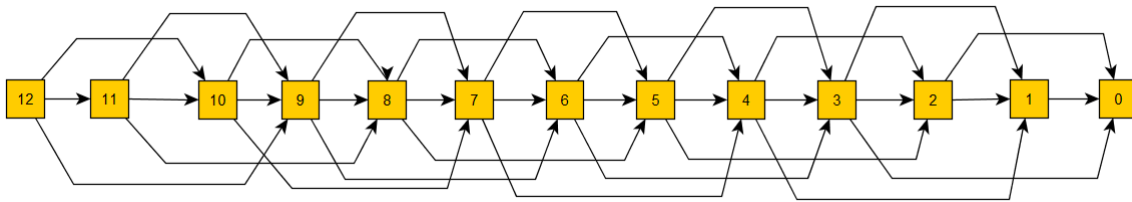


Figure 4 – Graphe du jeu des allumettes

4.1.2 Propriétés de ce graphe

On peut définir quelques propriétés à ce graphe grâce aux différents caractéristiques des jeux impartiaux.

Dans un jeu de Nim, une fois qu'on passe d'un état à un autre, il n'existe pas de mouvement permettant "d'annuler" le coup précédent (contrairement aux échecs notamment, où l'on peut faire reculer certaines pièces sur le plateau). Cette incapacité à retourner au coup ci-avant implique que le graphe du jeu est orienté sans circuit :

- Il est orienté, car pour aller d'un état à un autre, il n'existe qu'une seule direction : les arêtes du graphe ne peuvent être parcourues que dans un sens.
- Il est sans circuit, car si on passe d'un état E_1 à E_2 , il n'existe pas de suite de coups permettant de revenir à l'état E_1 .

Etant donné que le graphe d'un jeu de Nim est sans circuit, cela nécessite qu'il possède au moins une source et un puit. La source est représentée par la situation initiale du jeu. Le puit, qui est un sommet ne possédant pas de successeur, est représenté par l'état final du jeu, correspondant au moment où l'on ne peut plus jouer (aucune allumette restante, grille remplie de croix...). Pour la quasi-totalité des jeux de Nim, ce puit est unique. On est donc sûr d'arriver à ce puit et ce, quelle que soit la série de coups jouées.

4.1.3 De la complexité à définir le graphe d'un jeu – Exemple du Tic-Tac-Nim

Il était aisé de déterminer le graphe du jeu des allumettes puisqu'il n'y a que 13 états et que les règles du jeu limitent le nombre de transitions d'un état à un autre (il y en a 33 au total). Cependant, lorsque l'on arrive à des jeux avec des règles plus élaborées et des grilles plus grandes, on ne peut énumérer manuellement les différents états, et, sans programme informatique, construire le graphe de certains jeux aurait été très long.

Cette recherche de tous les états d'un jeu et de ses transitions s'inscrit dans la résolution de la complexité d'un jeu.

Tout d'abord, avant d'énumérer chaque situation existante, on peut en déterminer le nombre, ou, du moins, donner une valeur maximale possible de ce nombre, qu'on déduira en fonction des informations du jeu.

On sait que le Tic-Tac-Nim correspond à une grille de 3×3 , dont chaque case peut avoir 2 états : vide ou remplie. On aura donc $N = 2^9 = 512$ états possibles. C'est donc notre valeur maximale du nombre d'états possibles du jeu.

Cependant, on doit prendre en compte les rotations et les symétries de la grille, pour éliminer celles qui sont équivalentes. Les rotations et symétries sont les suivantes (on a une grille de base et 7 rotations/symétries) :

1	2	3	3	6	9	7	4	1	7	8	9	9	8	7
4	5	6	2	5	8	8	5	2	4	5	6	6	5	4
7	8	9	1	4	7	9	6	3	1	2	3	3	2	1
			9	6	3	1	4	7	3	2	1			
			8	5	2	2	5	8	6	5	4			
			7	4	1	3	6	9	9	8	7			

Figure 5 – Rotations et symétries d'une grille de Tic-Tac-Nim

Une fois celles-ci appliquées sur chacune de 512 grilles, on obtient $4096 (= 512 \times 8)$ grilles, puis, après avoir éliminé celles qui sont similaires, on obtient $N = 102$: c'est notre valeur exacte du nombre de grilles possibles. On peut ensuite générer le graphe du jeu grâce à cet algorithme :

1. Pour toute grille G_i du jeu :
2. On effectue un coup autorisé par les règles et on obtient une nouvelle grille G_i^* .
3. Si $G_i \neq G_i^*$, alors, pour toute grille G_j du jeu :
 4. Si $G_i^* = G_j$, alors on ajoute G_j à la liste des successeurs de G_i , et on quitte cette boucle.
 5. (On peut éventuellement ajouter G_i à la liste des prédecesseurs de G_j)

Cet algorithme a une complexité de $O(N \times M)$, avec N le nombre de grilles, et M le nombre maximum de coups autorisés sur une grille.

A partir d'une grille quelconque, les règles du jeu impliquent qu'on peut effectuer au maximum $M = 27$ combinaisons possibles de placement de croix pour la modifier. En effet, vu qu'on peut en placer 1, 2 ou 3, on a les possibilités suivantes :

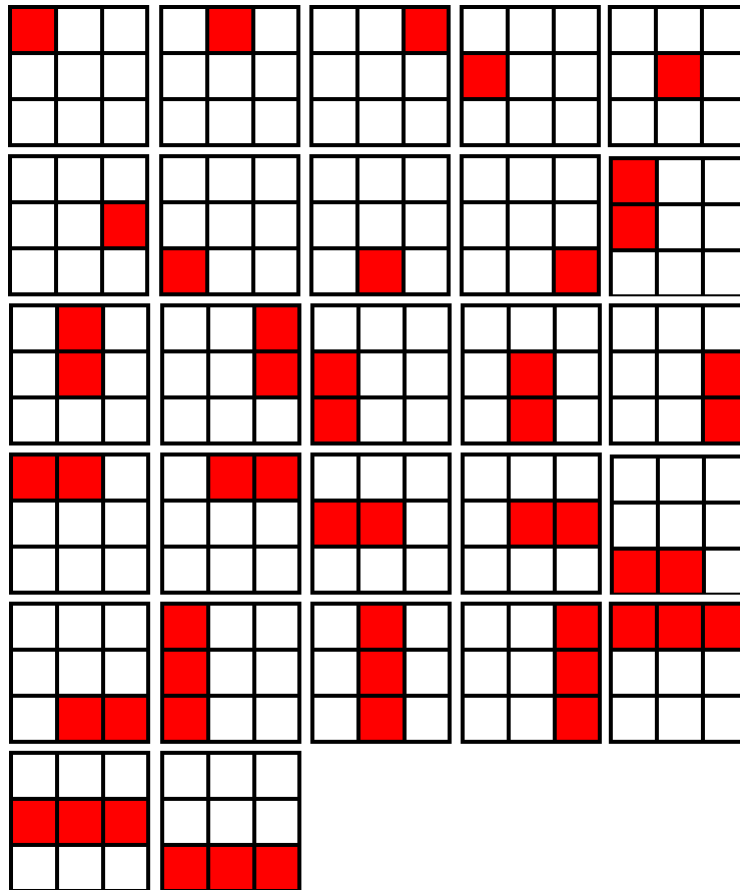


Figure 6 – Les 27 coups possibles sur une grille du Tic-Tac-Nim

Bien entendu, il se peut que l'algorithme place une croix sur une autre, ne modifiant ainsi pas la grille : on effectue bien une vérification (ligne 3).

On découvre ainsi que le graphe du jeu de Tic-Tac-Nim contient 725 arcs. A titre de comparaison, dans son article *Programming a Computer for Playing Chess* [7], Claude Shannon a estimé, pour le jeu des échecs, le nombre d'états possibles entre 10^{43} et 10^{50} , avec un graphe de jeu contenant 10^{120} arcs : le temps requis pour le générer est beaucoup trop long pour les ordinateurs de nos jours.

4.2 Recherche du noyau du graphe d'un jeu de Nim

4.2.1 Définition du noyau - exemple du jeu des allumettes

Maintenant que l'on a modélisé le jeu sous forme de graphe, on peut rechercher les positions gagnantes/sûres pour s'assurer la victoire.

Les graphes orientés sans circuit ont une propriété indispensable pour la résolution de la stratégie gagnante des jeux de Nim : ils possèdent un noyau unique. Le noyau d'un graphe correspond à un sous-ensemble de sommets, tel que :

- Il n'existe pas d'arc reliant 2 sommets du noyau. Du coup, tout sommet appartenant au noyau ne possède pas de successeur appartenant au noyau. [1]
- Tout sommet n'appartenant pas au noyau possède nécessairement au moins un successeur appartenant au noyau. [2]

On en déduit donc que l'état final E_f du jeu (le puit) appartient au noyau, puisqu'il ne possède pas de successeur, d'après [1]. Ainsi, tous les états $Ens_1 = \{E_1, E_2, \dots, E_n\}$ pouvant mener à cet état final ne sont pas dans le noyau, d'après [2].

En suivant ces 2 règles, on peut aisément déterminer le noyau du jeu des allumettes.

Etat	Dans le noyau?	Justification
0	Oui	L'état final appartient au noyau
1	Non	Il existe un successeur (0) appartenant au noyau
2	Non	Il existe un successeur (0) appartenant au noyau
3	Non	Il existe un successeur (0) appartenant au noyau
4	Oui	Les successeurs (3, 2 et 1) n'appartiennent pas au noyau
5	Non	Il existe un successeur (4) appartenant au noyau
...

On peut ainsi compléter notre graphe du jeu et mettre en évidence le noyau. On retrouve bien les positions et la stratégie gagnante établies dans la section 3.1.2.

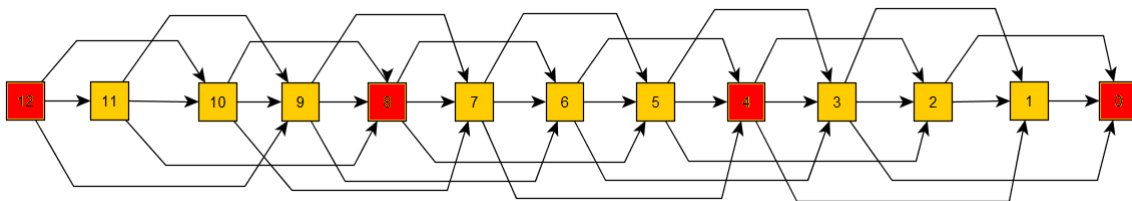


Figure 7 – Graphe du jeu des allumettes, avec mise en avant du noyau

4.2.2 Déterminer le noyau d'un graphe orienté sans circuit

On peut adapter cette recherche du noyau à tous les jeux de Nim, en utilisant l'algorithme suivant (qu'on retrouve sur le cours sur les graphes et théories des jeux de G. Montcouquiol [6]). Soient G le graphe, N le noyau $= \emptyset$

2. Tant que $G \neq \emptyset$
 3. On sélectionne tous les puits Ens_{puits} du graphe et on les ajoute au noyau ($N = N + Ens_{puits}$).
 4. On sélectionne tous les prédécesseurs Ens_{predec} de chaque sommet de Ens_{puits} , puis on supprime Ens_{predec} et Ens_{puits} du graphe G .

On va donc bien partir de l'état final (à la situation initiale de l'algorithme : $Ens_{puits} = \{E_f\}$), puis retirer ses prédécesseurs et E_f du graphe, ce qui va avoir pour effet d'obtenir un nouveau graphe possédant de nouveaux puits (qui vont être inclus dans le noyau). On continue ainsi jusqu'à ce qu'il n'y ait plus d'éléments dans le graphe.

Appliquons l'algorithme à un graphe simple. On va colorier en rouge les différents puits :

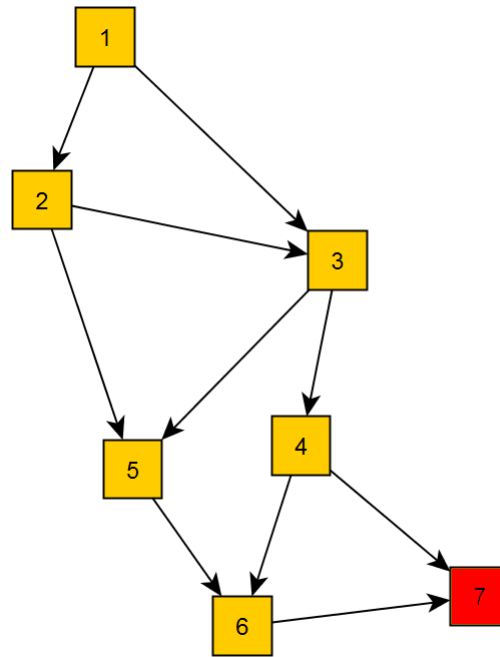


Figure 8 – Première étape de l'algorithme du recherche du noyau

Pour le moment, $N = \{7\}$. Les prédécesseurs de 7 sont 6 et 4 : ils ne sont pas dans le noyau. On élimine ensuite les états 4, 6 et 7, ce qui donne :

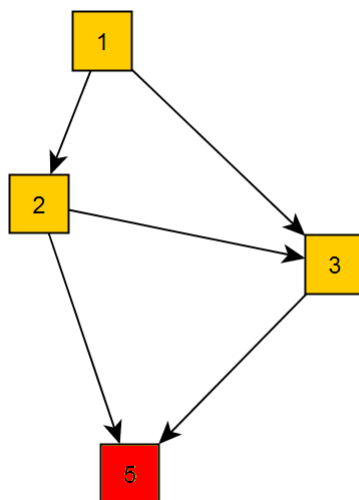


Figure 9 – Deuxième étape de l'algorithme du recherche du noyau

L'état 5 est un puit, car n'a pas de successeur : on l'ajoute donc au noyau. Il a pour prédécesseur 2 et 3. On élimine ensuite 2, 3 et 5, ce qui nous laisse uniquement l'état 1, qu'on ajoute donc au noyau. Finalement, on obtient le graphe suivant :

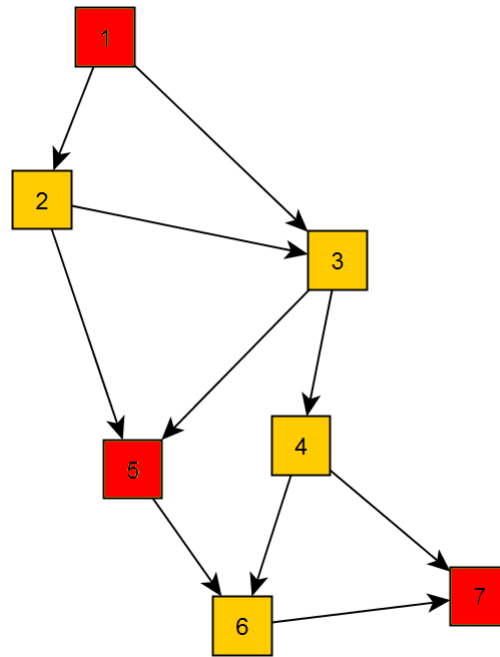


Figure 10 – Dernière étape de l'algorithme de recherche du noyau

Si on suppose que ce graphe est associé à un jeu, alors il est donc préférable pour ce petit graphe de laisser l'adversaire commencer, car il va quitter le noyau après son premier mouvement.

L'algorithme ci-dessus est simple et s'applique très simplement de manière visuelle. Nous allons désormais le spécifier pour qu'on obtienne une version la plus proche possible d'un pseudo-code.

Tout d'abord, pour avoir un algorithme efficace, les données du graphe sont traitées de la manière suivante : nous allons construire une liste dans laquelle on stocke, pour chaque état, ses successeurs et ses prédécesseurs. On utilise l'algorithme de création du graphe dans la section 4.1.3. La liste obtenue, appliquée au graphe précédent, serait la suivante :

Etat	Prédécesseurs	Successeurs
1	\emptyset	{1, 2}
2	{1}	{3, 4}
3	{1, 2}	{4, 5}
4	{3}	{6, 7}
5	{2, 3}	{6}
6	{4, 5}	{7}
7	{4, 6}	\emptyset

Ensuite, on effectue l'algorithme de recherche :

1. Tant que l'on trouve un nouvel élément à ajouter au noyau :
2. On cherche un E dans L qui n'a pas de successeur (= un puit du graphe).
3. Si on en trouve un :
 4. On ajoute E dans le noyau et on le marque comme appartenant au noyau.
 5. On sélectionne tous les prédécesseurs $Ens_{pred\acute{e}c}$ de E.
 6. On parcourt tout les noeuds du graphe. Si le noeud E_i parcouru n'est pas marqué, alors :

7. Si l'un des successeurs Ens_{succ} de E_i est dans Ens_{predec} , on retire cet état de la liste de Ens_{predec}
8. Sinon, si $E_i \in Ens_{predec}$, alors on marque E_i comme n'appartenant pas au noyau.

9. Sinon, on quitte la boucle : on a trouvé tous les éléments du noyau.

Au lieu d'avoir une notion de suppression, on a ici un système de marquage pour déterminer les états parcourus et non parcourus, qui appartiennent ou non au noyau.

Cet algorithme a une complexité polynomiale de $O(N \times M)$, avec N le nombre de sommets du graphe, et M son nombre d'arcs. Pour le Tic-Tac-Nim, on obtiendrait $N \times M = 102 \times 725 = 73950$ itérations au maximum.

5 Conclusion

On a vu que le jeu de Nim classique a été résolu depuis plus d'une centaine d'années, grâce à Charles L. Bouton. Cependant, on peut inventer des jeux de Nim tous les jours, ou modifier le plateau de jeu de ceux déjà existants, il faut donc trouver une méthode générique permettant leur compréhension et leur résolution, en définissant la stratégie gagnante.

L'apport de la modélisation des jeux de Nim sous forme de graphe a aidé à cette compréhension, notamment par rapport à la notion de noyau qui est indispensable ici. On peut déterminer les états le composant en parcourant toutes les combinaisons et/ou grilles possibles d'un jeu. On a vu qu'il y a 102 grilles possibles pour un Tic-Tac-Nim 3×3 , mais ce nombre peut être bien plus grand en augmentant la taille de la grille : construire le graphe du jeu et en déduire le noyau peut être potentiellement très long.

Finalement, créer une application offrant la possibilité d'ajouter son propre jeu de Nim ne dépend que de la capacité à résoudre ce jeu, en définissant le noyau. On a vu, selon le jeu, que la méthode de résolution peut être diverse : somme de Nim avec des jeux à tas d'allumettes (Marienbad) ou génération du graphe de jeu (Tic-Tac-Nim).

4

Analyse et conception

1 Premiers éléments à mettre en oeuvre

1.1 Mise en place d'un espace de projet

Concernant la gestion du projet, il a été suggéré d'ouvrir un espace de travail Redmine, fourni par Polytech. Le but va être de mettre en place les différents éléments, directives et méthodes pour obtenir une gestion efficace de développement de l'application.

Il sera également nécessaire de définir plusieurs points du développement : mise en place de prototypes et de jalons, période de tests...

1.2 Mise en place de l'algorithme pour résoudre le Tic-Tac-Nim

Mr Billaut m'a fourni la liste des 102 états du jeu (en éliminant les symétries et rotations), ainsi que les 28 grilles composant le noyau du Tic-Tac-Nim (voir la section 3.3.2 (Chapitre 3) .

On va donc écrire un programme C# pour le générer ; puisque c'est le langage de script utilisé par Unity. La première étape va être la capacité de générer ces 102 états. Pour cela, il faudra :

- Etre capable de générer les 512 différents états de la grille, dans un format commun. Puisque c'est une grille de 3x3, on générera sans doute un tableau de 9 cases, avec 0 représentant une case vide et 1 une croix. Une solution qui, à première vue, semble efficace, consiste à représenter en base 2 tous les nombres entre 0 et 512, sous un tableau de 9 cases (par exemple : 000000000 et 111111111 sont respectivement la grille vide et la grille remplie). On aura bien toutes les combinaisons possibles. Ensuite, on applique les symétries et rotations, pour obtenir une liste de 4096 grilles. Enfin, on élimine celles qui sont équivalentes, pour ensuite obtenir les 102 grilles voulues.
- Générer le graphe du jeu. Pour cela, on va parcourir chaque grille, essayer toutes les combinaisons possibles d'ajout de croix et déterminer si l'une des grilles obtenues est dans la liste. On va chercher à obtenir une liste de 102 objets contenant les éléments suivants : la grille, ses successeurs et ses prédécesseurs. (L'algorithme est décrit dans la section 4.1.3 (Chapitre 3))

- Appliquer l'algorithme de recherche du noyau sur ce jeu de noyau, tel qu'il est décrit au chapitre. (L'algorithme est décrit dans la section 4.2.2 (Chapitre 3))

On stockera ensuite ce noyau sous la forme d'un fichier texte. Selon le temps écoulé pour sa génération, on fera les choix suivants :

- L'opération est trop longue : on stockera en dur les différents états du noyau sous cette forme textuelle directement dans notre application.
- On laisse le programme le générer une unique fois lors du chargement de l'application ou du jeu, puis on le stocke en mémoire pendant toute la durée d'utilisation.

Il est fort probable que l'opération soit longue (plusieurs secondes), la piste de génération antérieure d'un fichier texte est donc privilégiée.

1.3 Mise en place du plateau de jeu générique

Une fois le noyau du Tic-Tac-Nim généré, on mettra en place le système de plateau abstrait (classe BaseGame du diagramme UML, section 4 (Chapitre 2)) et le spécifier pour implémenter le jeu du Tic-Tac-Nim. Cette étape est importante car elle va permettre de déterminer rapidement la robustesse de la modélisation du programme, afin de savoir s'il est aisé de rajouter d'autres types de grilles/plateaux de jeu. Le but est bien entendu de rajouter le moins de lignes de code possible pour la création d'un nouveau jeu.

2 Conception

2.1 Deuxième version du diagramme de classe

Bien qu'une première ébauche d'un diagramme de classe ait été avancée au semestre 9, il m'a fallu constamment le revoir pendant un mois environ durant le semestre 10. Le voici finalement :

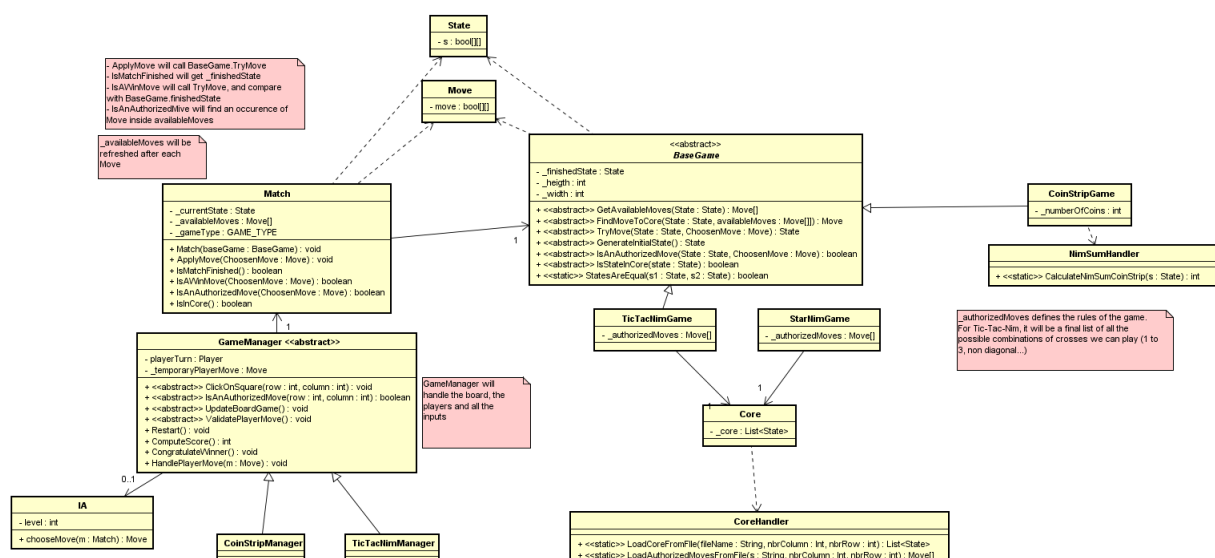


Figure 1 – Diagramme UML final

Une classe Match a été créée, possédant une instance du jeu de Nim. Elle contient également l'état actuel de la partie. De cette manière, le GameManager - gère le plateau graphique et le tour des joueurs - qui possède une instance du Match, ne manipule pas directement le jeu de Nim, et cette classe Match possède les méthodes nécessaires et suffisantes au Manager.

2.2 Respect des spécifications

L'application implémente les 2 jeux requis, que sont le Tic-Tac-Nim et le Star Nim. Ils sont entièrement fonctionnels.

L'intelligence artificielle a été développée avec les 3 niveaux de difficulté demandés ; le principal enjeu résidant dans le développement du mode Expert - rendant l'ordinateur imbattable si il est dans une situation favorable dès le début de la partie (voir section 4.2.1 (Chapitre 3)). On peut également jouer contre un autre joueur.

L'application propose une interface claire, épurée et dynamique. Elle est séparée en différentes scènes du jeu, que nous allons présenter.

2.3 Scènes

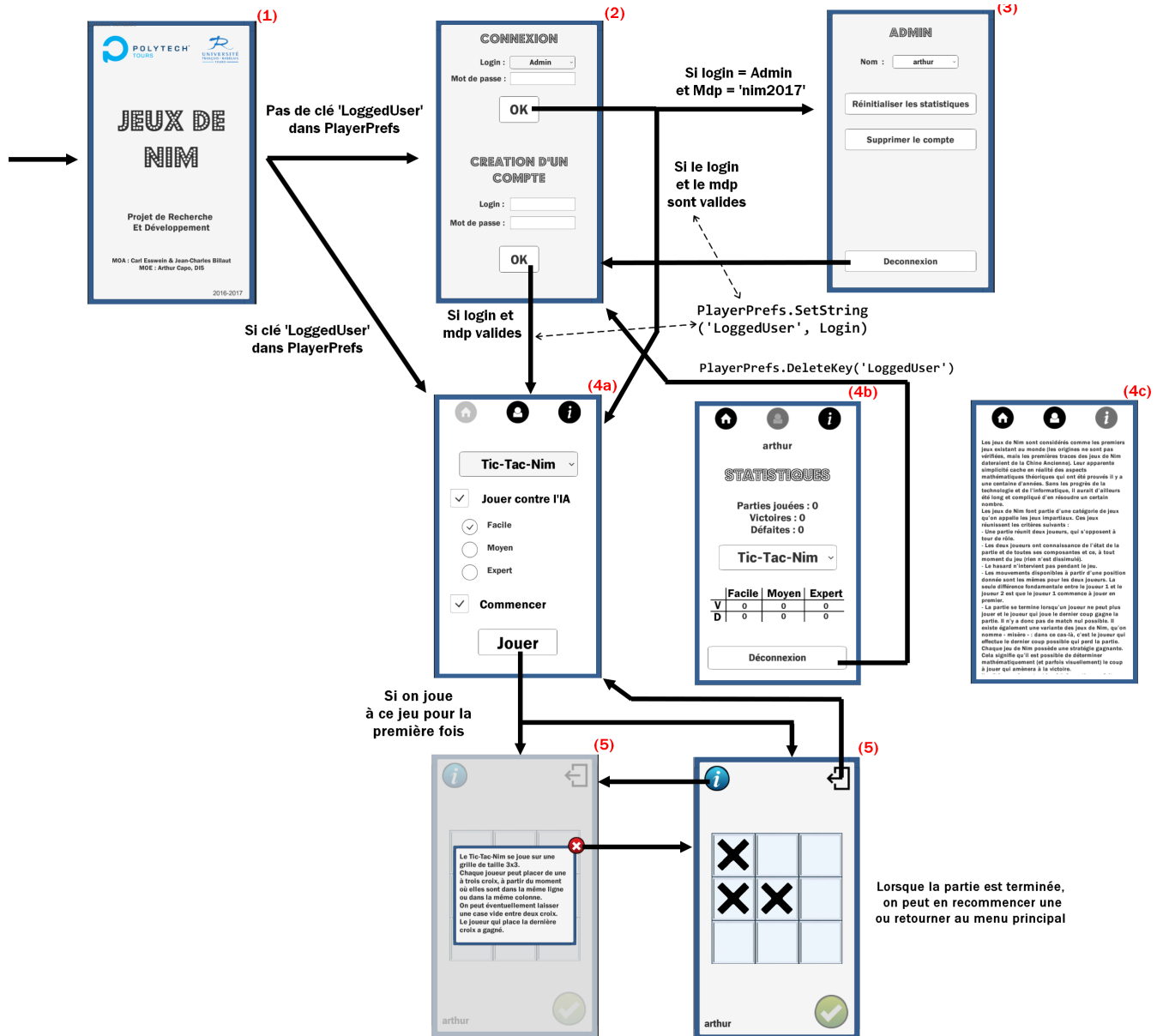


Figure 2 – Storyboard de l'application

- (1) : scène d'accueil, qui dure 2 secondes, avec lequel on ne peut pas interagir. Si un utilisateur était précédemment connecté, alors on charge le menu (4), sinon on charge la scène de connexion (2)
 - (2) : scène de connexion. Elle met en place plusieurs InputField, laissant la possibilité de :
 - Se connecter avec le compte Administrateur, pour aller à la scène de l'Administrateur (3)
 - Se connecter avec un compte joueur, pour se diriger ensuite vers le menu principal (4)
 - Créer un compte joueur, qui redirigera ensuite vers le menu principal avec ce compte (4)
- Des sécurités ont été mises en place pour ne pas créer plusieurs fois le même nom de

- compte. Le mot de passe est libre (il ne peut pas être vide cependant).
- (3) : scène Administrateur. Il peut sélectionner les joueurs parmi ceux qui se sont inscrits, les supprimer ou réinitialiser leurs statistiques.
 - (4) : scène du menu principal. Il se divise en 3 parties :
 - (4a) : scène du choix du jeu. On sélectionne le jeu, à l'aide d'un Dropdown (liste déroulante), la possibilité ou non de jouer contre une intelligence artificielle (si c'est le cas, on peut choisir sa difficulté), et le choix de commencer ou non la partie. On lance ensuite le jeu (5)
 - (4b) : scène Utilisateur. Le joueur peut consulter ses statistiques globales, par jeu et par difficulté. C'est également ici qu'il se déconnecte.
 - (4c) : scène A propos. On a un texte déroulant sur la théorie des jeux de Nim, il reprend en grande partie l'état de l'art de ce rapport)
 - (5) : la scène de jeu. C'est ici qu'on a le plateau de jeu, et qu'on joue contre son adversaire. Si le jeu est lancé pour la première fois, les règles s'affichent - et sont également consultables à tout moment. Le bouton situé en bas à droite permet de valider son coup. On peut quitter le jeu à tout moment avec le bouton situé en haut à droite.

5

Mise en oeuvre

1 Outils et versions

Comme décrit dans la partie 2.1 (Annexe A) , nous utiliserons le moteur de jeu Unity pour produire l'application. C'est la version 5.3.6 qui sera utilisée.

L'APK produit (executable sous Android) est compatible pour tous les smartphones supportant l'API 10 (Android 2.3.1 "Gingerbread"). Cela inclut plus de 99% des appareils Android aujourd'hui. Il a une taille de 20 Mo, ce qui en fait une application relativement petite par rapport à la moyenne des autres jeux.

La version iOS n'a pas pu être testée. En effet, je n'ai jamais eu de Mac à ma disposition. Cependant, exporter un projet Unity sous iOS n'est pas une manipulation particulièrement difficile, et de nombreuses ressources expliquent de manière didactique la procédure à suivre. Aucune autre librairie ou logiciel n'a été nécessaire durant le développement de l'application.

2 Implémentations

2.1 Nouvelles fonctionnalités

Au vu de l'avance relativement rapide du développement des fonctionnalités prévues, il a été possible d'en développer de nouvelles, qui m'ont été suggérées et/ou demandées par mes encadrants.

2.1.1 Développement du jeu de Star-Nim

Afin de tester la robustesse de la modélisation de l'application, j'ai essayé d'implémenter le jeu du Star Nim, présenté à la section 1 (Annexe B) . Pour rappel, l'un des points centraux de ce projet est d'offrir la possibilité à un développeur de créer son propre jeu de Nim.

Finalement, implémenter le jeu du Star Nim n'a pas été d'une grande difficulté - la moitié du temps a été dans la recherche du noyau du jeu, l'autre dans le développement, ce qui m'a pris

environ 5-6 heures de travail. Cela m'a également permis de peaufiner quelques méthodes, et de perfectionner mon modèle de données.

En complément de cela, un document "Comment ajouter son propre jeu de Nim" (chapitre C) a été écrit, décrivant étapes par étapes comme implémenter son jeu dans le projet.

2.1.2 Statistiques du joueur

Afin d'offrir de la rétention à l'utilisateur, il m'a été proposé de développer les statistiques d'un joueur.

En premier lieu, les statistiques "globales" ont été mises en place : on avait accès au nombre de parties jouées, de victoires et de défaites, indépendamment des jeux de Nim joués.

La deuxième étape a été de stocker les victoires et les défaites de chacun des jeux.

Enfin, la troisième et dernière étape a été d'afficher les victoires et les défaites de chaque jeu selon la difficulté. On a donc un écran de la sorte :



Figure 1 – Statistiques complètes d'un joueur

2.1.3 Comptes joueurs

Afin de permettre d'être à plusieurs personnes de jouer, et d'avoir des statistiques distinctes (pour suivre l'évolution de chacun), il a été nécessaire de proposer une dimension "Compte" à l'application. Elle est mise en place dans les scènes Login et Administrateur (voir 2.3 (Chapitre 4))

Pour les stocker, un fichier `users.dat`, qui contient la sérialisation de la liste d'utilisateurs (couple login/mot de passe), est mis à jour à chaque fois que l'on crée/supprime un compte. Il est stocké dans le dossier privé alloué à l'application. Au préalable, le mot de passe est haché avec un algorithme de type MD5.

Dans un second lieu, il a fallu stocker les statistiques de chaque joueur. Pour cela, un fichier `[nomUtilisateur].stats` est également stocké, qui est la sérialisation d'un dictionnaire associant

nom d'un jeu à des statistiques. Une classe Stats a été mis en place, contenant 2 tableaux de 3 cases : 2 tableaux pour victoire et défaite, et 3 cases chacun pour la difficulté. Les statistiques globales sont ensuite calculées à partir de ces données là.

Bien entendu, lorsque l'administrateur supprime un compte, ce fichier .stats est également supprimé.

2.2 Limites et faiblesses

Certains aspects du projet n'ont pas pu être réalisés. Bien que d'une importance peu capitales au fonctionnement de l'application, ceux-ci peuvent s'avérer préjudiciable pour une reprise sur le moyen terme.

- Le jeu n'a pas été testé sous différentes résolutions. J'ai eu l'occasion de tester l'application sur une tablette, mais sans aller très loin dans le détail : il est possible que certains éléments soient trop petits, ou mal agencés...
- Comme décrit section 1, la version iOS n'a pas pu être testée.
- Le design de l'application est assez limité, on utilise des éléments simples, il n'y a pas de transition entre les différents scènes... Des sons ont été envisagés, et n'ont finalement pas été implémentés.

3 Qualité et performance

3.1 Tests

Des tests ont été effectués pour vérifier si le comportement des jeux et de l'intelligence artificielle sont cohérents aux attentes du joueur et des règles.

Tout d'abord, nous allons voir si les jeux suivent bien les règles qu'on leur applique. Pour les 3 jeux, on effectue une partie, en proposant des mouvements prédéfinis à chaque tour, et on vérifie si tout est cohérent. Cela inclus :

- Vérifier si on est dans le noyau ou non
- Vérifier si, après coup, on obtient le bon état (en comparant l'état initial + le coup et l'état suivant)
- Vérifier si la partie est finie ou non
- Vérifier s'il existe un coup gagnant
- Vérifier qu'il n'y ait plus de mouvements disponibles, lorsque la grille est remplie (pour le Tic-Tac-Nim par exemple),

On vérifie la robustesse de l'IA lorsqu'il est en mode expert. Pour cela, on fait en sorte que lorsque le joueur commence à jouer, il soit dans le noyau - ainsi, il en sortira au coup suivant.

L'algorithme est le suivant :

- Tant que le match n'est pas fini
 - Le joueur choisit un coup aléatoire (parmi les coups disponibles) et le joue
 - On vérifie que le joueur n'est pas dans le noyau, et que l'état obtenu est différent de l'état final
 - L'IA choisit un mouvement dans le noyau, et le joue
 - On vérifie que le nouvel état E appartienne au noyau
 - S'il n'existe plus de coups à jouer
 - On sait que le match est fini, on regarde si E est égal à l'état final

On a donc la preuve que les règles du jeu sont cohérentes, et que la classe XGame a été correctement implémentée.

Une phase de bêta-test s'est déroulée du 23 au 30 mars auprès de plusieurs joueurs, afin de tester l'application avec une vision autre que la mienne, et éventuellement me faire part de divers retours. Au final, des ajustements minimums ont été faits (fautes d'orthographe), mais rien n'a été modifié en profondeur du programme. Ceci démontre à nouveau la robustesse de l'application, puisqu'aucune faille n'a été découverte pendant cette période.

3.2 Performances

Les performances de l'application n'ont jamais été une inquiétude, au vu de la relative simplicité du projet.

La seule problématique a été de rechercher le noyau du jeu de Tic-Tac-Nim. Un programme C# a été écrit pour rechercher et stocker ce noyau dans un fichier texte. On relève un temps d'exécution de 4-5 secondes, ce qui n'est pas acceptable pour le chargement d'une application, où le joueur ne s'attend pas à patienter aussi longtemps. Il a donc été décidé de fournir directement le fichier texte contenant le noyau dans le programme Unity, pour une expérience utilisateur optimale. D'ailleurs, si un développeur souhaite ajouter son jeu de Nim (contenant un noyau fixe), il doit fournir ce fichier texte.

Un profiler a été lancé pour déterminer des performances pendant l'exécution d'un jeu de Nim. Au final, on ne descend jamais en dessous des 60 frames par secondes, ce qui est largement acceptable - d'autant plus qu'il n'y a pas de réelle animation pendant le déroulement du jeu (hormis l'ajout ou le retrait/déplacement de croix/pièces).

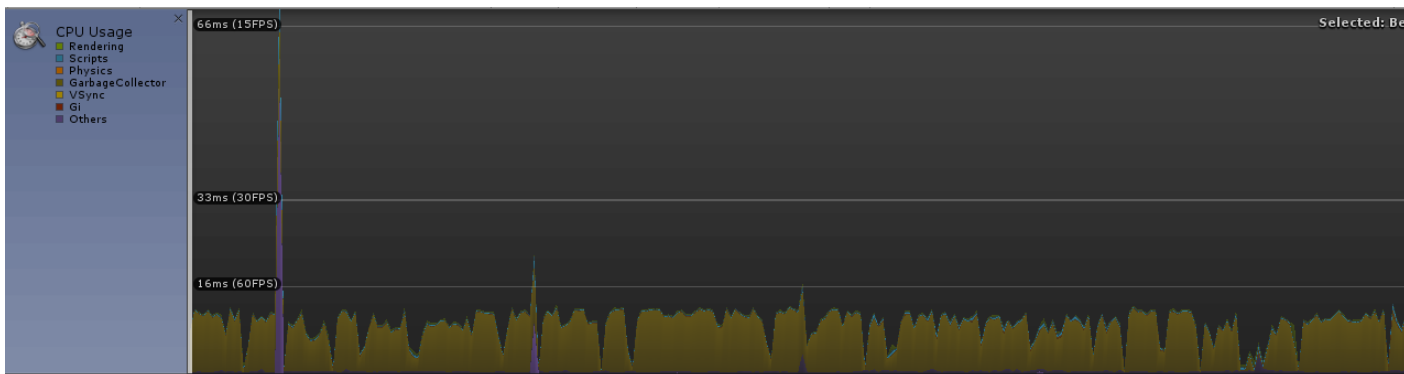


Figure 2 – Profiler

3.3 Améliorations

Bien que l'application soit complètement fonctionnelle, elle est toujours sujette à des améliorations.

- Il existe toujours des mécanismes pour améliorer la rétention du joueur. Pour le moment, le seul élément qui va au delà du jeu est l'affichage des statistiques. On peut en imaginer d'autres : un certain score en fonction du temps de jeu et de la difficulté, un jeu qui en débloque un autre...
- L'interface graphique est simpliste, et ne possède pas de réelle personnalité. Mettre en place une charte graphique, et donner une identité à l'application aurait donné un côté plus attractif.

- Cela aurait doublé le temps de développement, mais les jeux qui se jouent à 2 joueurs au tour par tour, comme les jeux de Nim, s'adaptent très bien pour jouer en ligne. On peut mettre en place des vrais comptes joueurs, des rooms...

6

Bilan et conclusion

La partie Recherche de ce PRD arrive à son terme, et elle m'a permis de prendre conscience de plusieurs aspects de ce projet.

En premier lieu, je me suis rendu compte de l'importance des spécifications : sans un document précis, résumant toutes les informations nécessaires, le projet serait flou, mal défini, et il serait difficile de se faire une idée concrète de la direction à prendre : les objectifs, les utilisateurs potentiels, la structure du système...

Par la suite, ces 3 mois m'ont permis de me documenter sur une catégorie de jeux qui m'intéressait, dont la littérature existante n'est pas très vaste : cela m'a permis de concentrer rapidement et efficacement les informations qui m'étaient nécessaires. La teneur de ce travail de recherche a abouti sur la création d'un algorithme de recherche du graphe de jeu de Nim, ainsi que le noyau le composant.

Cela constitue une bonne introduction à la théorie des jeux, pouvant amener à des recherches et des théories plus complexes (par exemple : DeepMind de Google, émulant un joueur de Go, faisait appel à des techniques de deep learning).

La manière dont s'est déroulée cette partie Recherche n'a pas été optimale. En effet, ma première version de l'état de l'art était très incomplète, et il m'a fallu le corriger en un laps de temps assez court - un problème que j'aurai pu éviter de 2 manières :

- Rendre une première version du document plus tôt dans la phase de recherche, afin d'effectuer des corrections rapidement.
- Avoir un recul nécessaire et rendre un document bien plus élaboré assez tôt.

La partie Développement de ce projet était très intéressante. Elle m'a permis d'appliquer directement les travaux effectués dans la partie Recherche, et notamment la mise en place de l'algorithme de recherche de noyau.

Développer une application de A à Z a été très satisfaisant. Du fait du bon déroulé de ce semestre, j'ai pu rendre une version complète du résultat attendu. J'ai également pu mettre en place de nouvelles fonctionnalités à l'application.

Je remercie Mr Esswein et Mr Billaut pour leur encadrement, leur conseil, et la confiance qu'ils m'ont accordé pendant tout ce projet. Je remercie également tous les acteurs du PRD pour leur accompagnement.

Annexes

A

Spécifications fonctionnelles et non fonctionnelles

1 Spécifications fonctionnelles

1.1 Choix du jeu

Cette fonctionnalité permet à l'utilisateur de choisir le jeu de Nim auquel il veut jouer. Dans l'application de base, il pourra choisir entre le jeu de Marienbad ou le Tic-Tac-Nim.

1.2 Personnaliser la partie

Cette fonctionnalité permet au joueur de modifier certains paramètres de la partie, notamment le choix de vouloir jouer contre un autre joueur ou une intelligence artificielle, et, si tel est le cas, de choisir son niveau de difficulté (facile, moyen, difficile). La taille éventuelle d'une grille peut également être modifiée (c'est le cas pour le jeu Northcott).

On laisse également le choix au joueur de décider d'effectuer ou non le premier coup de la partie.

1.3 Jouer un coup

Lorsque le joueur est dans une partie, il peut jouer un coup : cela peut être le déplacement d'un pion, remplir une case, retirer une allumette... chaque joueur joue un coup à tour de rôle. On sera éventuellement amené à valider le coup dans certains jeux avec l'ajout d'un bouton.

2 Spécifications non fonctionnelles

2.1 Contraintes de développement et conception

Pour le développement du jeu, il sera nécessaire de tester les différents prototypes du jeu sous différents smartphones. L'idéal serait de tester sur 2 systèmes et résolutions différentes :

- Un smartphone Android, comme un Samsung Galaxy par exemple
- Une tablette sous Android, telle que la tablette Archos fournie par la Région Centre en 1ère année du cycle ingénieur
- Un iPhone
- Un iPad

Cela implique donc qu'il faut souscrire à un compte développeur Apple, car on ne peut pas exporter d'applications sur les appareils Apple sans celui-ci. Il n'y a pas d'exigence particulière pour les systèmes Android.

Pour obtenir une application cross-plateforme, le choix s'est naturellement tourné vers Unity, qui est un moteur de jeu capable de produire des applications/jeux qu'on peut exporter sous différents systèmes (Android, iOS, fichiers exécutables, Web...) sans avoir à effectuer de modification selon les plateformes (hormis quelques configurations mineures). Deux applications natives iOS et Android auraient été trop longues à développer.

Comme il n'est pas possible de développer tous les jeux de Nim existants (sachant qu'il est possible d'en inventer de nouveaux), un enjeu est de prévoir la reprise de l'application par d'autres personnes. On doit donc fournir un ensemble d'outils, une architecture adaptée et claire, et les documents nécessaires pour un éventuel futur.

2.2 Contraintes de fonctionnement et d'exploitation : performances et capacités

On veut bien entendu faire en sorte que le joueur patiente le moins possible lors du chargement de l'application ou d'une partie. Cela ne devrait pas poser de problèmes, étant donné que les graphismes sont relativement simplistes et ne devraient pas prendre énormément de temps à charger.

Si l'on a affaire à une intelligence artificielle, on doit également optimiser la phase de choix d'un coup, notamment lorsqu'il est programmé pour utiliser la stratégie gagnante d'un jeu de Nim, qui prendra plus de temps qu'un mouvement aléatoire. Il faudra être attentif à cet aspect-là pendant le développement : on veut faire en sorte que l'IA choisisse son coup en moins d'1 ou 2 secondes. L'utilisateur sera plus enclin à patienter avant le lancement d'une partie, plutôt qu'à chaque choix de mouvement de l'adversaire.

Une fois de plus, dûe à la simplicité des graphismes, le jeu ne devrait pas requérir énormément de capacité de stockage : entre 20 et 30 Mo environ.

B

Présentation et résolution de plusieurs jeux de Nim

Dans cette annexe, nous présenterons divers jeux de Nim qui seront disponibles dans le plateau de jeu (cf 1 (Chapitre 1)).

1 Jeu de Star Nim

1.0.1 Présentation et règles

Ce jeu se présente sur un tableau de 8 cases, reliés entre elle selon un schéma bien précis :

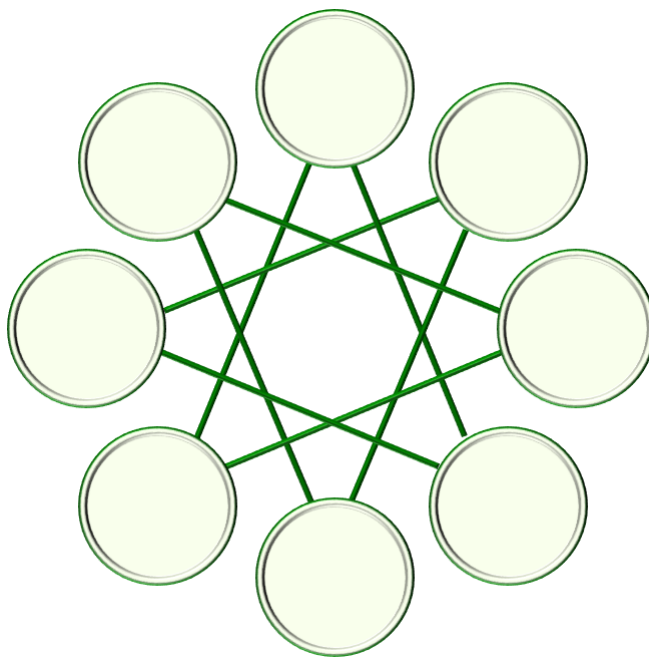


Figure 1 – Plateau du jeu de Star Nim

Chaque joueur peut à tour de rôle poser soit :

- Un pion sur une case

— Deux pions si et seulement si elles sont reliées entre elles par un segment.

Le jouer posant le dernier pion gagne.

1.0.2 Stratégie gagnante

Comme il n'y a que 8 cases, et une série limitée de liens entre celles-ci, on peut facilement énumérer la liste des états du jeu. Les voici représentés graphiquement, on place ici nos cases sur un repère à une seule dimension, classées par nombre de cases pour chaque cas. On en relève 23 (y compris la grille initiale et finale).

8 Grille vide

7 ○—○—○—○—○—○—○

6 ○—○—○—○—○—○ ○—○—○—○—○—○
○—○—○—○—○—○ ○—○—○—○—○—○

5 ○—○—○—○—○ ○—○—○—○—○ ○—○—○—○—○
○—○—○—○—○ ○—○—○—○—○ ○—○—○—○—○

4 ○—○—○—○ ○—○—○—○
○—○—○—○ ○—○—○—○ ○—○—○—○

3 ○—○—○ ○—○—○ ○—○—○

2 ○—○ ○—○

1 ○

0 Grille remplie

Figure 2 – Les 23 états du Star Nim

On est ensuite capable, sans trop d'effort, de recréer le graphe du jeu (les éléments du noyau ont une pastille rouge) :

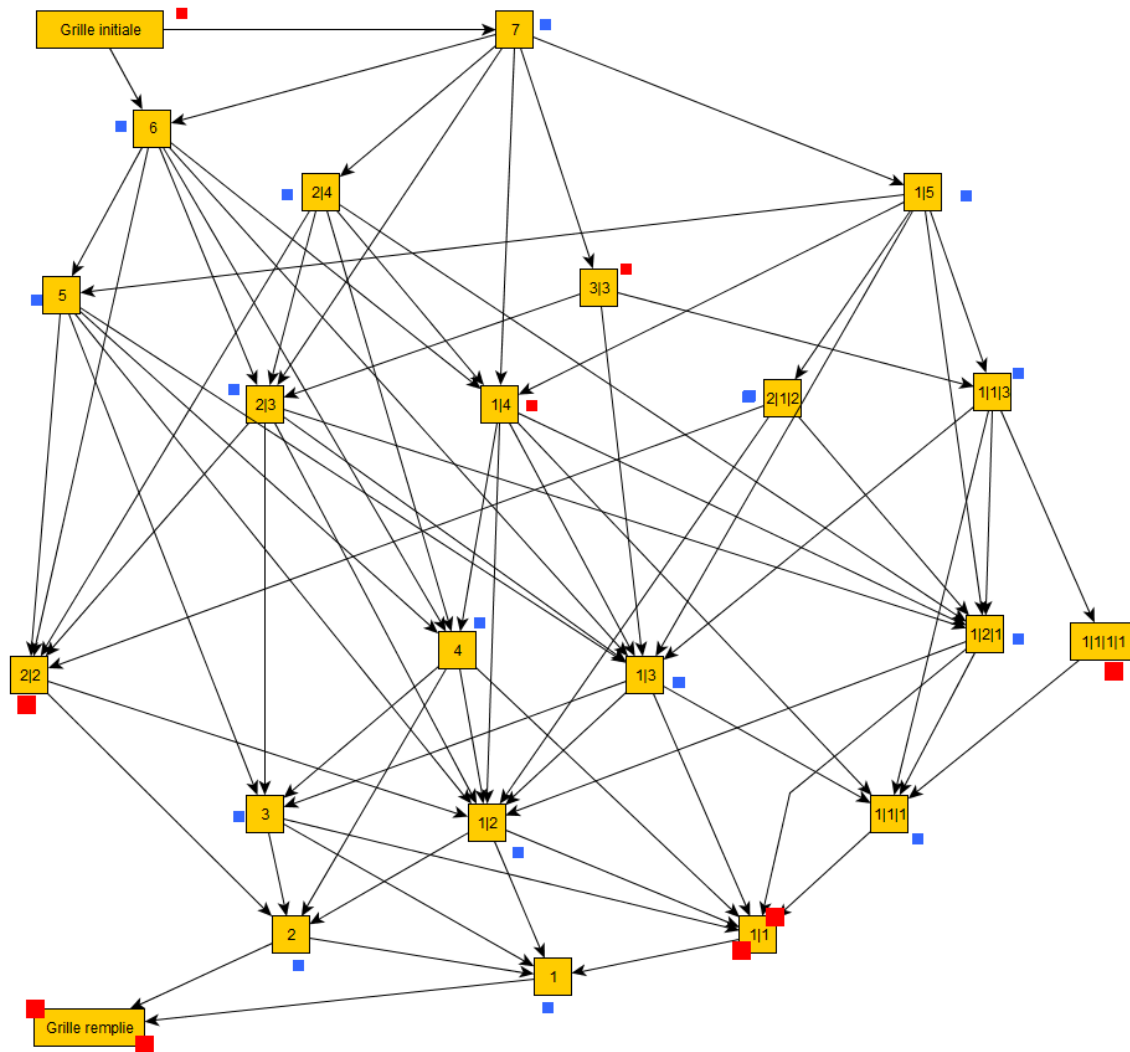


Figure 3 – Le graphe du jeu de Star Nim

La notation $X|Y$ signifie "X cases reliées entre elles, et Y cases reliées entre elles, mais aucun lien n'existe entre X et Y".

On découvre qu'il y a 7 états dans le noyau, incluant la grille initiale. Pour ce jeu, il est donc préférable de laisser l'adversaire commencer.

2 Coin-Strip

2.0.1 Présentation et règles

Ce jeu consiste en une ligne où l'on place plusieurs pièces. A chaque tour, le joueur déplace une pièce vers la gauche d'autant de cases qu'il le désire, à partir du moment où il ne passe pas par dessus une autre pièce. Le but est d'effectuer le dernier coup possible : l'adversaire ne pourra donc plus jouer et toutes les pièces seront les unes à côté des autres, en partant de la gauche.



Figure 4 – Exemple d'une partie de Coin-Strip

2.0.2 Stratégie gagnante

Comme décrit par John Conway, dans *On Numbers and Games* [2], les écarts entre les pièces peuvent être vus comme des tas d'allumettes, on peut donc utiliser la même méthode de résolution décrite dans la section 3.2.2 (Chapitre 3). La taille des différents tas sont les longueurs des écarts entre chaque pièce 2 à 2, en commençant par la pièce située le plus à droite. Si le nombre de pièces est impair, alors le dernier tas a comme taille l'écart entre la dernière pièce et le coin gauche de la grille. On inclut les tas de taille 0 (les 2 pièces sont côte à côte).

La particularité de ce jeu est que l'on est dans la capacité d'augmenter l'écart entre les pièces, et donc d'augmenter la taille des tas. On caractérise le Coin-Strip de version "bogus" (buggée) du jeu de Nim.

Dans l'exemple ci-dessus, on aurait des tas de taille 0, 3 et 1 :



Figure 5 – Ecart entre les pièces

On effectue donc le calcul de la somme de Nim des différents tas :

1	0	1
3	1	1
0	0	0
<hr/>		
	1	0

La somme est différente de 0 : on va donc chercher à commencer la partie. Ici, on remarque que 5 mouvements sont disponibles, on va donc tester ces différentes combinaisons et déterminer la somme de Nim de chacun des écarts obtenus.

Ici, l'un des coups à jouer peut être de déplacer la pièce en case 6 vers la case 4. Ainsi, les tas et le calcul de la somme de Nim seraient les suivants :

1	0	1
1	0	1
0	0	0
<hr/>		
	0	0

3 Jeu de Hex

3.0.1 Présentation et règles

Le jeu de Hex se joue sur un plateau, de taille variable : il existe des versions 9×9 , 11×11 , 14×14 ... A partir du moment qu'il y ait autant de cases dans la longueur et dans la largeur.

Les 2 joueurs sont représentés par une couleur. Le but est de poser, à chaque tour, un pion de sa couleur afin de relier les deux côtés du plateau qui lui ont été assignés. On doit donc être capable de tracer une ligne avec les pions posés pour déterminer si on gagne la partie.

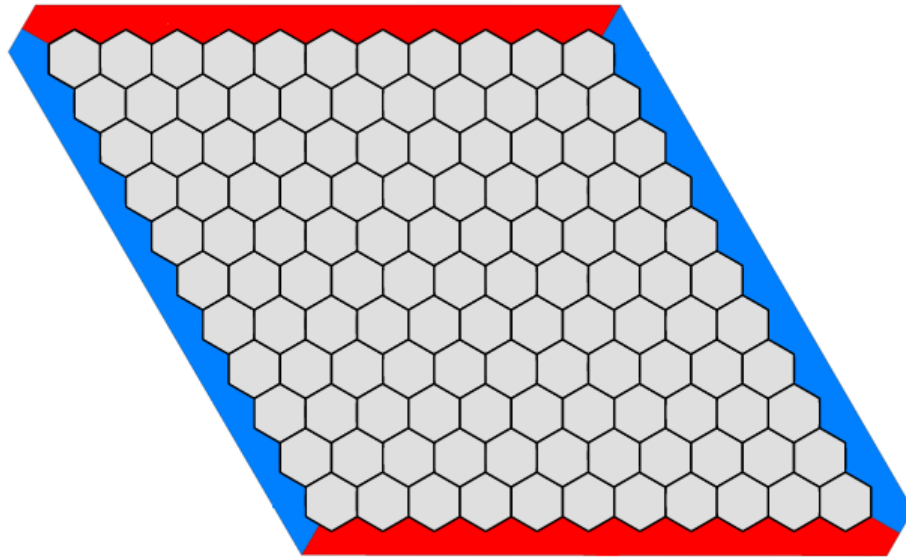


Figure 6 – Plateau d'un jeu de Hex 11×11

Bien qu'il y ait une notion de couleur, à tout moment de la partie, chaque joueur peut effectuer le même mouvement que l'autre. De plus, il n'existe pas de match nul possible pour ce jeu

3.0.2 Stratégie gagnante

Sur un plateau de taille 11×11 , on a un nombre d'états $N = 3^{11 \times 11} = 5.4 \times 10^{57}$. Construire le graphe du jeu est donc beaucoup trop long. De nombreux travaux ont été effectués sur le jeu de Hex, et la stratégie gagnante n'a pas été définie à ce jour. Il est prouvé que la recherche de cette stratégie est un problème NP-complet : il est difficile voire impossible de le résoudre avec un algorithme polynomial.

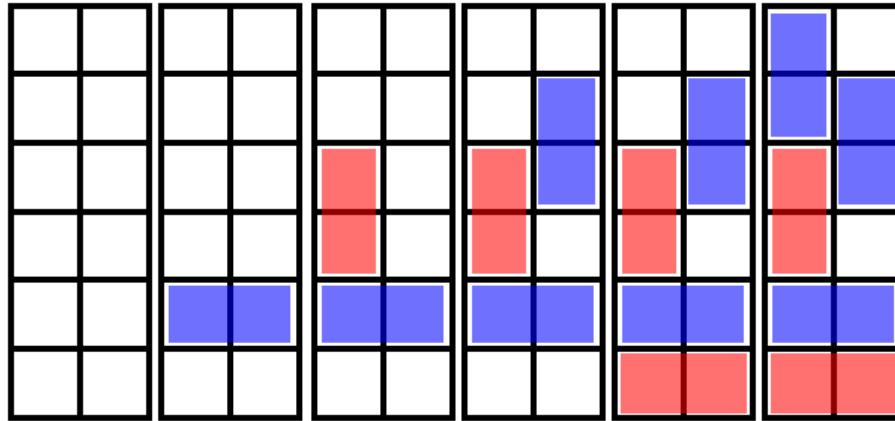
En théorie, le joueur qui commence à jouer possède une stratégie gagnante, mais c'est la seule information qu'on peut en déduire.

4 Jeu de Cram

4.0.1 Présentation et règles

Le jeu de Cram consiste en un plateau de taille $N \times M$, avec $N \geq 2$ et $M \geq 3$, dont le but est de placer à chaque tour un domino de taille 2×1 , horizontalement ou verticalement, afin d'être le dernier joueur à pouvoir effectuer un mouvement, sachant que les dominos ne se superposent pas.

Voici l'exemple d'une partie sur un tableau de taille 2×6 . Ici, c'est le joueur qui possède les dominos bleus qui commence. D'ailleurs, c'est lui qui va gagner la partie, puisque c'est le dernier à poser le sien.


 Figure 7 – Exemple d'une partie de Cram sur un plateau 2×6

4.0.2 Stratégie gagnante

La stratégie gagnante est basée sur le jeu par symétrie. Lorsque le joueur 1 place son domino, alors son adversaire doit placer le sien par symétrie au avec le point central du plateau.

Par exemple, sur un plateau de 2×6 , lorsque le premier joueur place son domino, remplissant les cases A et B, alors le second joueur doit faire de même sur les cases symétriques A et B correspondantes :

1	2	1	1
3	4	2	2
5	6	3	3
5	6	4	4
3	4	5	5
1	2	6	6

 Figure 8 – Symétrie des cases du jeu de Cram sur un plateau 2×6

Pour une grille de taille $N \times M$, avec N et M deux nombres pairs, c'est le second joueur (celui qui ne commence pas la partie) qui peut gagner en jouant par symétrie, tandis que pour un tableau de taille $N \times M$, avec N pair et M impair, c'est le premier joueur qui doit jouer de la sorte.

C

Comment ajouter son propre jeu de Nim dans l'application

Il est possible, pour un développeur ayant des connaissances de base dans le moteur Unity, d'ajouter son jeu de Nim. Pour cela, il y a une procédure à suivre :

1. Dans le fichier Global.cs (situé dans le dossier Scripts), ajouter au tableau NimGames une instance NimGameInfos du nouveau jeu de Nim. En paramètre, il est nécessaire de fournir :
 - L'identifiant Id le jeu de Nim
 - Le nom X du jeu de Nim
 - Sa couleur associée.Voir la ligne 36 (qui est commentée) du fichier Global.cs
2. Créer un dossier dans Resources, ayant pour nom Id
3. Dans ce dossier, il faut ajouter plusieurs éléments :
 - Un fichier image nommé sprite_128x128, correspondant au sprite avec lequel on va remplir une case (une pièce, une croix, etc). Utiliser préférentiellement un format .png, pour gérer la transparence de l'image.
 - Un fichier texte nommé infos, contenant les règles du jeu, qui sera affiché lors du premier lancement du jeu (également disponible avec un bouton Infos pendant le jeu)
 - Un prefab nommé GamePanel. Celui-ci doit contenir au moins 2 composants :
 - Un RectTransform, dans la position X et Y est égale à (0 ; 0)
 - Un script XManager
 - Eventuellement : une image (comme le StarNim) ou un Layout (CoinStrip et TicTacNim)Ce prefab doit OBLIGATOIREMENT doit également avoir des GameObject enfants de type Button (c'est-à-dire : ce GameObject doit contenir un composant de type Button et de type Image)
 - NON OBLIGATOIRE : un fichier texte nommé Core, contenant le noyau du jeu
 - NON OBLIGATOIRE : un fichier texte nommé Move, contenant les mouvements autorisés d'un jeu
4. Compléter les 2 scripts :
 - XGame.cs
 - XManager.cs

La classe XGame, héritant de BaseGame, BaseGame contient toutes les informations nécessaires à un jeu de Nim. Il va contenir sa taille et son état final en attributs, et définit

les règles du jeu par le biais des différentes méthodes à implémenter : quels sont les mouvements autorisés, comment passer d'un état à un autre, et toute la dimension "noyau" d'un jeu de Nim. Il ne faut pas oublier d'assigner des valeurs positives non nulles à width et height). La classe GameManager va servir de lien entre les entrées utilisateurs et l'aspect graphique du jeu avec le jeu en lui-même. Il possède un attribut de type Match, qui va charger un jeu de Nim (de type BaseGame). Voir les commentaires TODO de ces fichiers

D

Gestion de projet

1 Plannings

Voici le planning réel du semestre 9 :

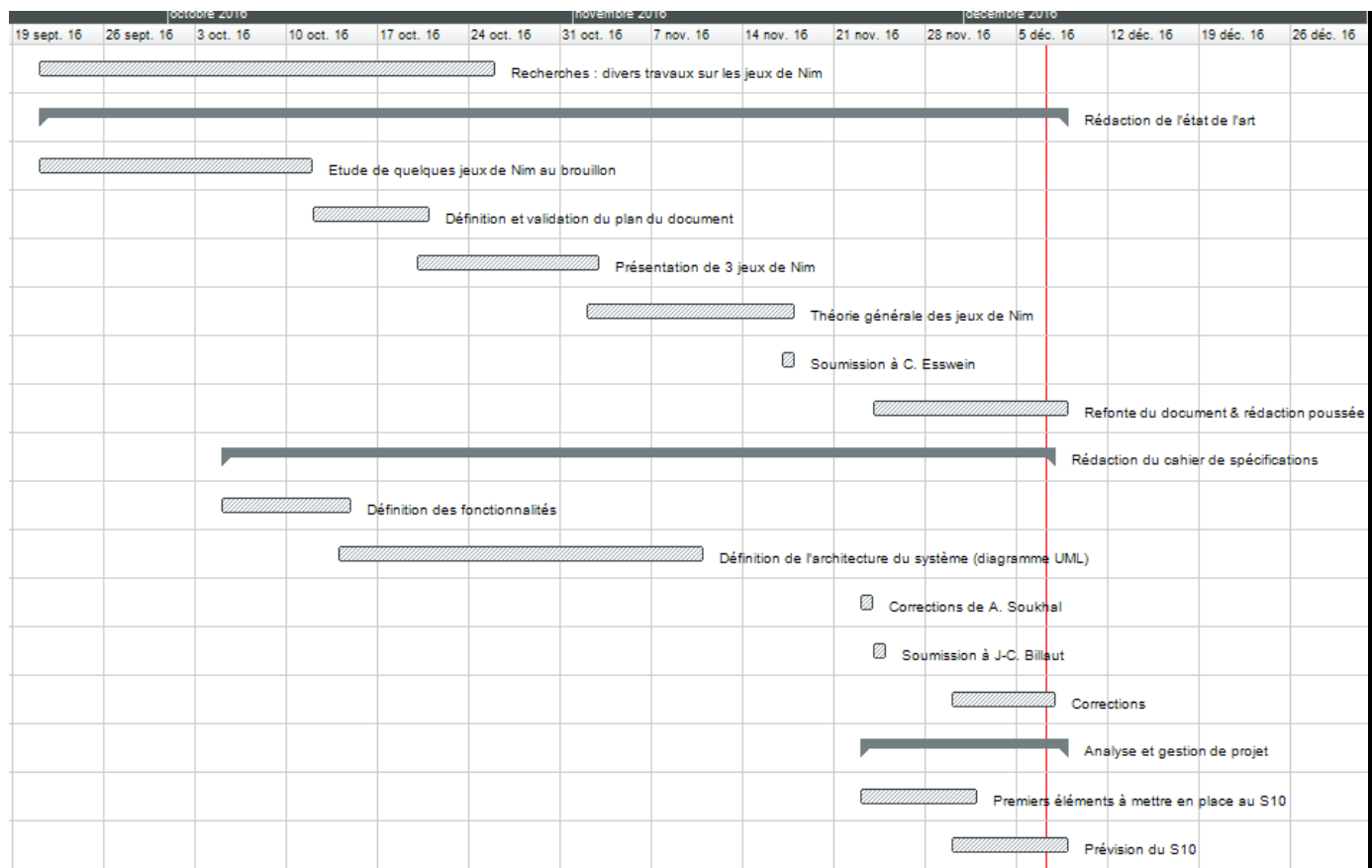


Figure 1 – Diagramme de Gantt réel du S9

Dans la phase d'analyse, il m'a également fallu prévoir les éléments à mettre en place au

semestre prévu au développement de l'application. Cette analyse se modélise sous la forme de ce diagramme de Gantt, qu'on essaiera de respecter le plus fidèlement possible :

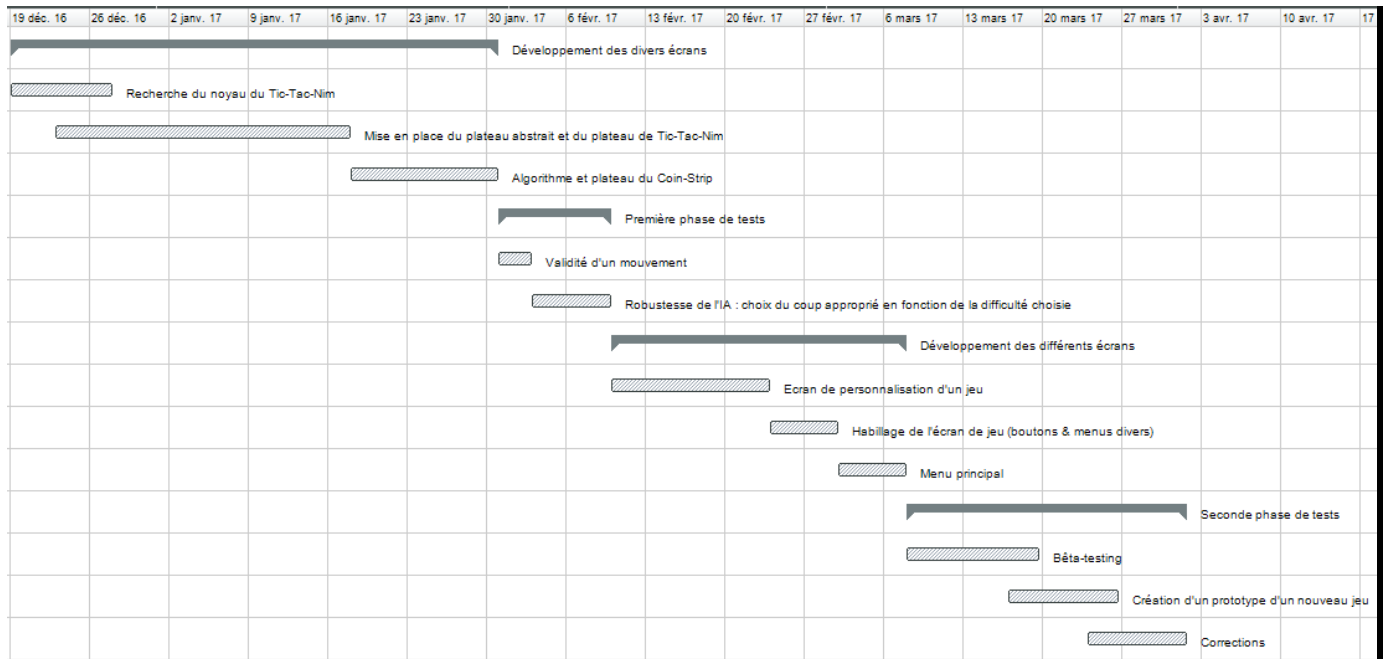


Figure 2 – Diagramme de Gantt prévisionnel du S10

Au final, il s'est avéré que certaines tâches ont été plus rapides que prévues, d'autres plus longues et d'autres n'ont pas complètement existé :

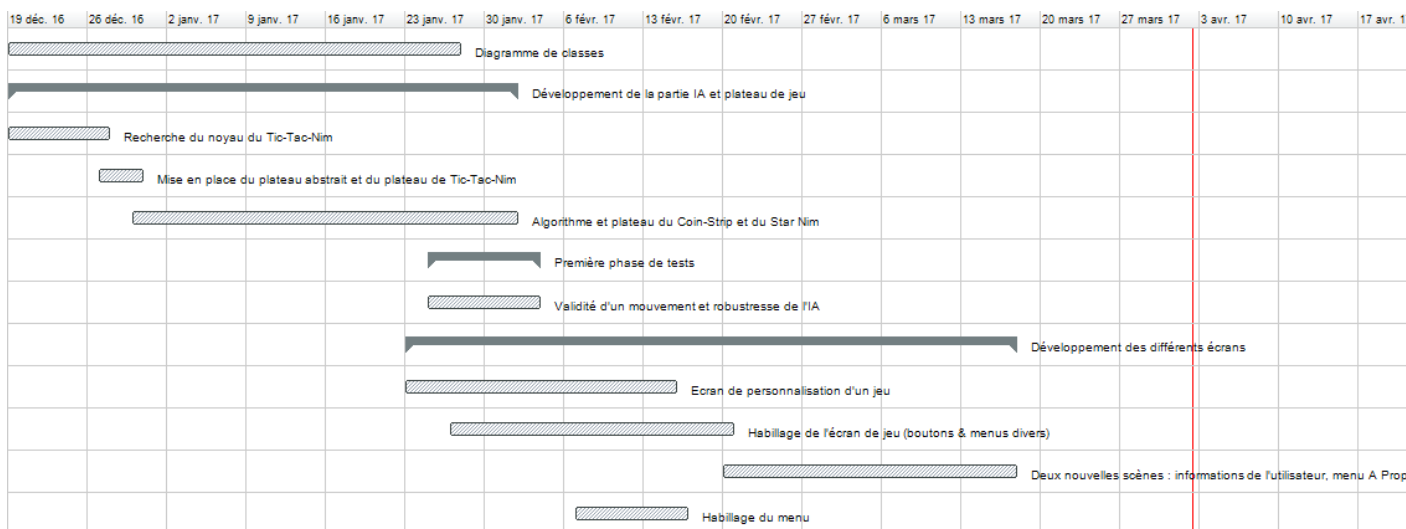


Figure 3 – Diagramme de Gantt réel du S10

Ce Gantt est donc peu représentatif. De plus, pendant le déroulement du projet, nous avons eu certaines périodes avec beaucoup de temps disponibles (périodes de Noël), parfois très peu (priorité des autres matières, etc...)

2 Analyse de faisabilité

2.1 Spécifications de base

Le projet comprend l'aspect suivant : on doit offrir la possibilité à un développeur de pouvoir ajouter son propre jeu de Nim. Pour cela, il a fallu penser l'architecture du programme de sorte à ce que cet ajout se fasse le plus aisément et le plus rapidement possible.

Pour mettre à bien cet aspect-là du projet, de nombreuses procédures ont été mises en place. Tout d'abord, l'élaboration du diagramme de classes a été étendue pendant un mois après la fin du semestre 9, et ce après plusieurs revues de la part de Mr Esswein.

Ce temps de réflexion a abouti en une élaboration de classes qui s'est avéré être efficace dans le sens où il m'a été aisé de développer un 3ème jeu de Nim, qui n'était pas prévu à la base. (Voir 2.1 (Chapitre 4))

Pour que le développeur reprenneur mette en place un nouveau jeu, un document a été écrit, décrivant point par point les éléments à mettre en place et à compléter. (Voir C) Il est nécessaire qu'il possède des notions avec le logiciel Unity pour être familier avec certains termes ; cependant, la procédure est rédigée avec un niveau de granularité d'explication assez fort.

2.2 Evolution des objectifs

Pendant le projet, les spécifications ont évolué, décrites section .

Comme il n'y a pas eu d'aléas pendant le développement, j'ai pu avoir le temps disponible pour développement des nouvelles fonctionnalités.

3 Analyse de risque

Durant le développement de l'application, et concernant le respect des spécifications, aucun échec n'est à relever.

Cependant, certaines tâches auraient pu être plus longues que prévus. Deux composantes ont été identifiées :

- Trouver un algorithme de recherche du noyau d'un jeu de Nim
- Proposer une architecture permettant d'implémenter son propre jeu de Nim

Ne pas réussir à implémenter un algorithme aurait été un échec, dans le sens où c'était le fruit du travail effectué au semestre 9, et l'aboutissement des recherches et des documentations que j'ai pu parcourir. Cependant, cela n'aurait pas été trop préjudiciable pour le développement de l'application. Seul le Tic-Tac-Nim utilise l'algorithme, puisque le noyau du Coin-Strip est calculé selon la situation proposée (voir 2.0.1 (Annexe B)) et le noyau du Star-Nim a pu être déterminé à la main.

Le deuxième point est par contre bien plus capital. C'est d'ailleurs pour cela que cette modélisation s'est étendue plus longtemps que prévu, pour aboutir à un modèle vraiment robuste.

Ensuite, j'ai essayé de développer les tâches du projet de manière suivante : si un retard est annoncé sur une des tâches, il est toujours possible de déléguer le travail à une autre équipe.

Par exemple, si j'ai pris du retard sur le développement des jeux, un autre développeur peut tout à faire mettre en place les différents écrans de menus, puisque ces scènes sont indépendantes. Il aurait fallu cependant définir quelques spécifications, notamment les données qui sont transmises entre les divers scènes (nom du jeu, difficulté de l'IA, etc...)

4 Suivi de projet

Mon projet comprenait 2 encadrants : Mr Esswein et Mr Billaut. Tout d'abord, un compte-rendu est envoyé toutes les semaines, comprenant les éléments qui ont été mise en place pendant la semaine et ceux à mettre en place pour la semaine suivante.

Nous avons défini, avec Mr Esswein, un rythme d'un rendez-vous par semaine (mercredi à 10 :15) pour discuter notamment du compte-rendu et des diverses avancées. Etant donné qu'un prototype d'application a été mis en place très rapidement, j'ai donc pu lui montrer, toutes les semaines, les modifications et changements effectués sur celui-ci.

De manière assez libre, j'allais à la rencontre de Mr Billaut pour également lui faire part de mes avancées, et recueillir son avis sur divers points de l'application.

Une première version de l'application a été envoyée le 13 février. Une deuxième a été envoyée le 8 mars, correspondant au rendez-vous de Mr Billaut avec les médecins du CHU Bretonneau de Tours, qui était intéressé par cette application.

5 Outils utilisés pour la gestion de projet

Divers logiciels et outils ont été utilisés pour mener à bien le projet :

- MindView m'a permis la mise en place des différents diagrammes de Gantt.
- Redmine (proposé par Polytech) est un outil de versionning de code. J'ai pu, à plusieurs reprises, revenir à une version antérieure du programme, après des changements qui ont pu entraîner des problèmes (erreur de compilation, comportements inattendus..).
- Astah et StarUML : outils de modélisation UML



Comptes rendus hebdomadaires

Compte rendu n°1 du 22/09/2016

Ce que j'ai fait :

- Premier rendez-vous avec les encadrants, définition des objectifs et des enjeux du projet
- Découverte des principaux jeux de Nim
- Définition de notions de bases : jeu impartial/partial, version misère/normal d'un jeu, nimber...
- Etudes des stratégies gagnantes sur certains de ces jeux (jeu de Northcott -> calcul de la somme de Nim)

Ce qui est prévu pour la semaine prochaine :

- Etude de la stratégie gagnante du Tic Tac Nim
- Compréhension totale de la stratégie gagnante du jeu de Northcott
- Début de rédaction d'un état de l'art

Compte rendu n°2 du 06/10/2016

Ce que j'ai fait :

- Rédaction de l'état de l'art
- Rédaction du cahier des spécifications – meilleure idée du design de la future application
- Premier prototype du Tic Tac Nim – implémentation d'algorithmes basiques (quel coup est autorisé, quand est-ce qu'un joueur a gagné...)

Ce qui est prévu pour la semaine prochaine :

- Peaufiner l'état de l'art et le cahier des spécifications
- Travail sur le prototype
- Réflexions sur une implémentation du programme pour une reprise future

Compte rendu n°3 du 13/10/2016

Ce que j'ai fait :

- Réflexion sur une architecture du système : diagramme de classe générique

- Réflexions sur l'interface Homme Machine de l'application : listing des différents écrans de jeu, des options à prévoir...
- Réflexions sur l'application en elle-même, notamment concernant la rétention d'un joueur (calculer un score final, proposer un historique des parties jouées...)
- Rédaction du cahier des spécifications et de l'état de l'art

Ce qui est prévu pour la semaine prochaine :

- Elaboration d'un diagramme de classe complet et mature et validation
- Validation et potentiel rajout de spécifications
- Continuer à définir les contours de l'IHM

Compte rendu n°4 du 20/10/2016

Ce que j'ai fait :

- Prise de rendez-vous avec Carl Esswein : revue des documents (état de l'art et cahier de spécifications), mise en place des priorités
- Réflexion sur un plan définitif de l'état de l'art
- Rédaction de l'état de l'art définitif

Ce qui est prévu pour la semaine prochaine :

- Rédaction des différentes parties de l'état de l'art
- Travail élaboré et mature sur la complexité de la création d'un graphe de n'importe quel jeu de Nim
- Regroupement définitif des connaissances sur les jeux de Nim

Compte rendu n°5 du 27/10/2016

Ce que j'ai fait :

- Validation du plan de l'état de l'art
- Rédaction de l'état de l'art : parties sur l'histoire et la présentation des 3 jeux de Nim étoffées
- Recherche de documents à la BU : utilisation de On Numbers and Games de Conway

Ce qui est prévu pour la semaine prochaine

- Rédaction des différentes parties de l'état de l'art
- Travaux sur la complexité et la résolution des 12 jeux de Nim
- Relecture et dépôt d'une première version du cahier de spécifications

Compte rendu n°6 du 03/11/2016

Ce que j'ai fait :

- Rédaction de l'état de l'art : partie sur les graphes des jeux
- Réflexion sur l'architecture du jeu (diagramme UML)
- Résolution du jeu Star Nim : énumération des états et détermination du noyau

Ce qui est prévu pour la semaine prochaine

- Rendez-vous avec A. Soukhal pour déterminer de l'avancée du projet
- Résolution d'autres jeux de Nim parmi les 12
- Relecture et correction des différentes parties sur l'état de l'art
- Démarrage du rapport de mi-semester

Compte rendu n°7 du 10/11/2016

Ce que j'ai fait :

- Validation de l'introduction et de la partie sur la présentation des jeux de Nim
- Dessin du graphe de tous les états du Star Nim
- Quelques explications sur le jeu de Northcott et Coin-Strip
- Rédaction du rapport de mi-semestre

Ce qui est prévu pour la semaine prochaine :

- Validation de la partie sur la présentation des 3 jeux de Nim (allumettes, Marienbad, Tic-Tac-Nim)
- Résolution de Wythoff et d'autres jeux de Nim
- Travail sur le rapport de mi-semestre

Compte rendu n°8 du 17/11/2016

- Finition des 3 parties de l'état de l'art (histoire et définition des jeux de Nim, exemples de 3 jeux de Nim : règles et stratégies à adopter et généralisation de la théorie autour des jeux de Nim)
- Résolution du jeu de Wythoff
- Travail sur l'architecture du système
- Rédaction du rapport de mi-semestre : intégration des spécifications et de l'analyse

Ce qui est prévu pour la semaine prochaine :

- Validation de l'état de l'art - éventuelles corrections
- Continuer sur les spécifications et l'analyse du système

Compte rendu n°9 du 24/11/2016

- Rendez-vous avec Mr Soukhal et Mr Billaut, pour discuter des spécifications.
- Rédaction poussée des objectifs, du contexte du projet
- Rendez-vous avec Mr. Esswein pour discuter de l'état de l'art
- Approfondissement sur la théorie des jeux de Nim.

Ce qui est prévu pour la semaine prochaine :

- Soumission d'une nouvelle version de l'état de l'art à Mr Esswein
- Correction des spécifications
- Travail sur la partie Analyse du PRD

Compte rendu n°10 du 01/12/2016

- Rendez-vous avec Mr Martineau pour déterminer la gestion du projet : ce qui est attendu (diagramme de Gantt prévisionnel du S10)
- Soumission d'une nouvelle version de l'état de l'art et du cahier de spécifications
- Rédaction des premiers éléments à mettre en place (partie Analyse)
- Travail sur le jeu de Hex, de Cram et Coin-Strip (en annexes du rapport)

Ce qui est prévu pour la semaine prochaine :

- Terminer le diagramme de Gantt prévisionnel, et soumission
- Rédiger les dernières annexes (sur les jeux de Nim du plateau)
- Mise en accord sur la partie Analyse
- Rédaction de la conclusion
- Relecture et correction du rapport (inclure la bibliographie)

Compte rendu n°11 du 22/12/2016

Ce que j'ai fait :

- Création d'un programme CSharp permettant la génération des 102 états du Tic-Tac-Nim et la recherche du noyau. Les 102 états sont correctement générés, mais le noyau obtenu n'est pas correct (le programme trouve 24 grilles, grille vide non incluse, alors qu'on doit en trouver 20, grille vide incluse) Ce qui est prévu pour la semaine prochaine

Ce qui est prévu pour la semaine prochaine :

- Corriger le programme pour obtenir les grilles correctes appartenant au noyau

Compte rendu n°12 du 30/12/2016

Ce que j'ai fait :

- Noyau généré correctement (avec et sans symétries). Création d'un projet Unity, mise en place du plateau abstrait.
- Réflexion sur l'architecture du système (Création d'une classe abstraite Game-Manager, dont héritera un manager par jeu)

Ce qui est prévu pour la semaine prochaine :

- Mise en place de la gestion de projet
- Continuer réflexion sur l'architecture

Compte rendu n°13 du 06/01/2016

Ce que j'ai fait :

- Création d'un espace Redmine (interne à Polytech) pour versionner le projet (SVN)
- Prise de rendez-vous avec Mr. Esswein pour déterminer la suite des événements
- Corrections mineures du placement des croix sur le Tic-Tac-Nim
- Début d'implémentation du Coin-Strip (algos et plateau de jeu)

Ce qui est prévu pour la semaine prochaine :

- Continuer l'implémentation du Coin-Strip
- Se former sur les tests Unity

Compte rendu n°14 du 12/01/2016

Ce que j'ai fait :

- Rendez-vous avec Mr Esswein
- Un modèle commun pour représenter les données d'un jeu de Nim était de type chaîne de caractère – on utilise désormais un tableau 2D de booléens
- Travail au niveau de la gestion des mouvements possibles
- Revue du diagramme de classes
- Formation sur les tests sous Unity

Ce qui est prévu pour la semaine prochaine :

- Continuer l'implémentation du Tic-Tac-Nim et du Coin-Strip avec les changements effectués pour avoir 2 jeux fonctionnels (ce n'est pas le cas actuellement)
- Mettre en place des exceptions (mouvement possible, etc)
- Revue de l'avancée avec Mr Billaut et Mr Esswein

Compte rendu n°15 du 20/01/2016

Ce que j'ai fait :

- Rendez-vous avec Mr Esswein pour discuter de l'avancée et de la modélisation UML
- Rendez-vous avec Mr Billaut pour discuter de l'avancée – petites corrections à effectuer
- L'implémentation du Tic-Tac-Nim et du Coin-Strip ont l'air fonctionnels
- Revue du diagramme de classes

Ce qui est prévu pour la semaine prochaine :

- Revoir le diagramme de classe, pour expliciter certaines notions (Mouvement, Partie) et le rendre plus clair et accessible.
- Génération d'un premier prototype basique des 2 jeux

Compte rendu n°16 du 27/01/2016

Ce que j'ai fait :

- Rendez-vous avec Mr Martineau pour discuter de la gestion du projet. Retour sur mes questionnements vis-à-vis de mon planning et des périodes de tests
- Mise en place de tests d'intégration : simulation d'une partie de Tic-Tac-Nim
- Revue du diagramme de classes
- Quelques corrections diverses du programme : UI adapté à une résolution 16/9, le mouvement temporaire (placement ou déplacement d'une croix) est mis en valeur
- Création d'une scène Menu permettant le choix et la difficulté d'un jeu (pour la génération d'un exécutable)

Ce qui est prévu pour la semaine prochaine :

- Continuer les tests d'intégration
- Rendez-vous avec Mr Esswein pour effectuer une revue du diagramme de classes.
- Génération d'un premier prototype basique des 2 jeux

Compte rendu n°17 du 02/02/2016

Ce que j'ai fait :

- Génération d'un prototype des 2 jeux, avec un menu succin. Rendez-vous avec Mr Esswein et Mr Billaut pour effectuer des retours sur ce premier prototype
- Mise en place de tests : génération d'une partie de CoinStrip, vérification d'une partie avec une IA en mode expert
- Quelques modifications dans le code : séparation des Button et Image dans une classe GameGUIManager, et non plus dans les managers respectifs des différents jeux

Ce qui est prévu pour la semaine prochaine :

- Correction éventuelle du diagramme de classe
- Continuer les tests d'intégration

Compte rendu n°18 du 09/02/2016

Ce que j'ai fait :

- Mise en place d'un mode 2 joueurs
- Création de divers panels (voulez-vous quitter, fin du jeu)

- Tous les éléments du jeu sont chargés dynamiquement (dans GameGUIManager)
- Les deux scènes sont plus jolies

Ce qui est prévu pour la semaine prochaine

- Effectuer des tests sur le Star Nim
- Rendre l'interface toujours plus accessible
- Tester l'application sur diverses résolutions

Compte rendu n°19 du 16/02/2016

Ce que j'ai fait :

- Rendez-vous avec Mr Ramel pour déterminer de la qualité de la modélisation et du code
- Mise en place d'une doc Doxygen
- Rédaction d'un document décrivant la procédure pour implémenter son propre jeu de Nim
- Rendez-vous avec Mr Esswein pour implémenter de nouvelles fonctionnalités

Ce qui est prévu pour la semaine prochaine :

- Fonctionnalités à implémenter : création d'une scène « à propos », bouton « infos » pendant un jeu
- Rendre le design moins austère, plus chaleureux
- Tester sur différents émulateurs Android

Compte rendu n°20 du 23/02/2016

Ce que j'ai fait :

- Mise en place d'un menu (boutons Choix du jeu, Utilisateur et A propos)
- Création d'une scène A propos, contenant un texte déroulant
- Ajout d'infos pour chaque jeu

Ce qui est prévu pour la semaine prochaine :

- Rédaction et validation du texte dans la scène A propos
- Rendre le design moins austère, plus chaleureux
- Mise en place des éléments d'un utilisateurs : nom, victoires et défaites pour chaque jeux...

Compte rendu n°21 du 02/03/2016

Ce que j'ai fait :

- Scène « Utilisateur » : nom, victoires et défaites totales et pour chaque jeu
- Quelques corrections : affichage des règles lorsque le jeu est lancé pour la première fois, correctifs graphiques divers
- Rédaction du texte « A propos »

Ce qui est prévu pour la semaine prochaine :

- Validation du texte dans la scène « A propos »
- Finition de la scène « Utilisateur »
- Documentation du code plus poussée, et manuel de l'utilisateur plus détaillé

Compte rendu n°22 du 09/03/2016

Ce que j'ai fait :

- Validation des différents textes (règles et A propos) par Mr Billaut

- Création d'une scène de Login, l'application comprends désormais une gestion basique des utilisateurs
- Documentation du code et rédaction du document de synthèse de la gestion de projet

Ce qui est prévu pour la semaine prochaine :

- Terminer la documentation du code
- Rédaction du rapport
- Gestion des utilisateurs plus poussées (administration des comptes)

Compte rendu n°23 du 16/03/2016

Ce que j'ai fait :

- Avancer sur la documentation du code
- Ajout d'une scène Administrateur, qui gère les différents comptes du jeu.
- Les statistiques sont maintenant sous-fractionnées par jeu ET par difficulté de l'adversaire

Ce qui est prévu pour la semaine prochaine :

- Terminer la documentation du code
- Rédaction du rapport

Compte rendu n°24 du 24/03/2016

Ce que j'ai fait :

- Fin de la documentation du code
- Correction de la scène Administrateur
- Modélisation UML Globale
- Création du storyboard de l'application

Ce qui est prévu pour la semaine prochaine :

- Relecture de la documentation
- Fin de la modélisation UML et du storyboard
- Préparation à l'évaluation
- Rédaction du rapport

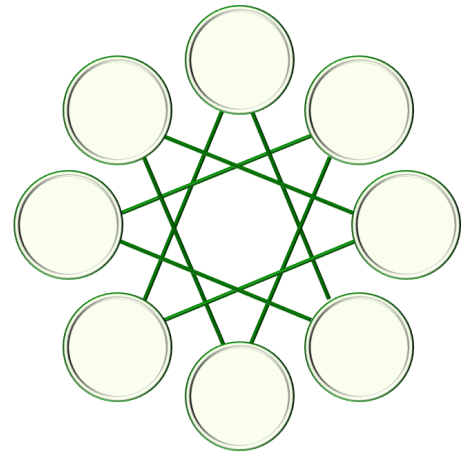
Jeux de Nim

Arthur Capo

Encadrement : Jean-Charles Billaut et Carl Esswein

Les jeux de Nim

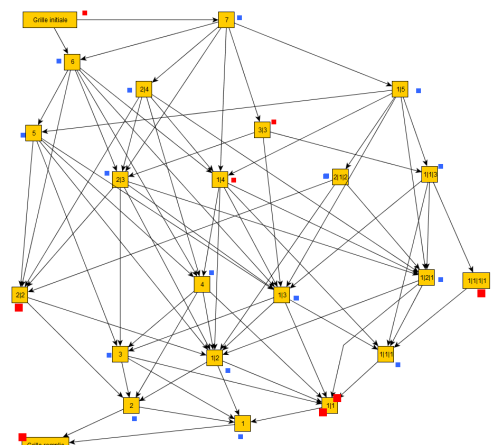
Un jeu de Nim possède plusieurs caractéristiques : Il se joue à 2 joueurs, et pas de partie nulle possible. Chaque joueur possède la même série de mouvements, et le hasard n'intervient pas. Par exemple, le jeu du Star Nim se joue sur le plateau ci-contre. Chaque joueur place, à tour de rôle : ou un pion sur une case, ou deux pions si et seulement si les 2 cases choisies sont reliées par un segment. Le joueur qui pose le dernier pion a gagné.



Plateau du Star Nim

Résolution d'un jeu de Nim

Pour résoudre un jeu de Nim, l'objectif est de déterminer le graphe du jeu. Ce graphe possède un noyau, correspondant à un sous-ensemble de coups, qui définit la stratégie gagnante du jeu : si on est dans une situation favorable (hors du noyau), alors on peut toujours jouer un coup pour le rejoindre et, ainsi de suite, arriver jusqu'à la grille finale.



Graphe du jeu de Star Nim

Objectifs

Le but de ce PRD consiste au développement d'une application smartphone sur les jeux de Nim. Les objectifs sont :

- Proposer deux jeux de Nim jouables
- Proposer au joueur de jouer contre un humain ou une intelligence artificielle
- Fournir une architecture évolutive



Ecran de jeu

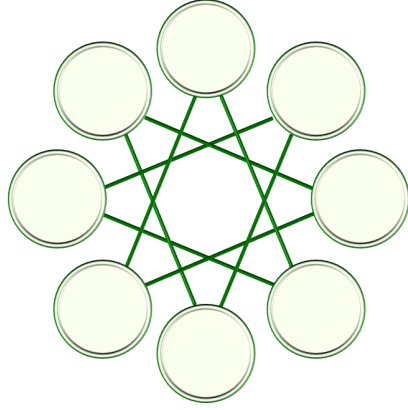
Jeux de Nim

Arthur Capo

Encadrement : Jean-Charles Billaut et Carl Esswein

Les jeux de Nim

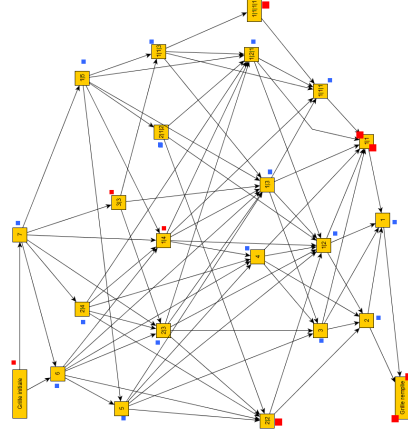
Un jeu de Nim possède plusieurs caractéristiques : Il se joue à 2 joueurs, et pas de partie nulle possible. Chaque joueur possède la même série de mouvements, et le hasard n'intervient pas. Par exemple, le jeu du Star Nim se joue sur le plateau ci-contre. Chaque joueur place, à tour de rôle : ou un pion sur une case, ou deux pions si et seulement si les 2 cases choisies sont reliées par un segment. Le joueur qui pose le dernier pion a gagné.



Plateau du Star Nim

Résolution d'un jeu de Nim

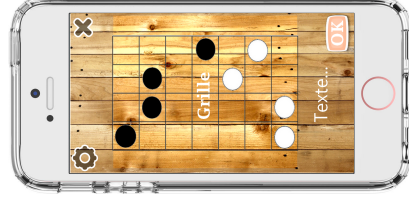
Pour résoudre un jeu de Nim, l'objectif est de déterminer le graphe du jeu. Ce graphe possède un noyau, correspondant à un sous-ensemble de coups, qui définit la stratégie gagnante du jeu : si on est dans une situation favorable (hors du noyau), alors on peut toujours jouer un coup pour le rejoindre et, ainsi de suite, arriver jusqu'à la grille finale.



Graphe du jeu de Star Nim

Objectifs

- Le but de ce PRD consiste au développement d'une application smartphone sur les jeux de Nim. Les objectifs sont :
- Proposer deux jeux de Nim jouables
 - Proposer au joueur de jouer contre un humain ou une intelligence artificielle
 - Fournir une architecture évolutive



Ecran de jeu

Jeux de Nim

Résumé

Application sur smartphone proposant plusieurs jeux de Nim, jouable à 2 ou contre une intelligence artificielle

Mots-clés

jeu, Nim, smartphone

Abstract

Development of multiple Nim games on smartphone, playable by 2 players or against an artificial intelligence

Keywords

game, Nim, smartphone

Tuteurs académiques

Jean-Charles BILLAUT
Carl ESSWEIN

Étudiant

Arthur CAPO (DI5)