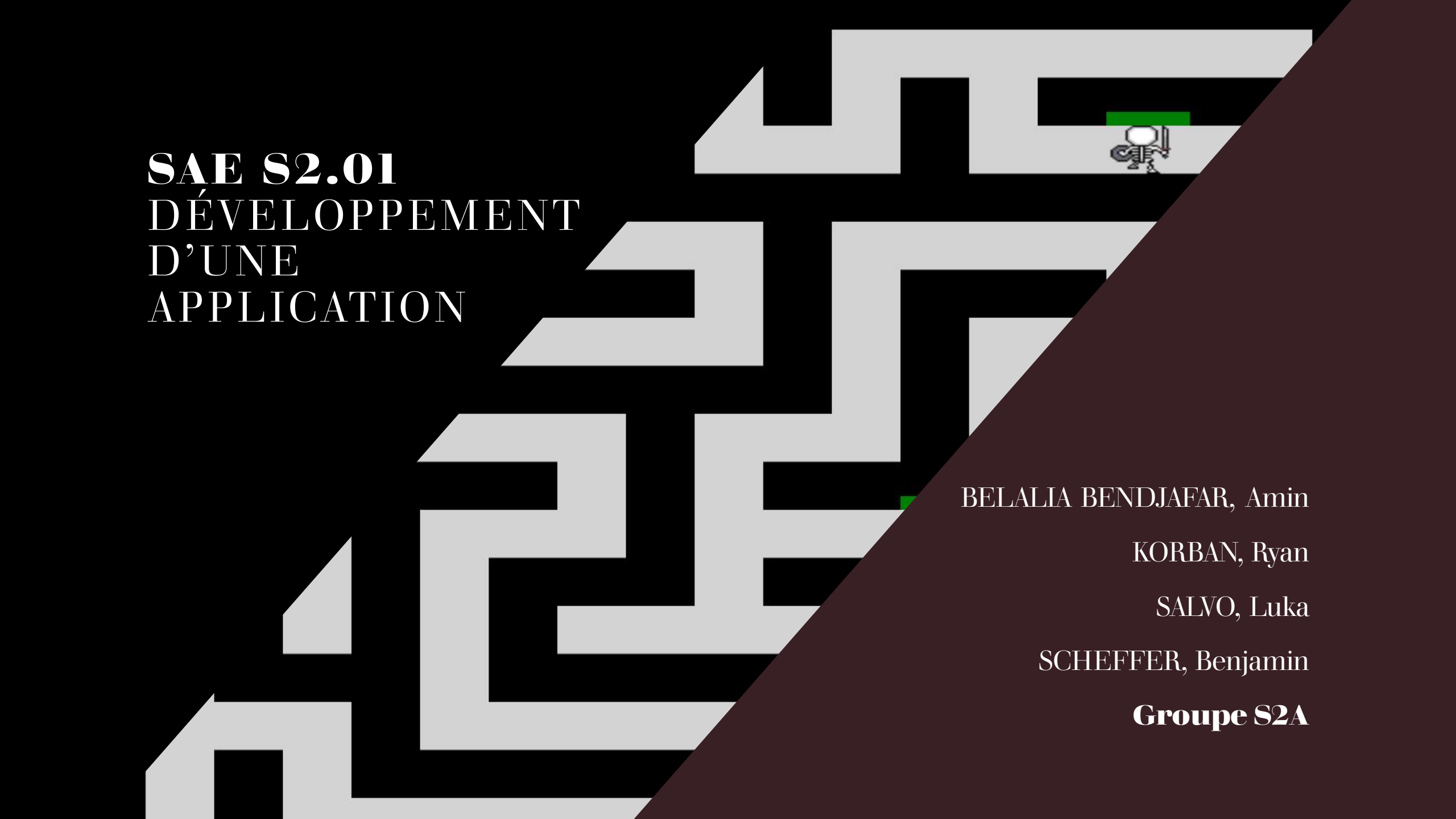
SAE S2.01
DÉVELOPPEMENT D'UNE APPLICATION



BELALIA BENDJAFAR, Amin

KORBAN, Ryan

SALVO, Luka

SCHEFFER, Benjamin

Groupe S2A

Sommaire

Introduction – Fonctionnalités du jeu

- Jeu avec comme carte un labyrinthe
- Présence de monstres pouvant attaquer le joueur
- Monstres ayant un déplacement intelligent
- Présence de cases piégés sur la carte
- Présence d'une amulette sur la carte pour al victoire
- Fin du jeu via la mort du personnage ou victoire

Présentation des versions

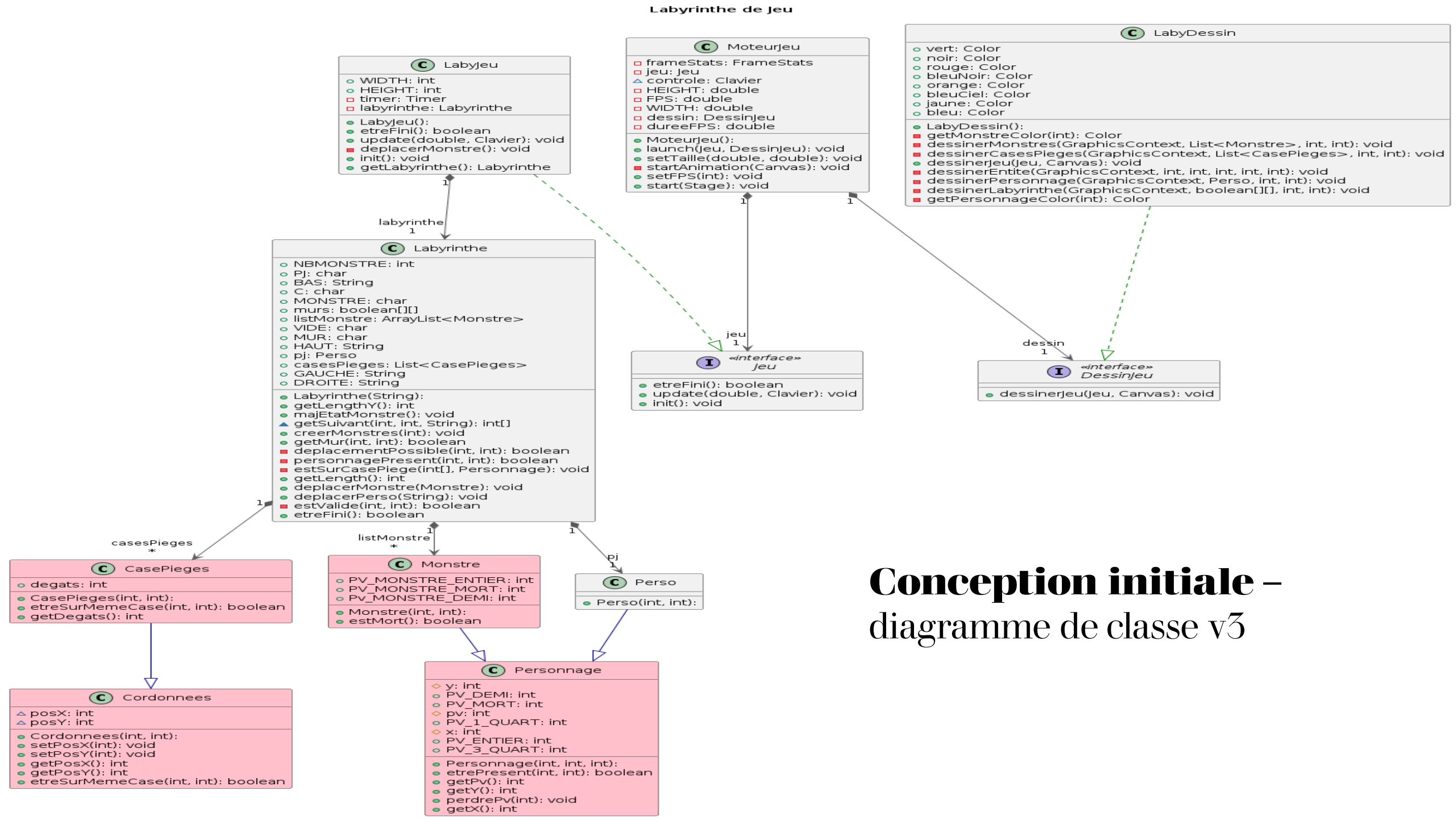
>**Version 1 et 2 – Travail préparatoire**
Affichage du labyrinthe
Détection des collisions
Déplacement des monstres

>**Version 3 – 1ère Itération**
Génération de monstres
Déplacement aléatoire des monstres
Gestion de la mort des monstres
Implémentation des cases piégés

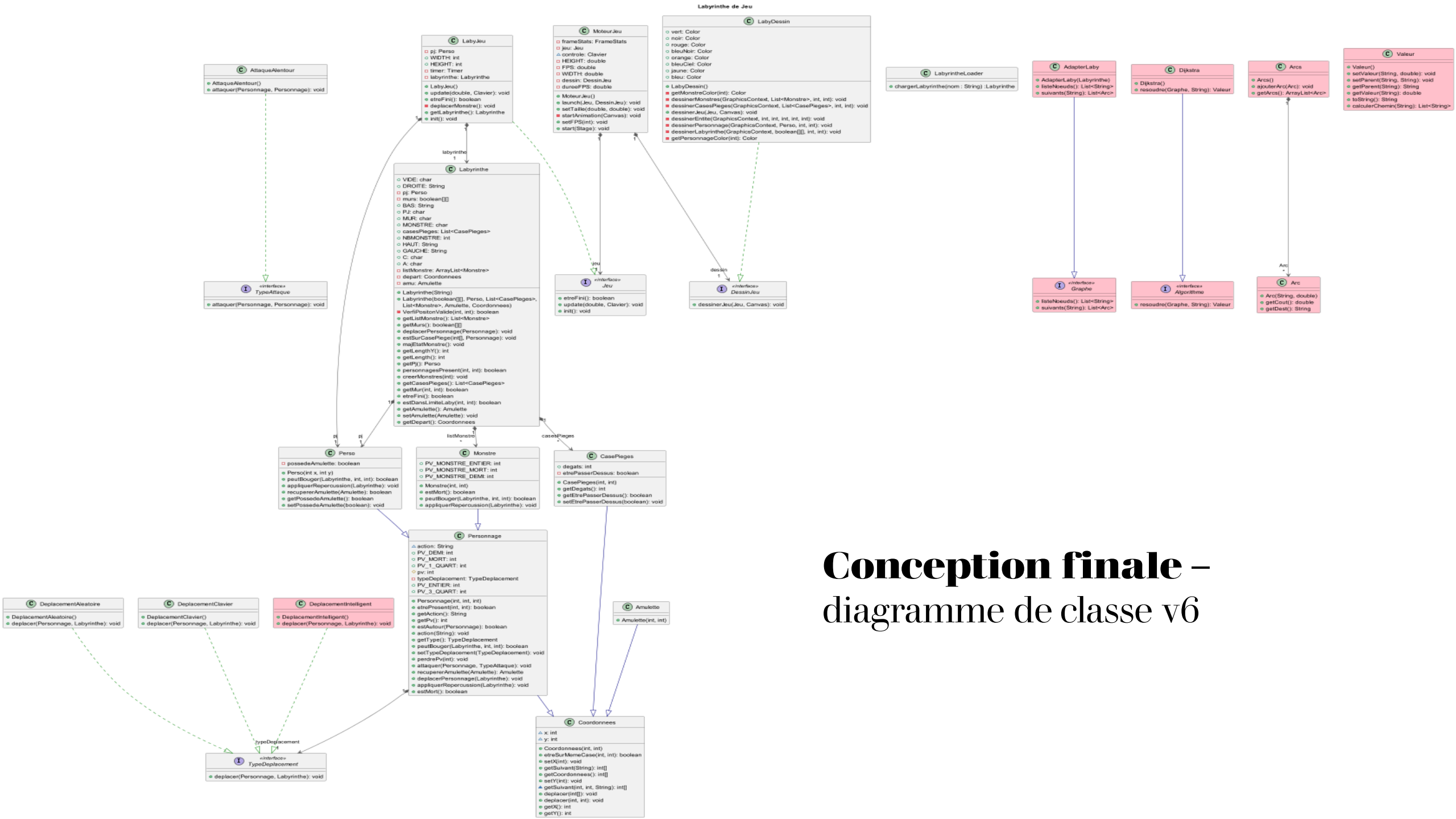
>**Version 4 – 2ème Itération**
Attaque des monstres et joueurs
Travail sur la conception

>**Version 5 – 3ème Itération**
Gestion et acquisition de l'amulette
Fin du jeu (mort ou victoire)

>**Version 6 – 4ème Itération**
Génération automatique de labyrinthe
Monstres intelligents
Barre de vie et affichage amélioré par des sprites



Conception initiale –
diagramme de classe v3



Conception finale –
diagramme de classe v6

Partage des tâches

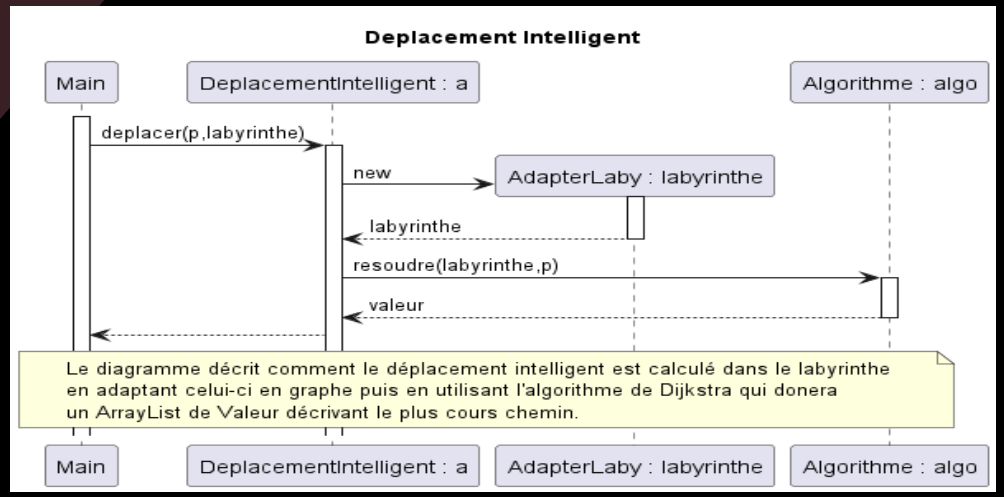
>**Amin**
Diagrammes de séquence, tests et java doc – v3
Conception et code fonctionnalité attaque des monstres + refactorisation – v4
Fin de jeu en cas de victoire + refactorisation – v5
Conception et code comportement intelligent du monstre – v6

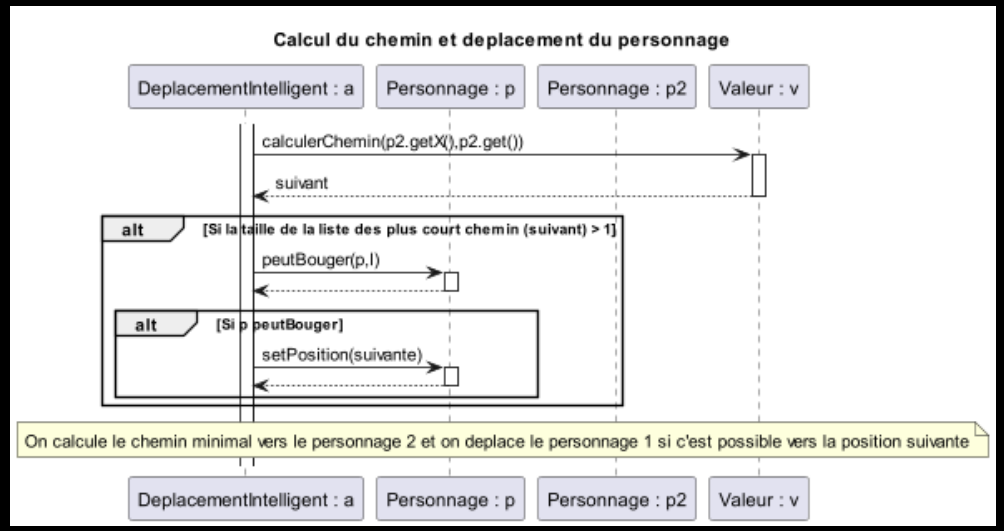
>**Ryan**
Conception et code du déplacement aléatoire des monstres – v3
Conception et code fonctionnalité attaque du joueur + refactorisation – v4
Acquisition de l'amulette + refactorisation – v5
Conception et code génération automatique labyrinthe– v6

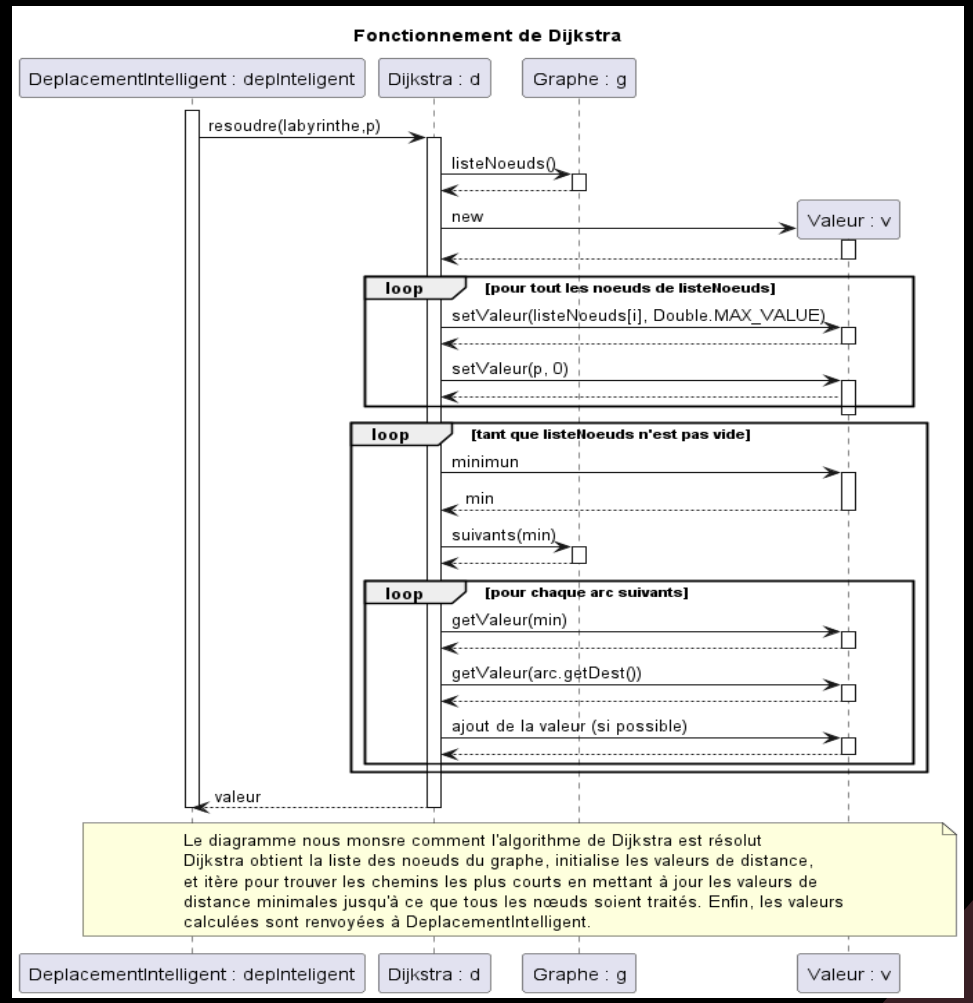
>**Luka**
Gestion des points de vie, mort des monstres et modification d'affichage – v3
Conception et tests fonctionnalité attaque des monstres + refactorisation – v4
Fin de jeu en cas de mort + refactorisation – v5
Conception et test comportement intelligent du monstre – v6

>**Benjamin**
Conception et tests fonctionnalité cases piégés – v3
Finalisation cases pieges, conception et tests fonctionnalité attaque des monstres + refactorisation – v4
Mise en place de l'amulette + refactorisation – v5
Conception et test génération automatique labyrinthe + barre de vie et sprites – v6

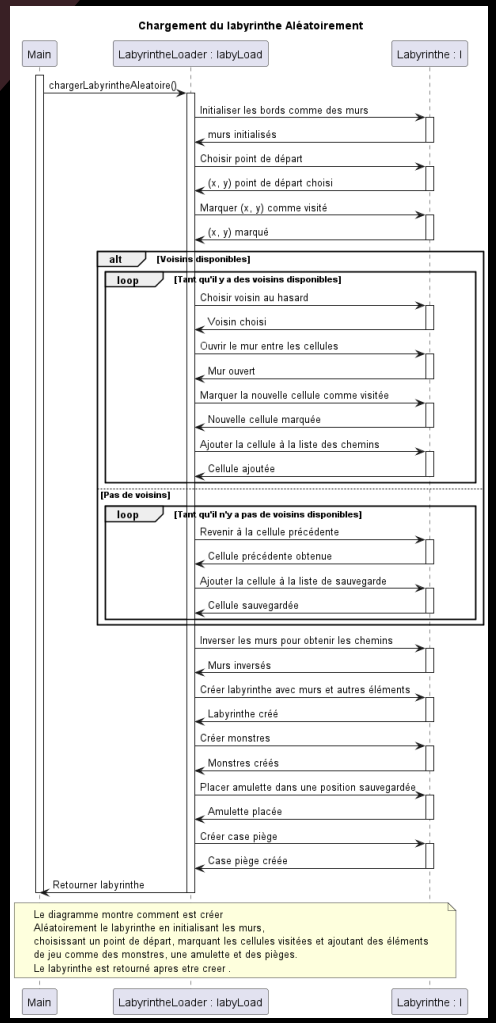
Détail sur la conception – Déplacement intelligent







Détail sur la conception – Génération aléatoire



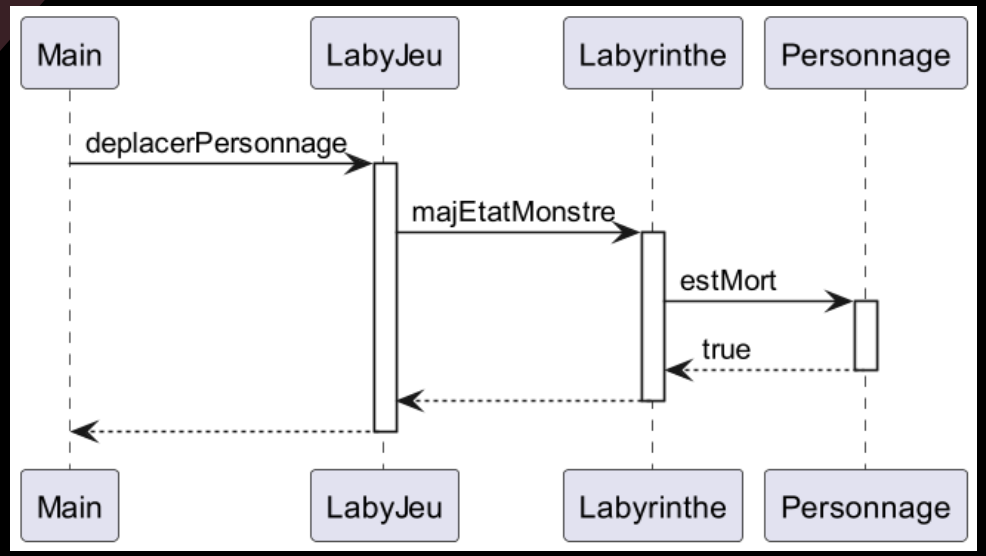
Algorithme de fusion aléatoire de chemins

Cet algorithme, utilise une propriété des labyrinthes parfaits précédemment énoncée :

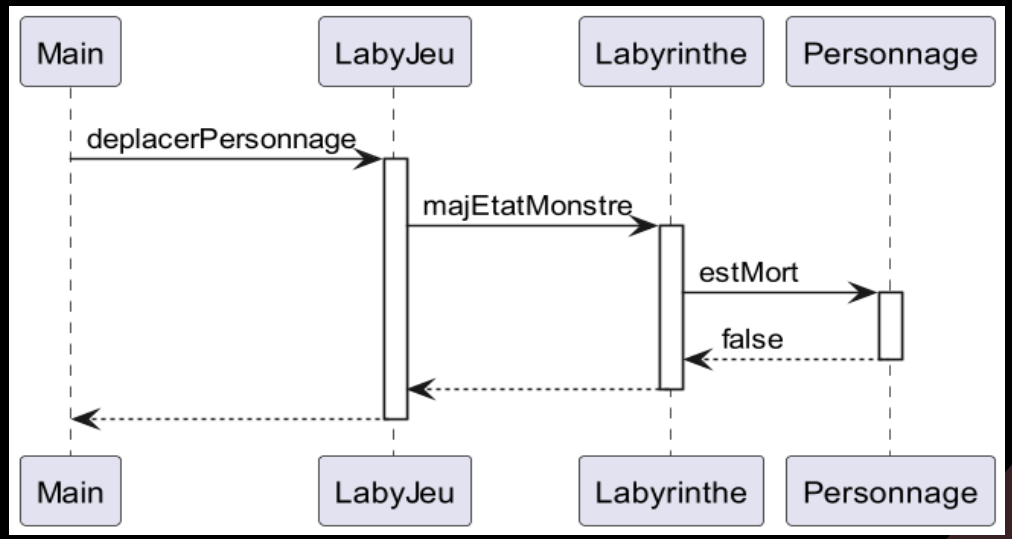
Chaque cellule est reliée à toutes les autres, et ce, de manière unique.
Il fonctionne en fusionnant progressivement des chemins depuis la simple cellule jusqu'à l'obtention d'un chemin unique, il suit donc une approche ascendante.

- L'algorithme associe une valeur unique à chaque cellule (leur numéro de séquence, par exemple) et part d'un labyrinthe où tous les murs sont fermés.
- A chaque itération, on choisit un mur à ouvrir de manière aléatoire.
- Lorsqu'un mur est ouvert entre deux cellules adjacentes, les deux cellules sont liées entre elles et forment un chemin.
- L'identifiant de la première cellule est recopié dans la seconde.
- À chaque fois que l'on tente d'ouvrir un mur entre deux cellules, on vérifie que ces cellules ont des identifiants différents.
- Si les identifiants sont identiques, c'est que les deux cellules sont déjà reliées et appartiennent donc au même chemin. On ne peut donc pas ouvrir le mur.
- Si les identifiants sont différents, le mur est ouvert, et l'identifiant de la première cellule est affecté à toutes les cellules du second chemin.

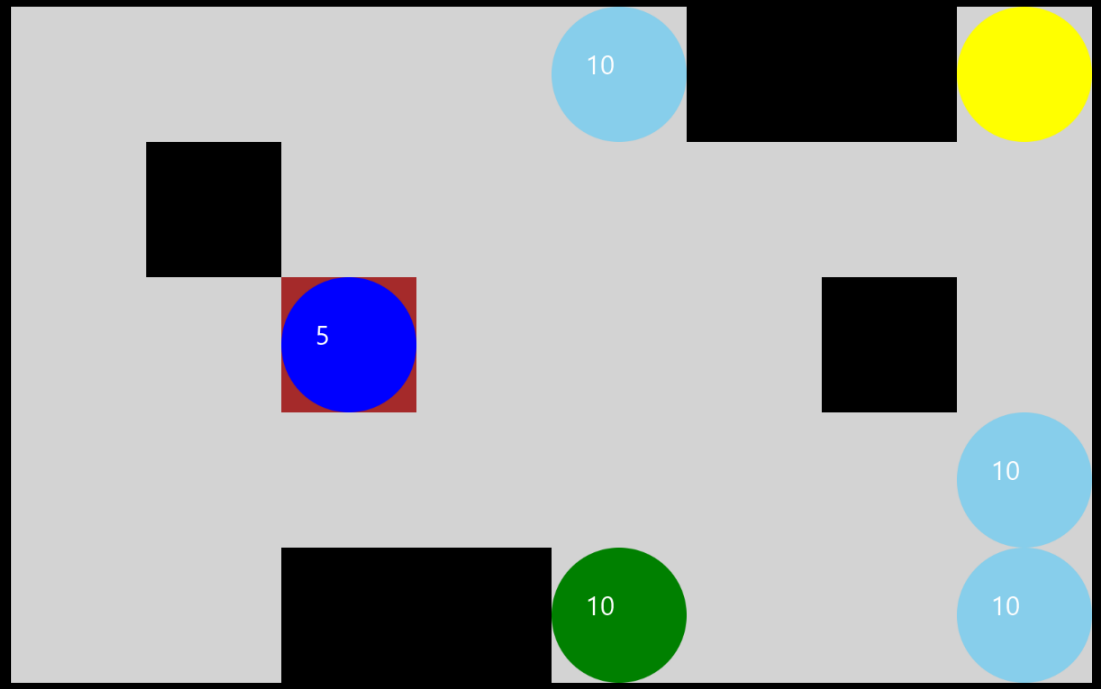
Détail sur la conception – Mort du monstre



Mise en place de l'Iterator au lieu d'un ArrayList
→ Eviter des conflits entre le Timer et liste des monstres.



Point sur fonctionnalité – Sprites



Conclusion