Hassan Meite
Alexis Wamster

Rapport SAE21_2022

Thésée perdu dans le labyrinthe

Sommaire

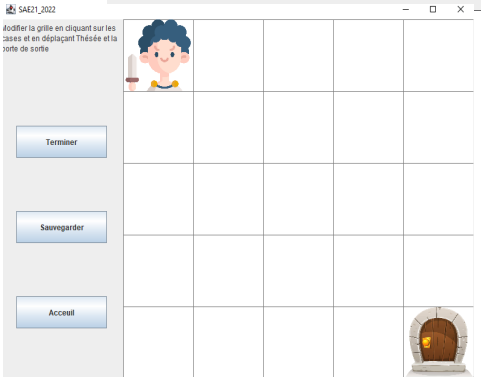
À propos du jeu

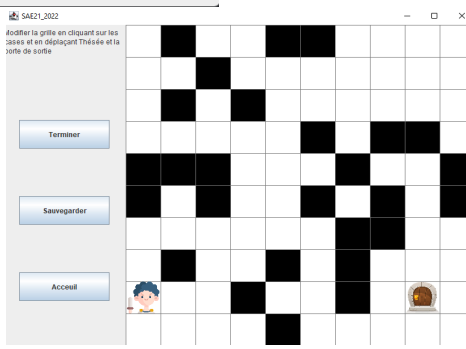
Nous avons développé un jeu en Java inspiré du célèbre mythe du labyrinthe de Thésée pour notre deuxième projet de développement de l'année. Le jeu est basé sur le principe du fil d'Ariane, où Thésée s'est égaré dans un labyrinthe et a utilisé un fil pour en sortir. Les règles sont simples : le joueur débute avec le choix du mode qu'il souhaite effectuer, modélisé par une interface facile à utiliser. Une fois le mode choisi, c'est-à-dire, la grille aléatoire ou une grille vide et les dimensions entrées, il peut ensuite modifier le labyrinthe pour aider Thésée à trouver la sortie. Le joueur peut effectuer le parcours du labyrinthe manuellement ou de manière aléatoire et a la possibilité de sauvegarder sa partie en cours, y compris les modifications apportées au labyrinthe. L'utilisateur peut créer une grille de jeu allant jusqu'à 255 x 255.
Ce rapport explique en détail comment nous avons réalisé ce jeu, ses fonctionnalités et enfin notre impression et nos remarques vis-à-vis de sa réalisation.

Description

Notre programme comporte toutes les fonctionnalités demandées dans le sujet de SAé. Lorsque vous lancez le programme, vous pouvez choisir comment générer votre grille.

Vous pouvez le faire à partir d'un fichier, d'une grille vide avec une taille spécifié ou d'une grille aléatoire avec une taille spécifié.

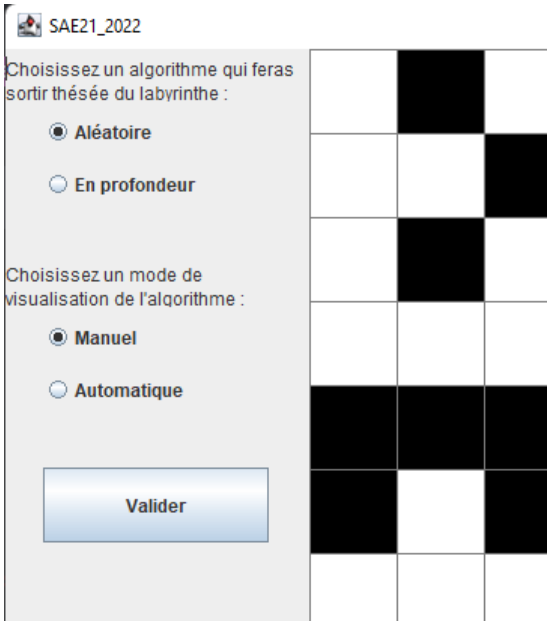




Vous pouvez ensuite personnaliser votre grille à volonté.

- pour changer la couleur d'une case, il vous suffit de cliquer dessus
- pour changer la position de thésée ou de la sortie, il vous suffit de cliquer sur l'une de ces deux case et de déplacer cette case ou vous voulez sans relâcher le clic

Vous pouvez sauvegarder votre grille à tout moment de la personnalisation ce qui vous permet de créer plusieurs variations de la même grille.
Si la grille ne vous convient pas vous pouvez retourner à l'accueil. Ou si au contraire vous avez terminé vos modifications vous pouvez passer à l'étape suivante.

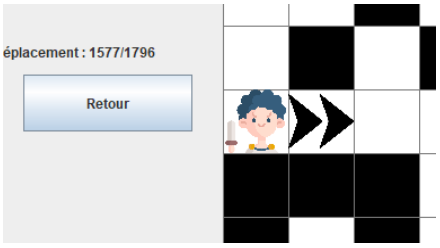


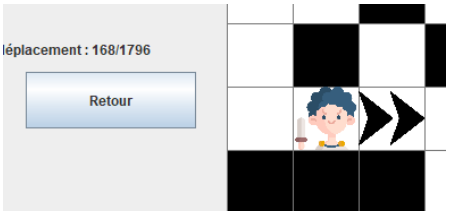
Lors de cette étape, vous ne pouvez plus personnaliser la grille.
Si jamais vous voulez la personnaliser de nouveau, il vous suffit de cliquer sur le bouton Retour.

Cette étape vous permet de choisir quel algorithme vas utiliser thésée pour sortir du labyrinthe et comment vous voulez le visualiser.

Le mode Automatique vous donnera directement le résultat. Ce résultat représente une moyenne si vous choisissez l'algorithme aléatoire.

Le mode automatique vous permet de visualiser chaque déplacement de thésée et aussi sa prochaine position. Vous pouvez avancer et reculer dans le temps à volonté. Il est à noter que vous pouvez visualiser les déplacements de thésée même lorsque votre labyrinthe n'admet aucune solution pour sortir.





Si vous avez terminé, vous pouvez revenir en arrière pour choisir un autre algorithme, ou bien revenir à l'accueil pour totalement changer de labyrinthe.

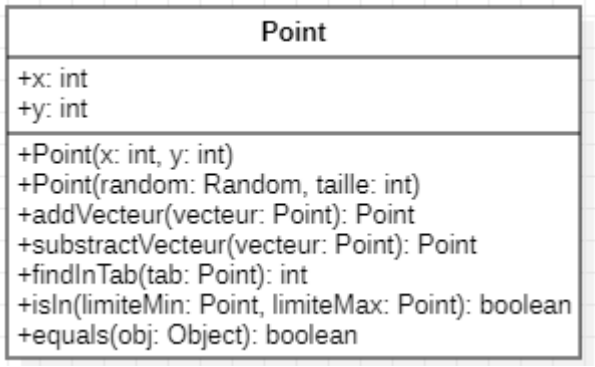
Fonctionnement du programme

1. Détails de chaque classe

Dans les diagrammes ci-dessous, les attributs et méthodes statiques ne sont pas soulignés car StarUML ne le permet pas.
Les tableaux à 2 dimensions ne comportent pas de crochets [][] car StarUML ne le permet pas.

- Point

La classe point permet de représenter des objets mathématiques qui possèdent des coordonnées. Dans notre programme, elle est majoritairement employée pour représenter des points dans un espace 2D. Mais elle est aussi utilisée pour représenter des vecteurs 2D.



Elle possède 2 attribut (x et y) modifiable et lisible à partir de n'importe quelle partie du programme.

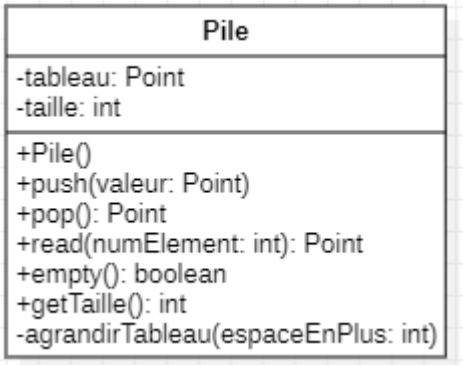
Elle possède 2 constructeurs qui permette de créer un point à des coordonnées précise ou aléatoire dans une zone délimitée par une taille.

à ce point, on peut ajouter ou soustraire un vecteur.

On peut aussi vérifier si il est dans un tableau et retourner l'indice auquel il se trouve.
On peut vérifier si il est dans une zone rectangulaire délimité par deux points
Et on peut vérifier si deux points sont égaux.

- Pile

La classe Pile représente une pile de points. Elle permet dans le programme de stocker toutes les différentes positions de thésée lorsque celle ci parcours le labyrinthe



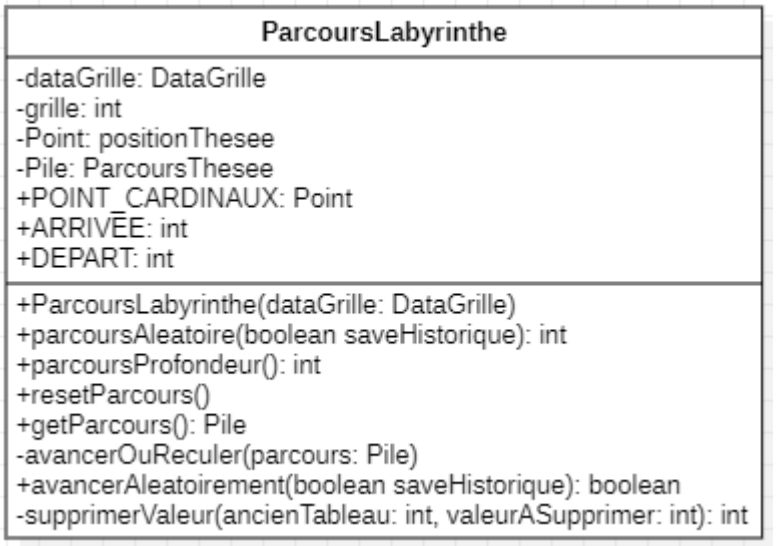
Elle possède un tableau de point (la pile), et une taille qui permet de savoir combien de points sont dans la pile.

Elle possède des méthode très classique comme push, pop et empty.
Elle possède aussi une méthode read qui permet de lire n'importe quelle valeur dans la pile sans la dépiler.

Une méthode getTaille() pour récupérer le nombre de valeurs qui se trouve dans la pile.
Et agrandirTableau() qui permet d'agrandir la pile lorsque celle-ci atteint la capacité.*

- ParcoursLabyrinthe

Cette classe Permet de fabriquer une pile qui contiendras toutes les positions de thésée lors de son parcours du labyrinthe



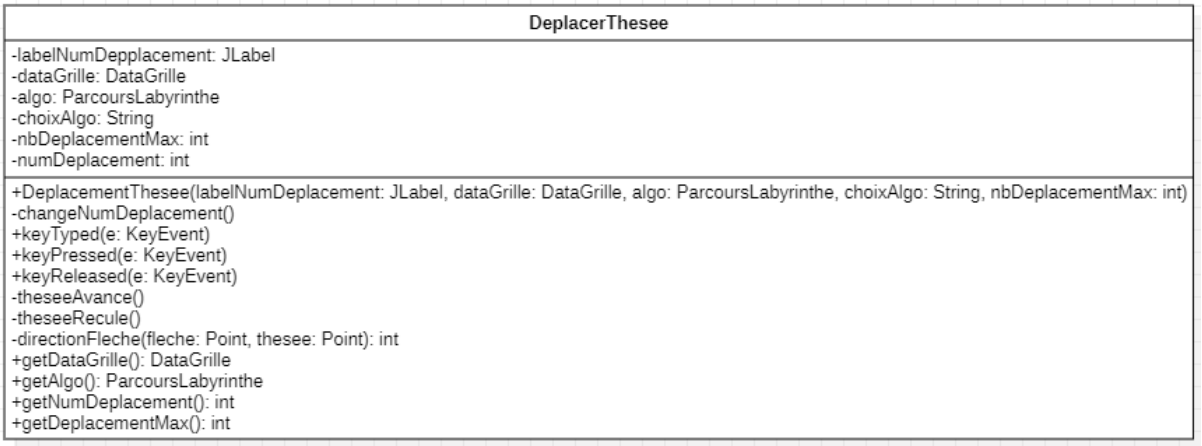
Elle contient toutes les information de la grille (datagrille);
un tableau entier qui est une copie de la grille réelle;
la position de thésée actuel;
toutes les position de thésée lors de son parcours;
et des constantes qui correspondent au vecteurs pour se déplacer dans 4 directions différentes, la valeur du point de départ et du point d'arrivée.

Elle possède également plusieurs méthodes:

- un constructeur
- une méthode qui simule un parcours aléatoire et qui renvoie le nombre de déplacement effectué par thésée lors de celui- ci. On peut choisir ou non de sauvegarder les positions de thésée dans l'attribut Pile.
- une méthode similaire mais qui cette fois simule un parcours en profondeur.
- une méthode qui vide l'historique des position de thésée
- une méthode qui renvoie l'historique des position de thésée
- une méthode avancerOuReculer() qui fait bouger thésée d'une case lors de son parcours en profondeur
- une méthode avancerAleatoirement() qui fait bouger thésée d'une case lors de son parcours en aléatoire
- Et une méthode qui permet de supprimer une valeur dans un tableau entier. (elle permet de savoir qu'elle déplacement aléatoire n'a pas encore essayé thésée)

- DeplacerThesse

Cette classe est un contrôleur qui fait évoluer graphiquement thésée sur la grille lorsque les flèches sont pressées lorsque l'utilisateur choisit le mode de visualisation manuel.



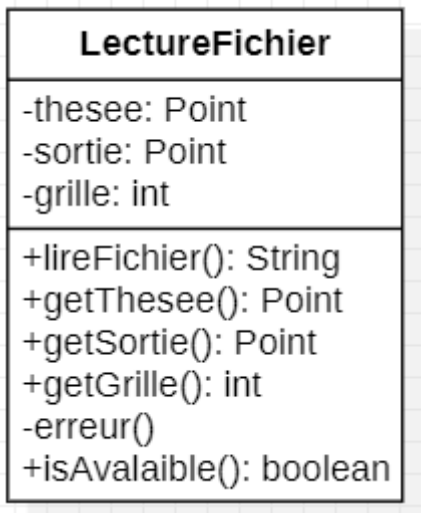
Cette classe contient des attributs:

- un JLabel qui indique combien de déplacement à effectuer thésée
- Des informations sur la grille
- L'historique des déplacement effectué par thésée
- L'algorithme qui à été choisis
- Le nombre de déplacement que thésée va mettre
- Et le nombre de déplacement qu'elle à déjà effectué

Elle contient des méthodes comme:

- un constructeur
- changeNumDeplacement() qui met à jour le texte du JLabel avec le nombre de déplacement qu'à fait thésée
- les méthodes keyTyped() et keyReleased() qui sont obligatoire mais qui ne font rien
- keyPressed() appelle la méthode theseeAvance() ou theseeRecule() en fonction de la flèche pressée.
- theseeAvance() et theseeRecule() fait avancer et reculer thésée et son prochain déplacement graphiquement. Cela met aussi à jour l'attribut labelNumDepplacement.
- directionFleche() renvoie la direction dans laquelle doit être orientée la flèche du prochain déplacement de thésée en fonction de la position de thésée et de celle de son prochain déplacement.
- Enfin les méthodes getAlgo(), getNumDeplacement(), et getDeplacementMax() permettent de transmettre les données de cette classe.

- LectureFichier



Cette classe permet de lire un fichier bits à bits et d'y interpréter des données qui serviront pour fabriquer une grille.
elle contient les attributs:

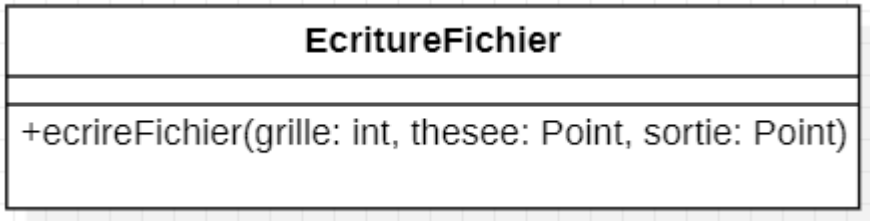
- thesee pour stocker la position de thésée
- sortie: pour stocker la position de la sortie
- grille: pour stocker la couleur des cases

elle contient les méthodes:

- lireFichier() utilise un JFileChooser pour que l'utilisateur choisisse un fichier. puis stocke si possible à partir de ce fichier les données de la grille qui correspond. Elle renvoie sous forme de string le message d'erreur s' il y en a.
- getThesee(), getSortie() et getGrille(), renvoie les données des attributs respectifs
- erreur() met tout les attribut à null lorsqu'il y a eu une erreur dans la lecture du fichier
- isAvailable() vérifie si il y a bien des données dans les attributs

- **EcritureFichier**

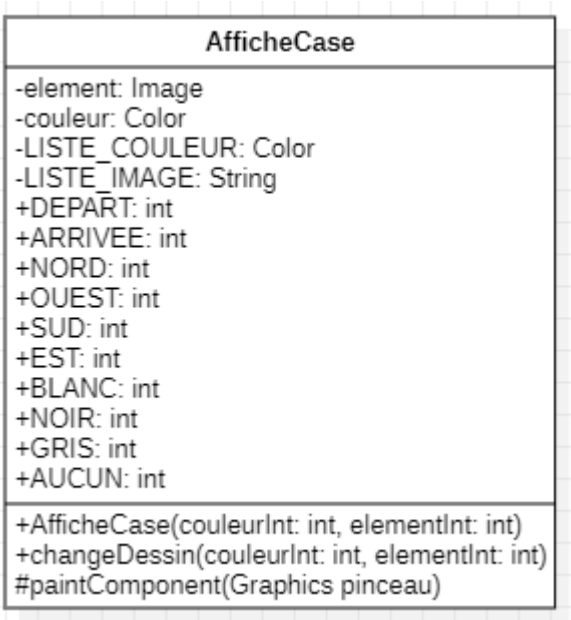
Cette classe permet de stocker les données d'une grille dans un fichier



La méthode ecrireFichier() utilise un JFileChooser pour que l'utilisateur choisisse ou sauvegarder sa grille, ensuite la méthode écrit les données qu'il faut dans le fichier.

- AfficheCase

Cette classe dessine une case. elle hérite de JComponent



Elle contient les attributs:
- element : qui contient l'image qui est présente dans la case
- couleur : qui contient la couleur de la case
- et des constantes qui permettent de choisir facilement depuis une autre classe l'apparence de la case.

elle des méthodes:
- un constructeur
- changeDessin() pour changer l'image ou la couleur de la case
- et PaintComponent pour redessiner automatiquement la case en cas de redimensionnement de la fenêtre.

- EvenementSouris

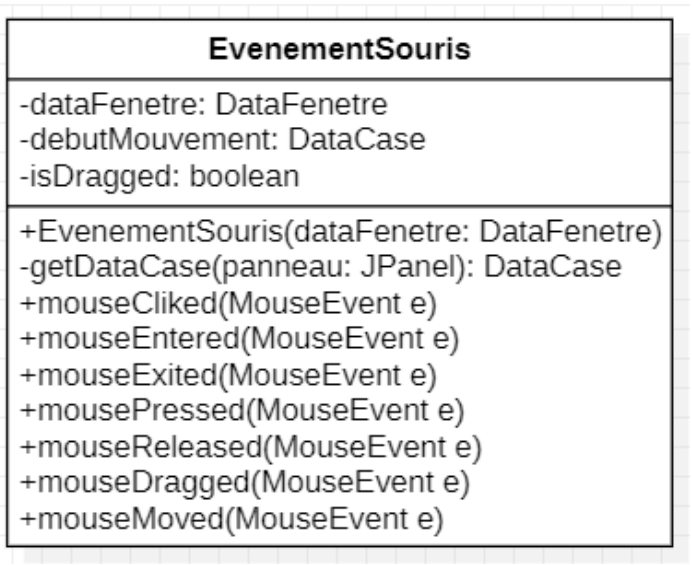
Cette classe qui implémente MouseListener et MouseMotionListener permet la personnalisation de la grille. Elle permet de déplacer thésée, la sortie et changer la couleur des cases.

elle contient les attributs:

- dataFenetre qui donne des information sur la grille entière
- debutMouvement qui contient les données de la case sur laquelle l'utilisateur à commencé son clic.
- Et isDragged qui indique si l'utilisateur est en train de déplacer une case ou non.

elle contient également comme méthode:

- un constructeur
- getDataCase() qui renvoie toutes les données d'une case (un objet DataCase) lorsqu'on lui donne le JPanel qui correspond.
- mouseClicked() qui change la couleur de la case lorsqu'on clic sur celle ci
- mouseEntered() qui, s'y on est en train de déplacer thésée ou la sortie, place thésée ou la sortie dans la case ou l'on vient d'entrer et met l'ancienne case qui représentait thésée ou la sortie en blanc sans image.
- mouseExited() et mouseMoved() qui ne font rien
- mousePressed() permet de stocker la case sur laquelle on vient de cliquer dans debutMouvement si jamais on veut déplacer thésée ou la sortie
- MouseReleased() permet de remettre debutMouvement à null et de remettre isDragged à false
- mouseDragged() permet de mettre isDragged à true pour indiquer qu'on déplace une case comme thésée ou la sortie

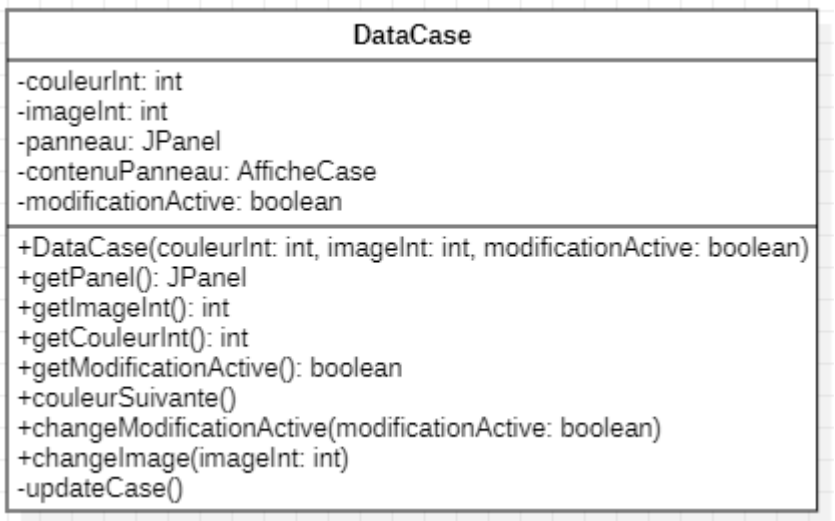


- DataCase

Cette classe contient toutes les données d'une case

elle contient les attributs:

- couleurInt qui contient la couleur de la case sous forme d'entier
- imageInt qui contient l'image de la case sous forme d'entier
- panneau qui contient le composant graphique de la case (JPanel)

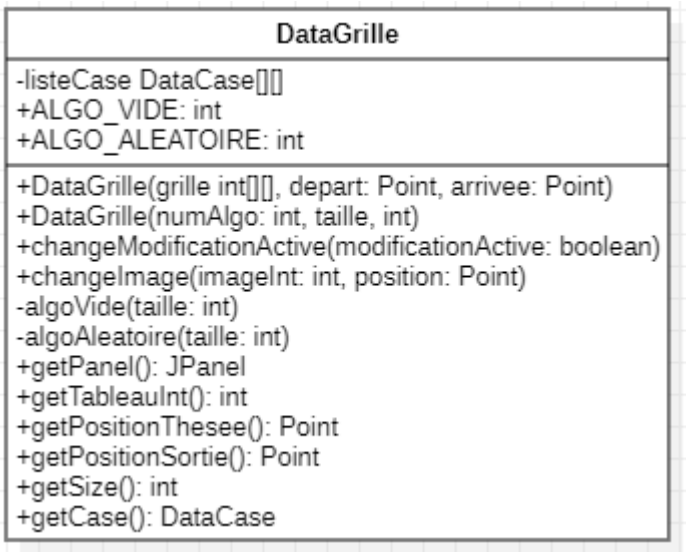


- contenuPanneau qui est la classe qui dessine la case
- modificationActive qui indique si la case est modifiable ou non

elle comme méthode:
- un constructeur
- getPanel(), getImageInt(), getCouleurInt() et getModificationActive() qui transmettent les données de la case.
- couleurSuivante() permet de changer la couleur de la case si cela est possible
- changeModificationActive(), permet de changer la valeur de l'attribut modificationActive et donc de déterminer si la case est en mode édition ou non
- changeImage() change l'image de la case
- update() permet de mettre la fenêtre à jour à chaque changement

- DataGrille

Cette classe contient toutes les données d'une grille. C'est cette classe qui fabrique la grille soit à partir de données soit à partir d'un algorithme de génération

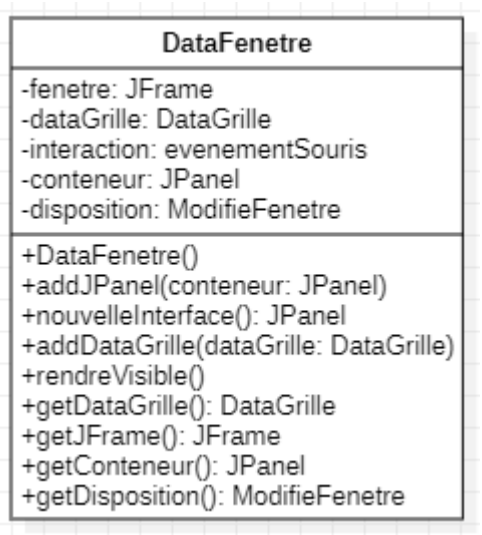


elle contient les attributs:
- listeCase, qui sont toutes les données de chaques case de la grille dans un tableau à deux dimension
- et les constante ALGO_VIDE et ALGO_ALEATOIRE qui permette de choisir qu'elle algorithme utiliser

elle contient comme méthode:
- 2 constructeurs: un qui permet de fabriquer la grille à partir d'un algorithme, et un autre qui la fabrique à partir de la position de thésée, de la sortie et d'un tableau remplis de 0 et de 1
- changeModificationActive() qui permet de changer l'état d'édition de la grille entière
- changeImage() permet de changer l'image d'une case à un point précis dans la grille
- algoVide et algoAleatoire sont appelés par le constructeur et permettent de créer la grille avec l'algorithme correspondant
- getPanel() permet de récupérer dans un tableau à deux dimension tout les JPanel de la grille
- getTableauInt() renvoie un tableau à d'entier deux dimensions qui représente la grille remplis de 0 et de 1. Avec cependant juste un 3 et un 4 pour la position de thésée et de l'arrivée.
- getPositionThesee() renvoie le point avec les coordonnée de thésée
- getPositionSortie() renvoie le point avec les coordonnée de la sortie
- getSize() revoie la taille de la grille
- getCase() renvoie un tableau à deux dimensions remplis de DataCase. (les données de chaques cases.

- DataFenetre

Cette classe contient toutes les donnée de la JFrame



elle contient les attributs:

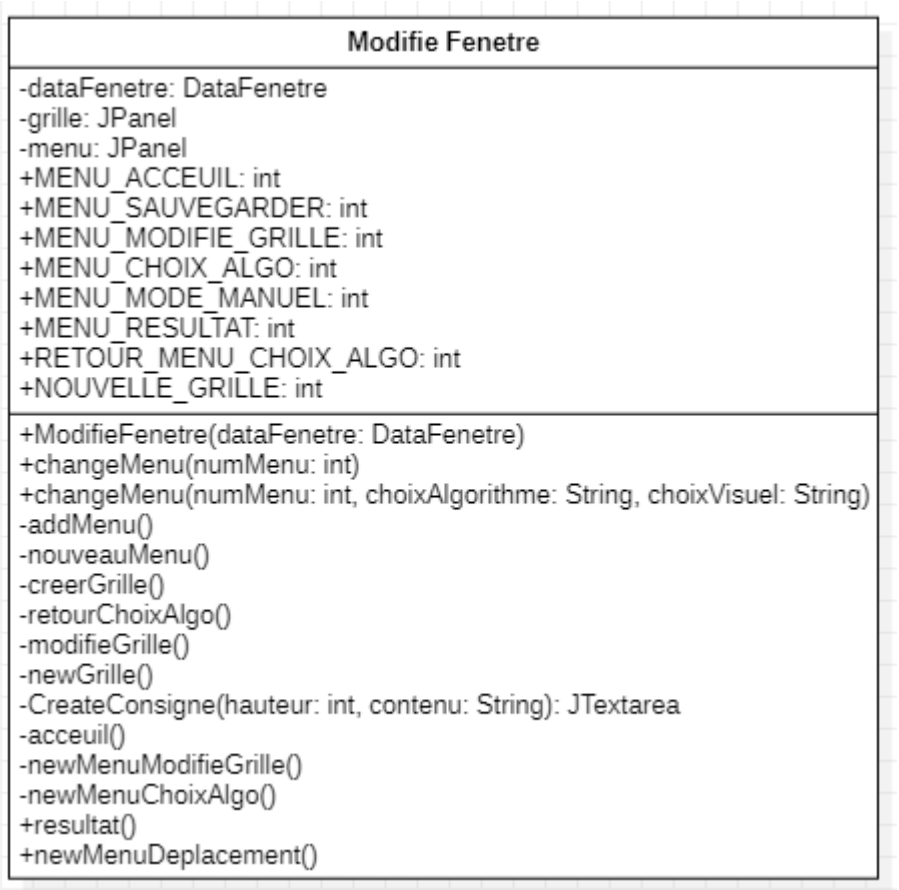
- fenetre qui est la JFrame sur laquelle se déroule le projet
- dataGrille qui est les information de la grille lorsque la fenetre en contient
- interaction qui est le contrôleur lorsqu'on veut personnalisé la grille
- conteneur qui est le panneau qui contient la fenetre
- disposition qui est la classe qui permet le passage d'une page à une autre sans changer de JFrame

elle contient également plusieurs méthodes:

- un constructeur
- addJPanel() qui permet de changer le contenus de la fenêtre et de l'actualiser
- nouvelleInterface() qui vide le panneau principale de la fenêtre et le renvoie
- addDataGrille() qui permet l'ajout d'une grille à la fenêtre
- rendreVisible() qui rend la fenêtre visible
- getDataGrille() qui renvoie les données de la grille présente dans la fenêtre
- getJFrame() qui renvoie la fenêtre
- getConteneur qui renvoie le panneau qui contient tous les éléments de la fenêtre.
- getDisposition() qui renvoie la classe capable de modifier la fenêtre

- ModifieFenetre

Cette classe modifie le contenu de la fenêtre.



elle contient les attributs:

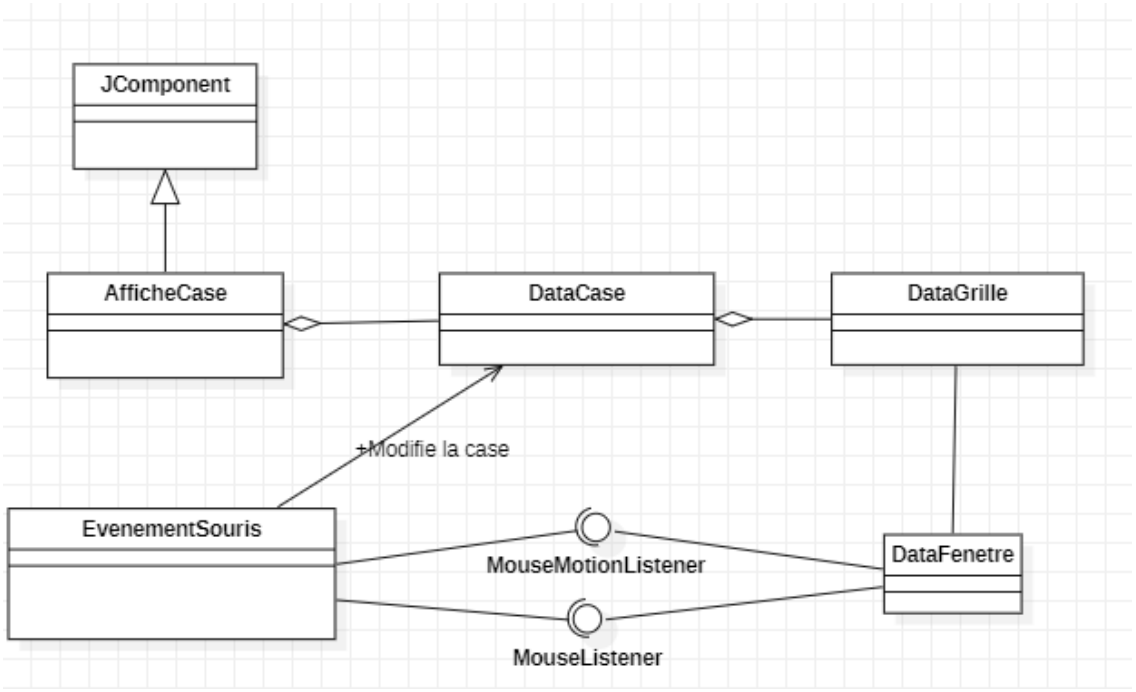
- dataFenetre: qui contient des données sur la fenetre
- grille: qui contient les données de la grille afin que celle ci soit stocker même lorsque la grille n'est pas affiché
- menu: qui contient composants du menu actuel
- De nombreuses constantes qui permette de connaître le menu que l'on veut faire apparaître sur la fenêtre

elle contient comme méthode:

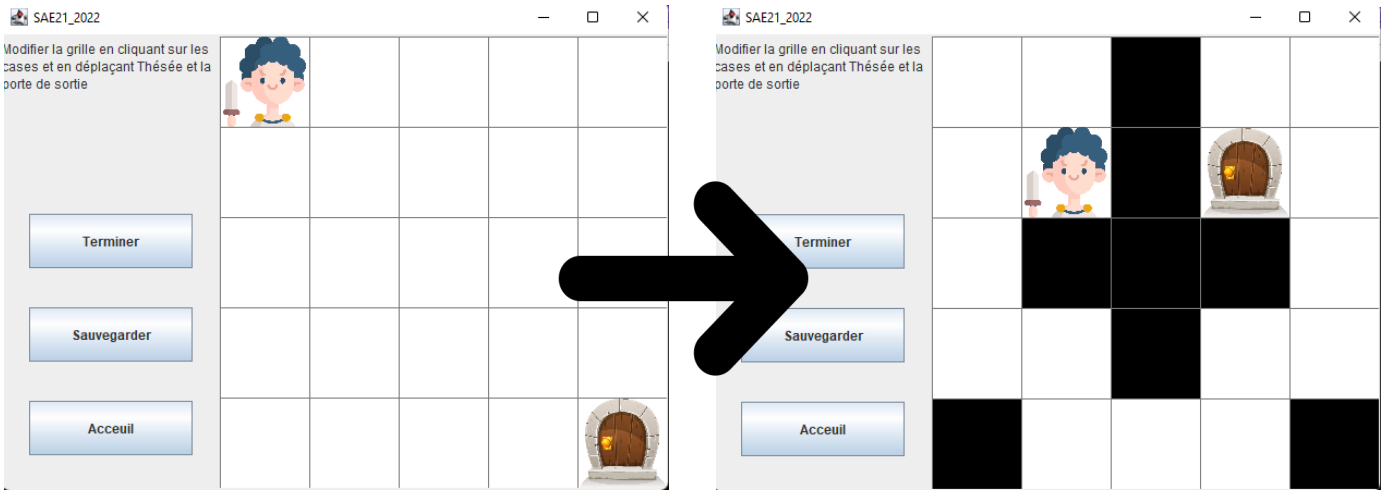
- un constructeur
- 2 méthodes qui permettent de changer l'apparence de la fenêtre en fonction du menu que l'on veut afficher.
- addMenu() permet d'ajouter le menu à la fenêtre
- nouveauMenu() permet de retirer l'ancien menu de la fenêtre et de renvoyer un nouveau menu vierge
- ModifieGrille(), retourChoixAlgo(), creerGrille(), modifieGrille(), newMenuModifieGrille(), newMenuChoixAlgo(), resultat() et newMenuDeplacement() permettent d'afficher sur la fenêtre le menu correspondant.
- newGrille() permet de créer graphiquement une grille et de la stocker dans l'attribut grille. cette méthode est appelé par modifieGrille()
- createConsigne() permet de fabriquer un message de type JTextArea.

2. structure du programme

- Personnalisation de la grille

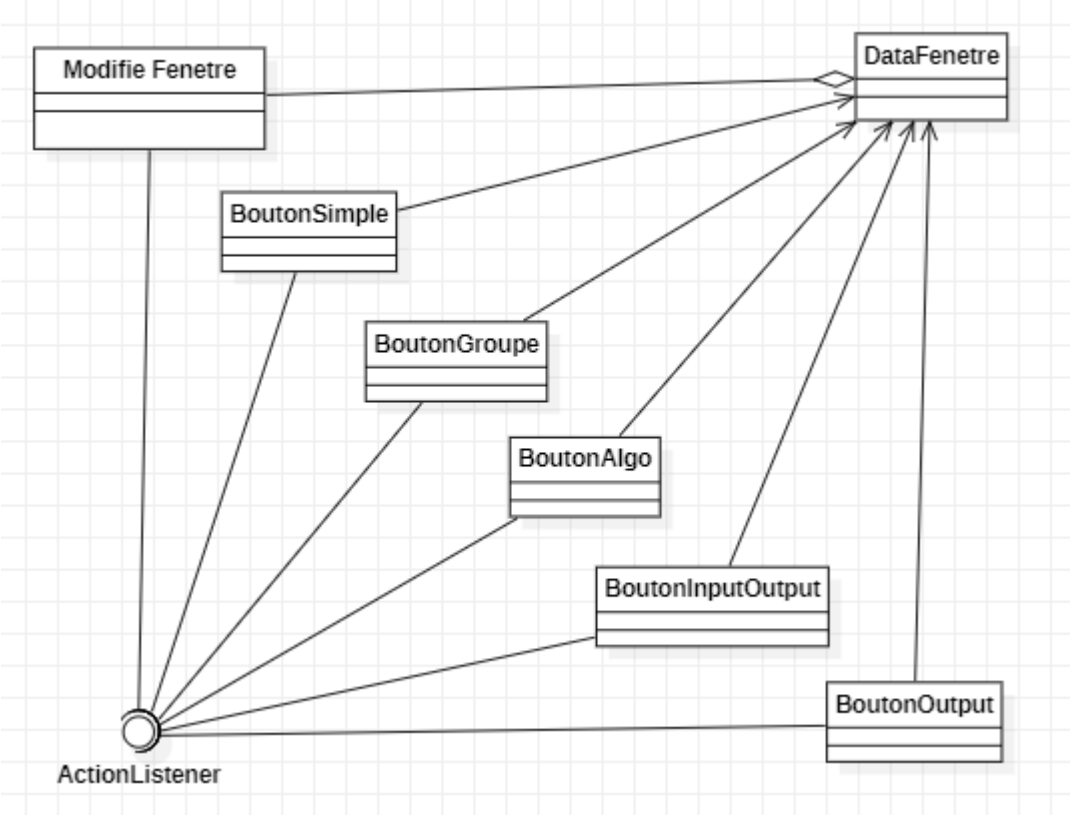


Ce diagramme simplifié d'une partie du projet montre comment l'utilisateur peut personnaliser la grille
Nous avons une fenêtre (DataFenetre). Dans celle-ci nous avons une grille (DataGrille), La grille est composée de cases (DataCase) et chaque case est dessinée dans la fenêtre (AfficheCase)
Sur la fenêtre un écouteur d'évènement à été ajouté (EvenementSouris). Celui-ci utilise 2 interfaces MouseMotionListener et MouseListener. Ainsi, lorsque l'utilisateur interagit avec une case, le contrôleur (EvenementSouris) va modifier les données de DataCase, ce qui va



se répercuter sur AfficheCase et cela va de ce fait changer l'apparence de la case visible sur la fenêtre.

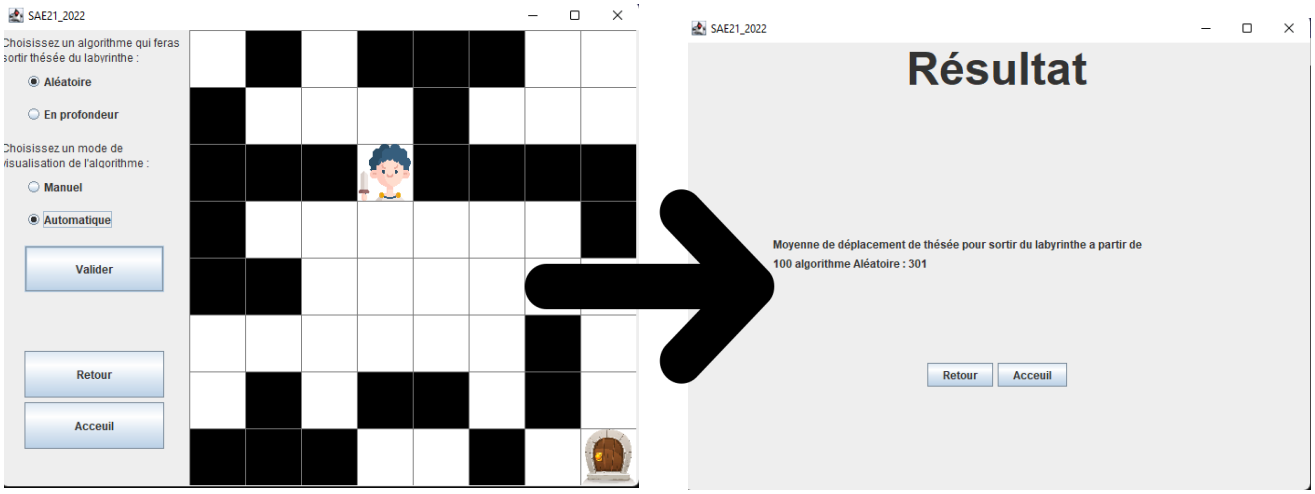
- Le changement d'interface



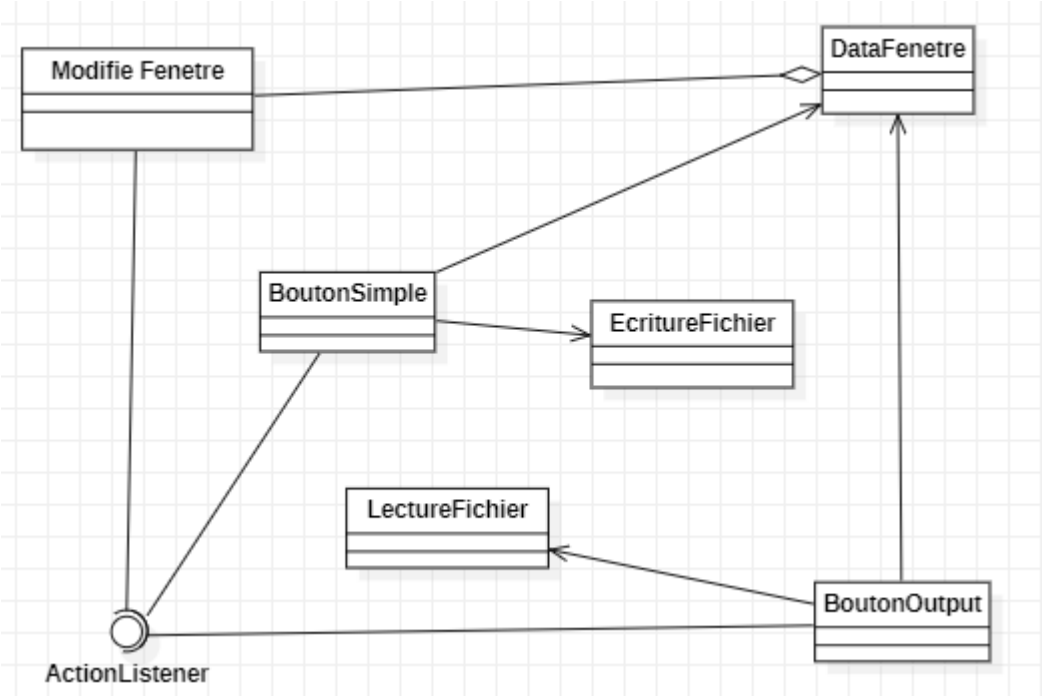
DataFenetre contient un attribut de type ModifieFenetre, qui interagit directement avec les composants de la fenêtre.

Souvent, ModifieFenetre crée un JPanel avec des JButton à l'intérieur. Il remplace l'ancien JPanel par celui qu'il vient de créer ce qui fait une nouvelle interface.

Sur chacun des boutons, un contrôleur est ajouté (ex: BoutonSimle). Celui-ci permet d'interagir avec la fenêtre et de la modifier en appelant l'objet de la classe ModifieFenetre lié à cette fenêtre.



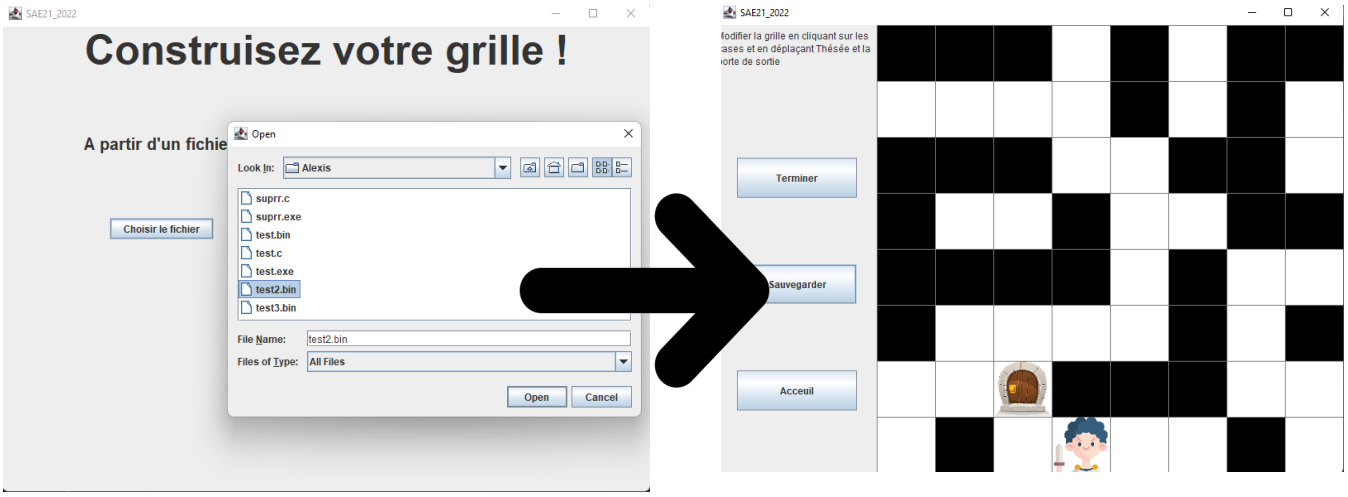
- La sauvegarde



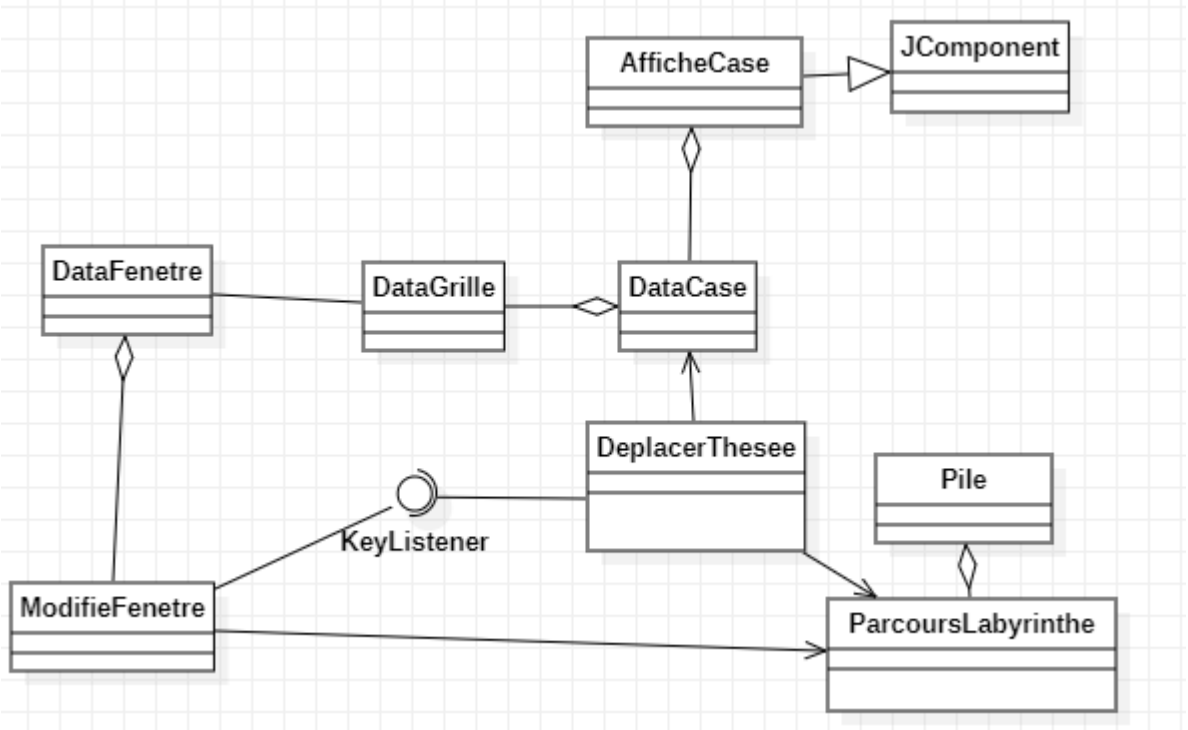
Le système de sauvegarde est basé sur le même principe que précédemment. ModifieFenetre permet de mettre sur la fenêtre une interface avec des boutons.

Certains de ces boutons permettent de sauvegarder ou charger une grille. Pour ce faire, on utilise une interface ActionListener pour détecter lorsque l'utilisateur clic sur le bouton.

Ensuite, les contrôleurs correspondant utilisent les méthodes Lecture fichier ou EcritureFichier pour sauvegarder ou charger une grille. Ils peuvent éventuellement changer l'apparence de la fenêtre lorsque c'est nécessaire (par exemple aux chargement d'une grille celle ci apparaît sur la fenêtre)



- La visualisation de l’algorithme



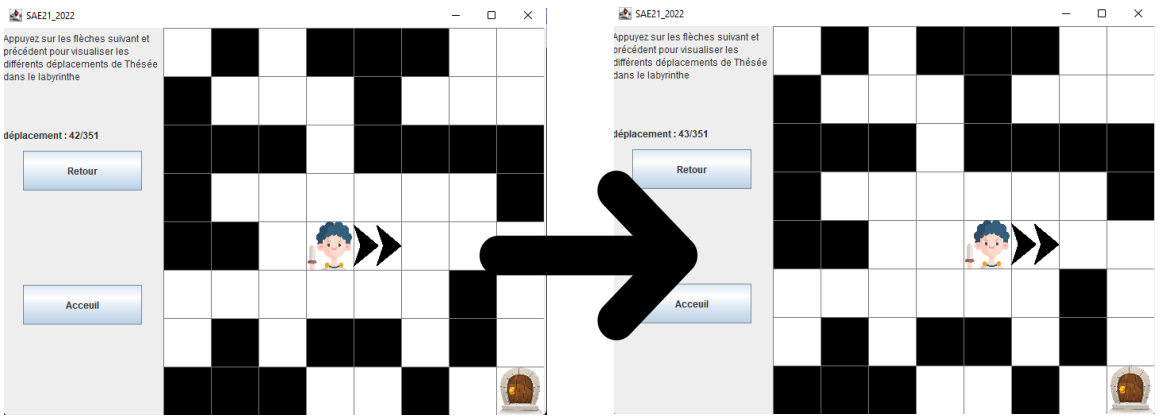
Lorsqu’on est en mode manuel, on peut visualiser l’algorithme.

ModifieFenetre permet de créer l’interface graphique qui correspond à cela. Dans certaines circonstances, on calcule les positions successives de thésée avant l’affichage de cette interface. D'où la flèche qui relie ModifieFenetre et ParcoursLabyrinthe.

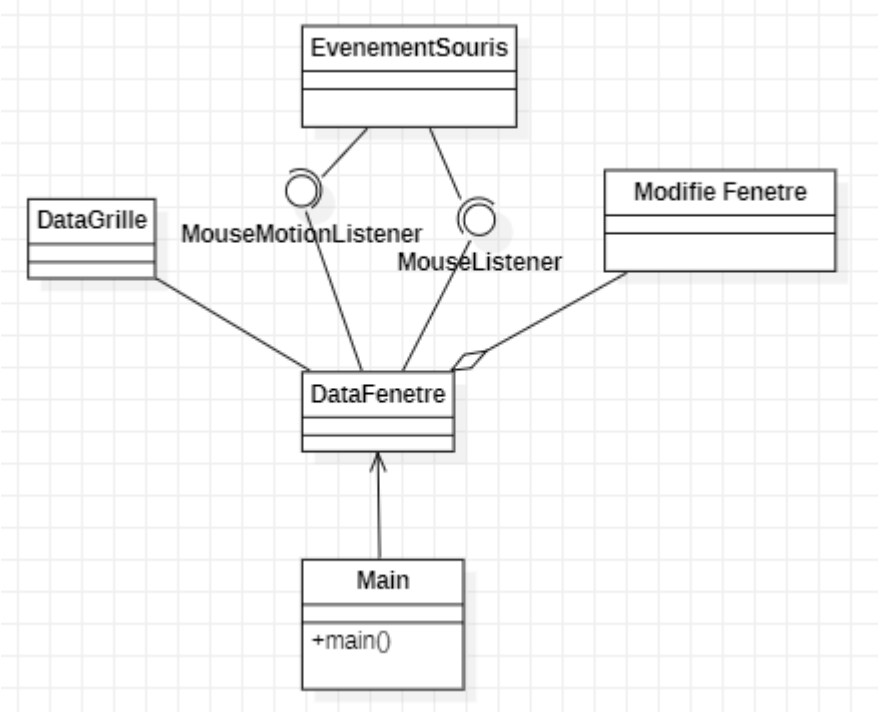
Sur l'élément qui à le focus dans cette interface, on place un écouteur d'événement qui va détecter la pression sur les touches du clavier (KeyListener). Cet écouteur d'événement c’est DeplacerThesee.

Il va pouvoir lire ou calculer la prochaine ou la dernière position de thésée avec la classe ParcoursLabyrinthe. Celle-ci utilise une pile pour stocker toutes ces positions.

Ensuite, comme vous avez pu le voire dans un diagramme plus haut, pour changer l’apparence des cases, une fois que le contrôleur (DeplacerThesee) à obtenus les positions à afficher, il vas modifier les cases correspondantes (DataCase), cela va ensuite se répercuter sur les dessin affichées sur la fenêtre (AfficheCase)



- Le Main



Lorsqu'on exécute le programme, le Main interagit avec DataFenetre. et c'est ensuite tous les écouteurs d'événements et autres objets liés à DataFenetre qui vont faire évoluer la suite du programme.

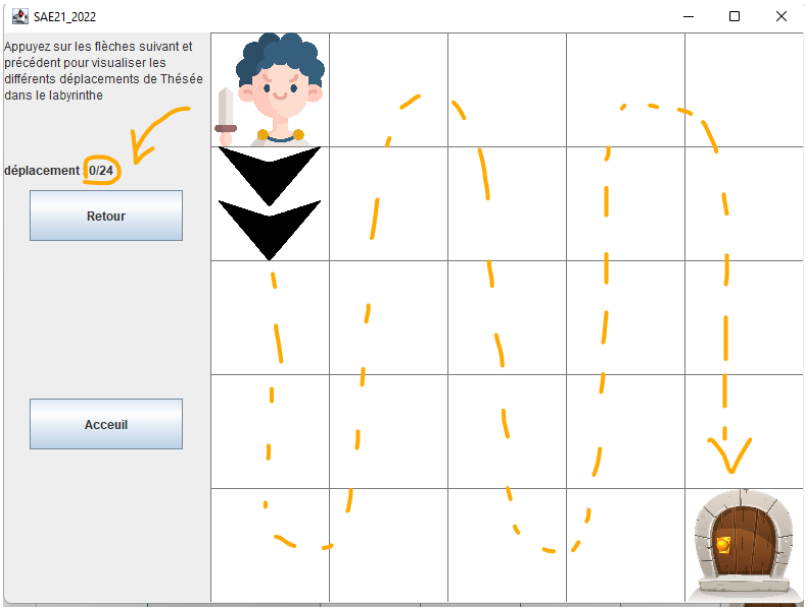
- Le MakeFile

Nous avons aussi créé un Makefile qui permet de compiler nos fichiers , les exécuter et supprimer les fichiers .class en 3 commandes.
Le **makefile** se sert du main pour tout compiler avec la commande **make**. Les fichiers .class ainsi créés sont stockés dans le dossier source **"src"** . Une fois compilé il suffit de taper la commande **make run** pour exécuter le programme ce qui nous ouvre notre interface et donc notre jeu. Si on souhaite supprimer les fichiers, il suffit de taper la commande **make clean** ce qui supprime tout les fichiers .class de notre dossier source.

algorithme déterministe

1. Fonctionnement

Pour sortir du labyrinthe efficacement, thésée utilise un parcours en profondeur.



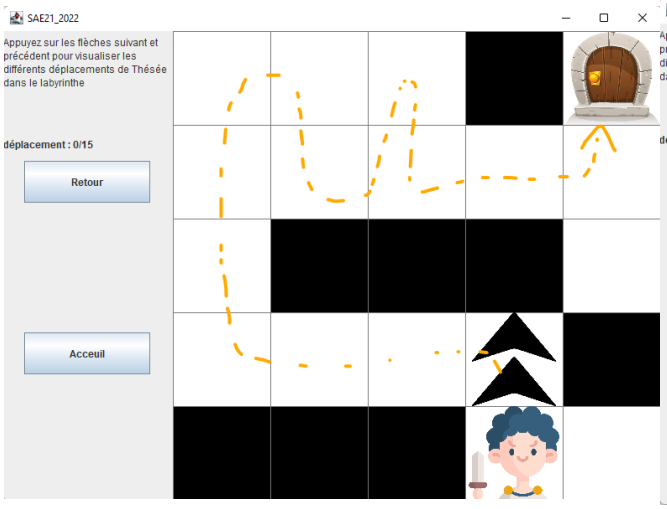
Ce n'est pas l'algorithme le plus efficace comme on peut le voire ci contre.
Ici thésée passe par toutes les cases alors qu'elle aurait pu être 3 fois plus efficace.

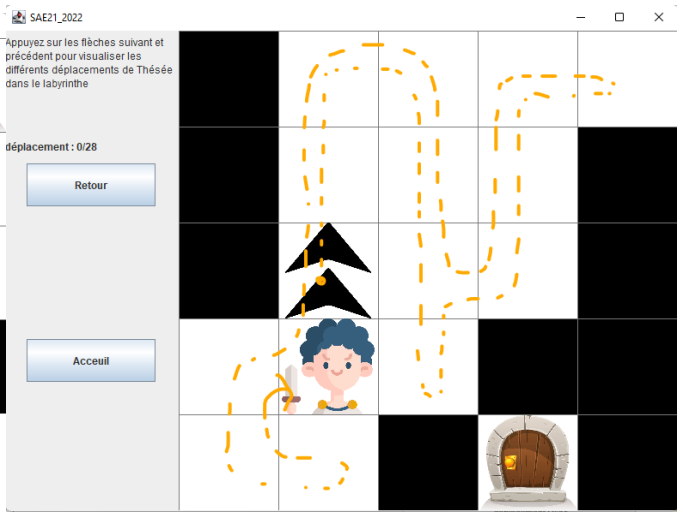
Mais comme il est dit dans la consigne que thésée ne voit que ce qu'elle à autours d'elle, ça reste l'algorithme le plus efficace qui répond à ces conditions.

Dans notre algorithme en profondeur thésée essaie toujours de passer sur des cases qu'elle n'a encore jamais exploré.
A chaque fois qu'elle arrive sur une case, thésée, essaie en premier lieu d'aller au nord. Si elle ne peut pas accéder à cette case ou qu'elle l'a déjà visité, elle essaie la case de l'ouest, puis celle du sud et enfin celle de l'est.
Si Thésée n'a pas réussi à atteindre de nouvelle case, elle revient sur ses pas et recommence le même processus pour découvrir de nouveau chemin à partir de sa nouvelle position.

Cet algorithme est relativement efficace car thésée ne passe que grand maximum 4 fois sur une même case.

2 issus sont possibles. soit thésée à finis par découvrir la case de sortie, soit elle à essayer tous les chemins et est revenue à son point de départ à attendre de mourir de faim.





Voici deux autres exemples illustratifs:

2. Preuve qu'il trouve toujours une sortie si il y en a accessible:

Démonstration par l'absurde.
Le parcours en profondeur visite successivement toutes les cases voisines
Supposons que la porte de sortie ne soit pas accessible par l'algorithme en profondeur qui part de thésée. Cela signifie que soit la porte est isolée, soit qu'elle est connectée à une autre zone du labyrinthe non accessible par thésée.

Cela montre que les seul cas où thésée ne peut pas accéder à la porte c'est que celle ci n'est pas connecté au labyrinthe dans lequel se trouve thésée

Conclusion

1. Hassan Meite

En conclusion, le projet que nous avons partagé a été un défi, car le langage de programmation Java est relativement nouveau pour nous et nous avons dû faire preuve de réflexion et de recherche. Cependant, cela nous a donné l'opportunité d'approfondir nos connaissances et nos compétences en Java, ainsi que d'améliorer notre gestion du temps, de la tâche et notre capacité à travailler efficacement en équipe pour répondre aux attentes de la SAé. Alexis a été un bon équipier toujours à l'écoute et prêt à expérimenter beaucoup d'idées.Nous avons pu avancer correctement ensemble pour le travail en tant et en heure.

2. Alexis Wamster

Effectuer un projet en équipe n’est pas si évident qu’il n’y paraît.
Au début, on commence par faire un diagramme de classe et à diviser les tâches, mais on se rend compte rapidement que le diagrammes est faux et qu’on ne peut pas réellement travailler chacun dans son coin. En effet, on ne travaille pas à la même vitesse alors on s’entraide et les classes sur lesquelles on travaille sont tout de même liées.
Il ne faut donc pas oublier la cohésion de l’équipe.