Nell TELECHEA-LOJOU
Benjamin BRIBANT
Maxence RAYMOND

BUT2 Informatique Fontainebleau

RAPPORT POINT PURSUIT

SAE DEV 4.5

TABLE DES MATIERES

Contenu

I. Introduction

II. Description des fonctionnalités

III. Structure du programme

IV. Algorithme de détection de la résolution

V. Conclusion

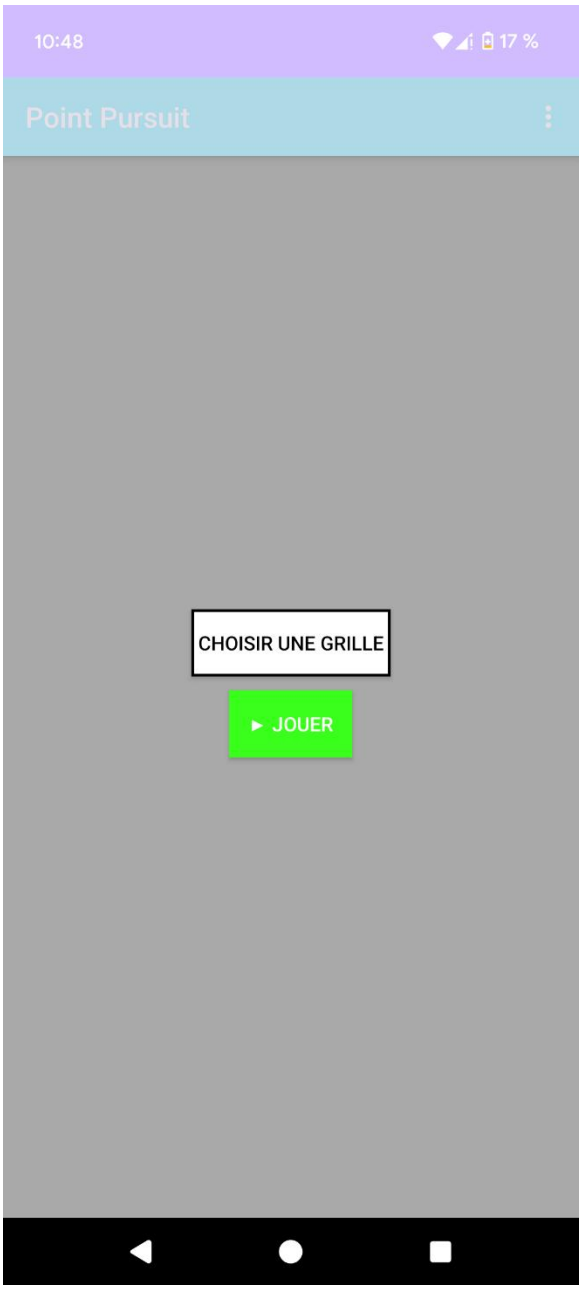
INTRODUCTION

I. Introduction

Dot Link (ici Point Pursuit) est un jeu de puzzle qui consiste à relier des points de même couleur qui vont par paire et sont disposés dans une grille carrée. Les liaisons des points forment un certain chemin qui permet de compléter le puzzle. Le jeu se joue sur API 19 minimum, il y a un maximum de 30 paires.

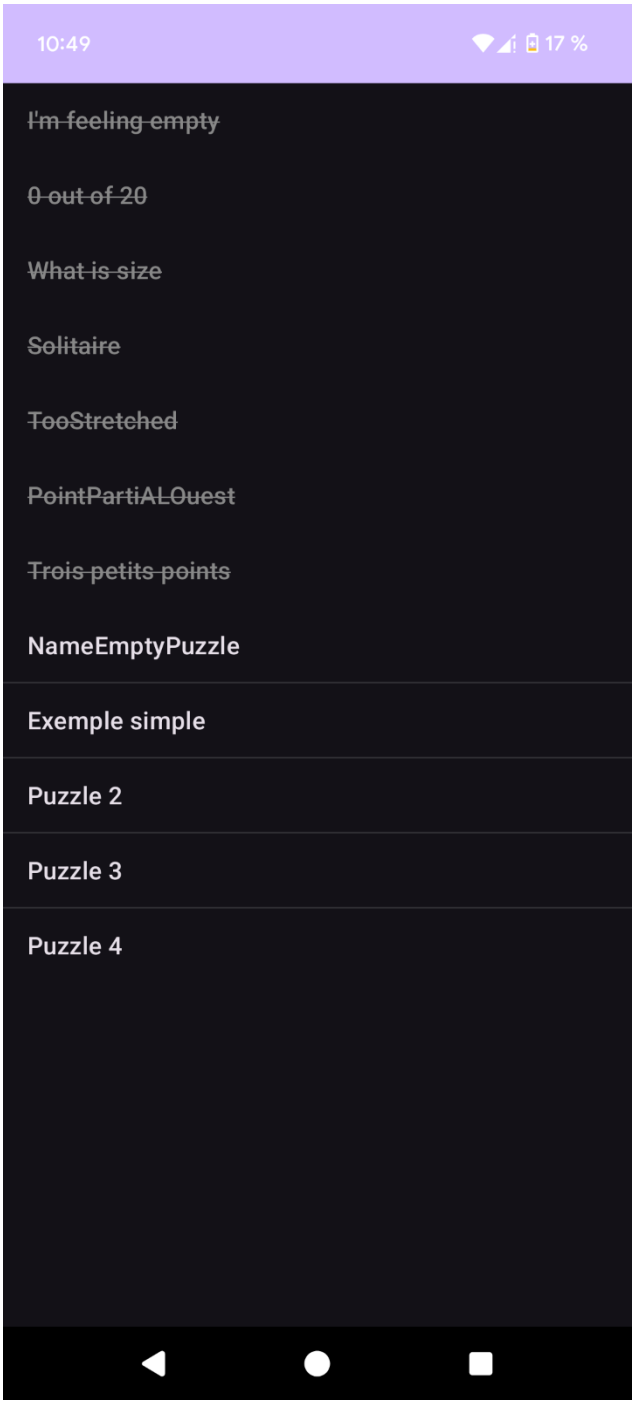
II. Description des fonctionnalités

Lorsqu'on lance le jeu, un menu s'ouvre avec deux boutons, dont un menu d'options.



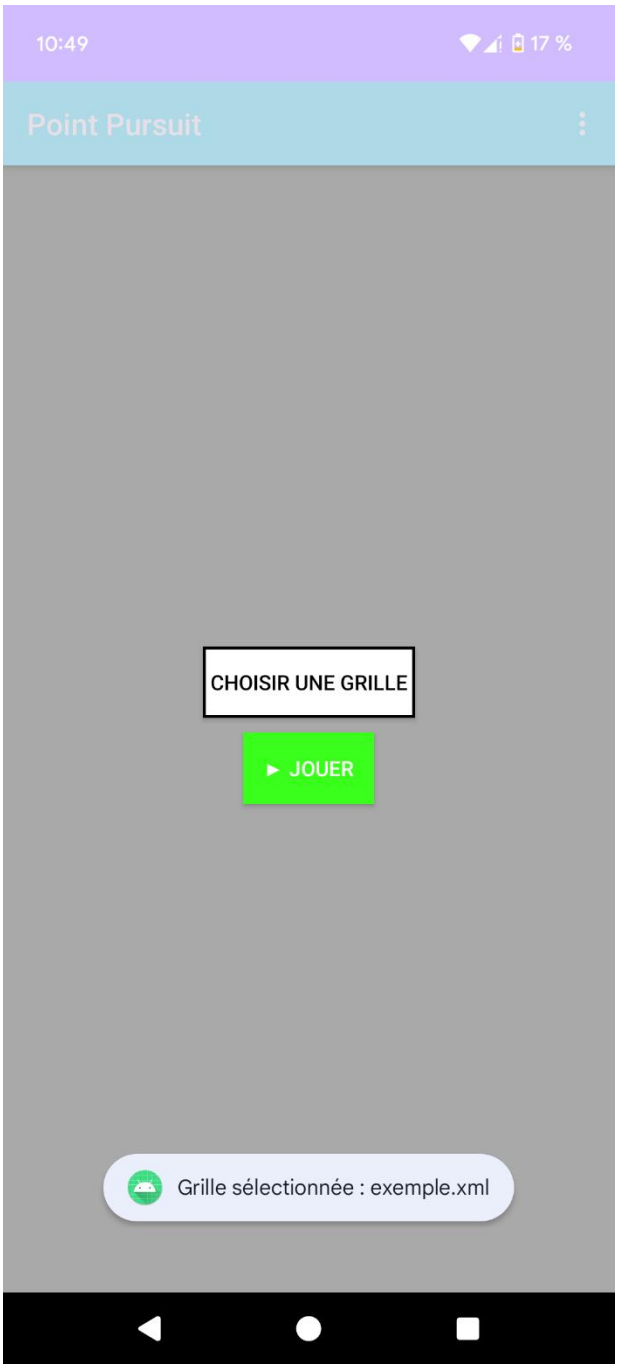
DESCRIPTION DES FONCTIONNALITES

Le bouton « choisir une grille » permet de choisir la grille avec laquelle on va jouer parmi une liste dans une autre fenêtre.



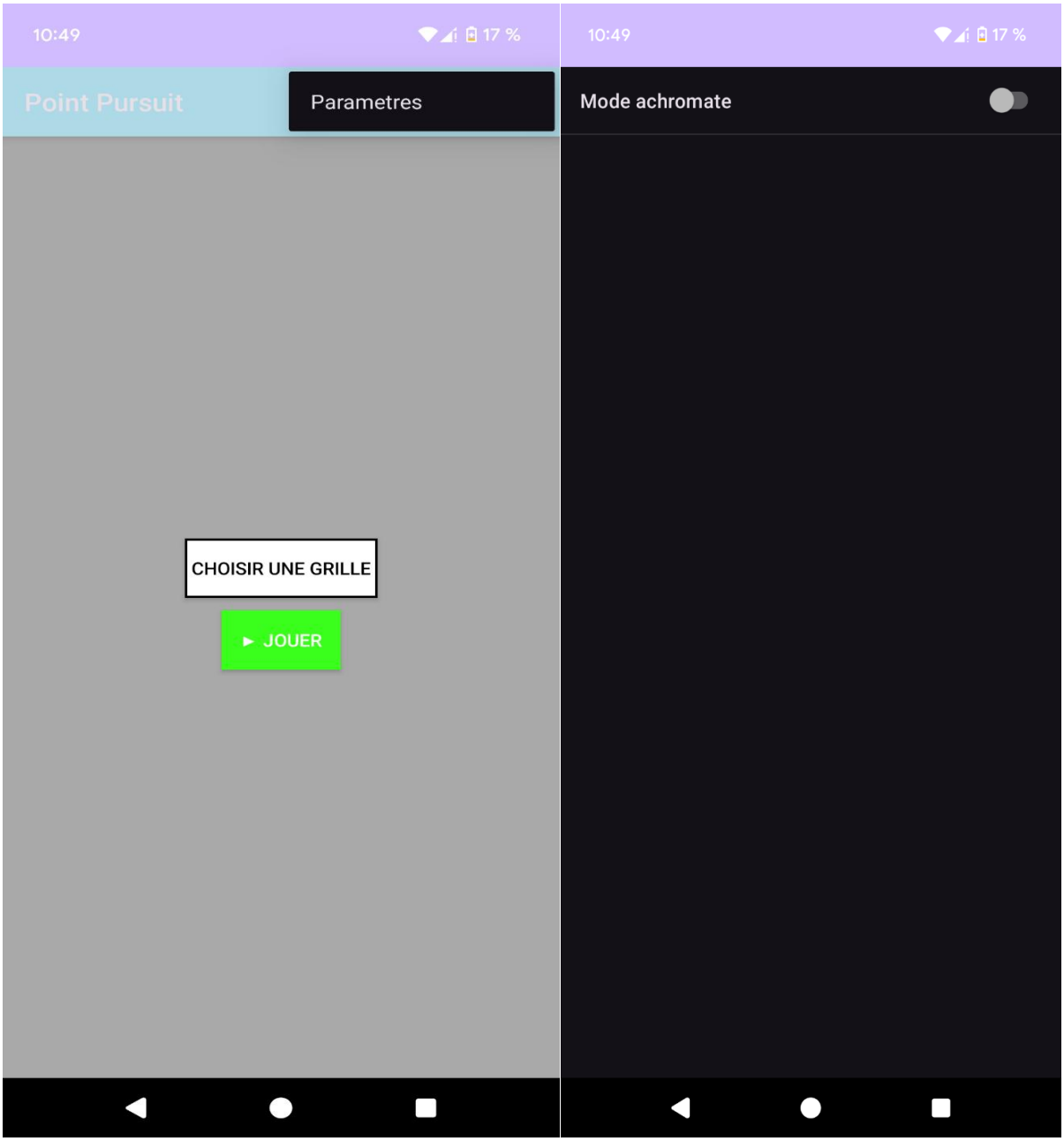
DESCRIPTION DES FONCTIONNALITES

Si un puzzle n’est pas valide il est impossible le sélectionner. Le joueur est renvoyé sur la page du menu avec le nom du puzzle sélectionné qui s’affiche sur un message.



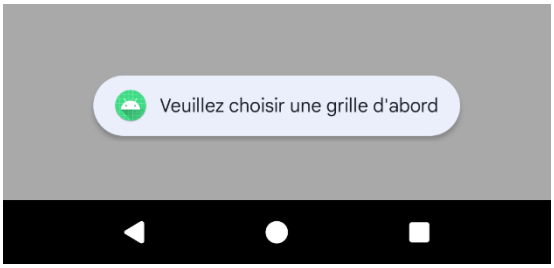
DESCRIPTION DES FONCTIONNALITES

Le menu d’options permet de modifier les options, ici d’activer ou non le mode achromate dans le jeu dans lequel les différentes couleurs sont remplacées par des tons de gris.



DESCRIPTION DES FONCTIONNALITES

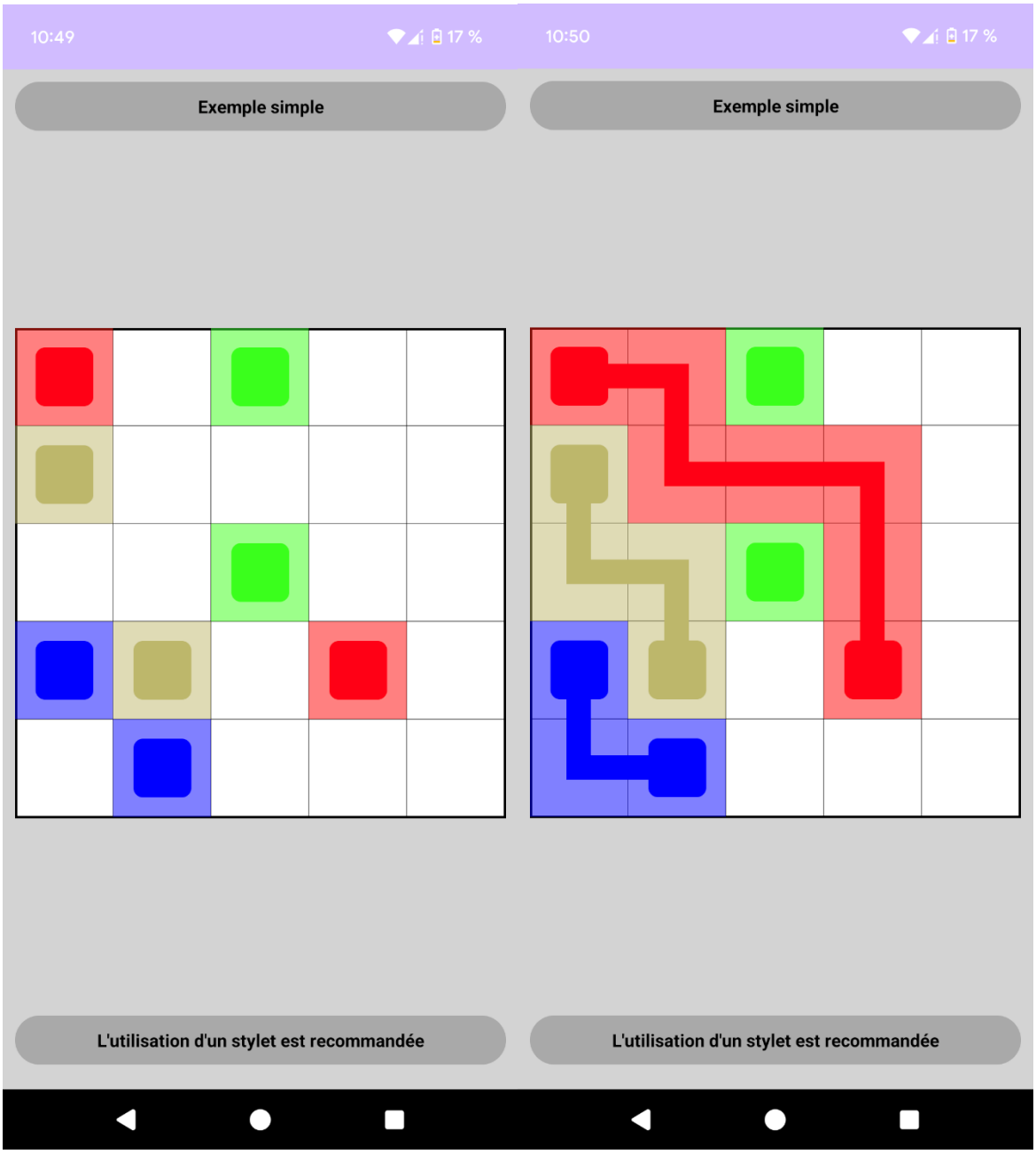
Le bouton « jouer » va lancer la partie, si aucune grille n’a été choisie, un message d’erreur s’affiche en indiquant au joueur d’en choisir une, sinon le jeu se lance.



Dans le jeu, on peut tracer une ligne pour relier deux points, si le tracé est en diagonale, n’atteint pas l’autre point, croise une ligne ou sort de la grille, le tracé est annulé.

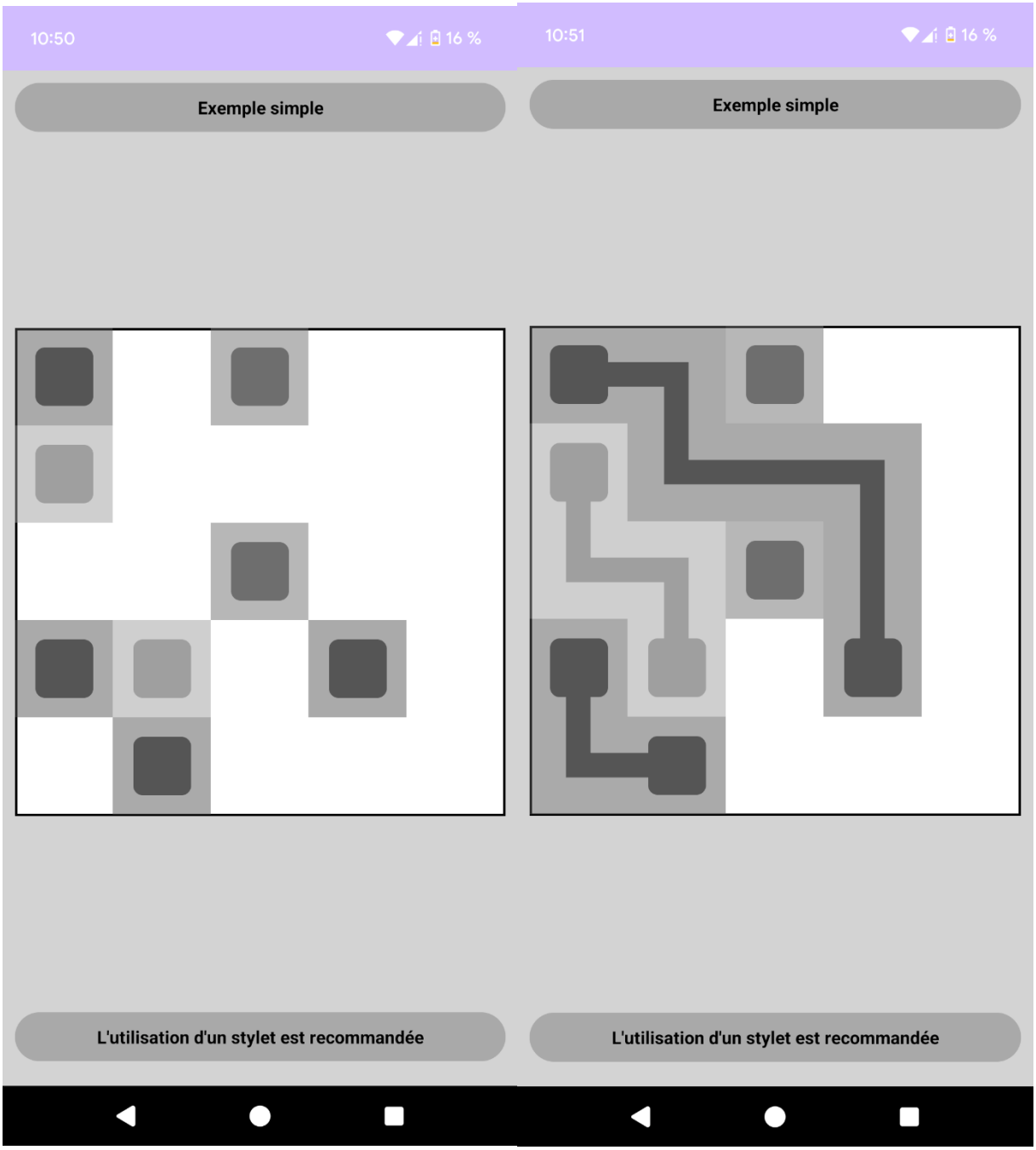
Lorsque le joueur touche une extrémité d’une ligne déjà tracée cela l’efface et on peut la retracer.

DESCRIPTION DES FONCTIONNALITES



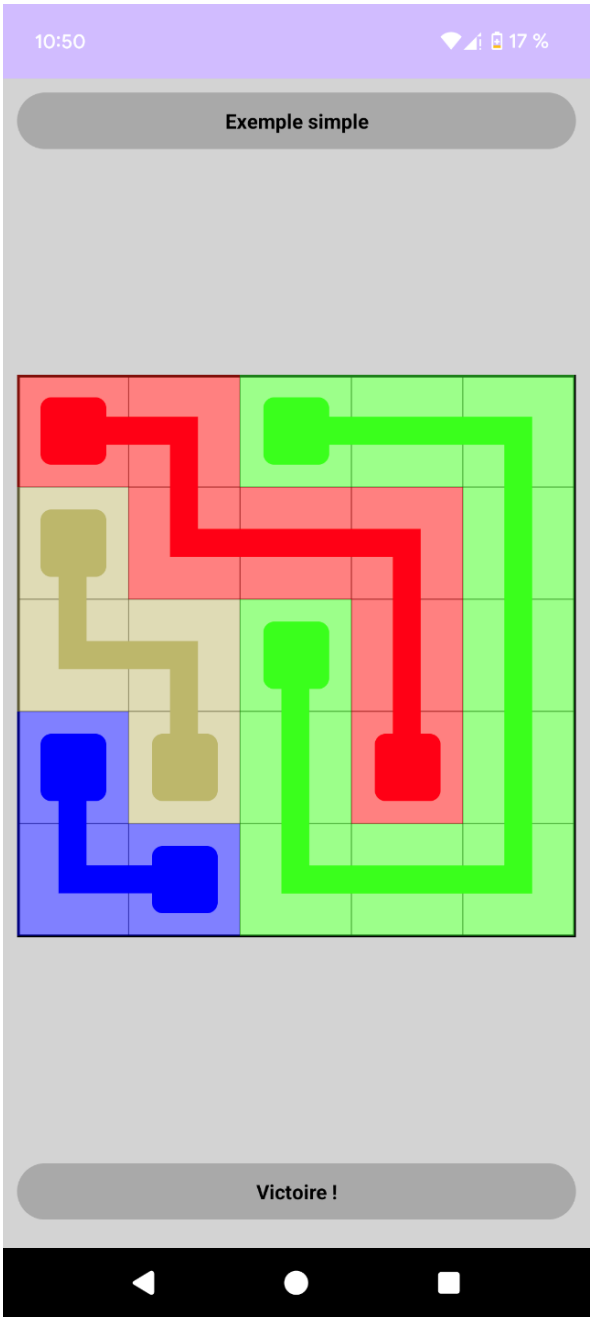
DESCRIPTION DES FONCTIONNALITES

En mode achromate :



DESCRIPTION DES FONCTIONNALITES

Lorsque le jeu est fini, un message de victoire s’affiche et la grille ne permet plus aucune modification.



III. Structure du programme

Activité du menu :

MenuActivity.java

MenuActivity
-selectedGrid: String
#onCreate(Bundle): void #onActivityResult(int requestCode, int resultCode, Intent intent): void #onSaveInstanceState(Bundle): void #onRestoreInstanceState(Bundle): void +getSelectedGrid(): String +setSelectedGrid(String selectedGrid): void +onCreateOptionsMenu(Menu menu): boolean +onOptionsItemSelected(MenuItem item): boolean

Cette classe gère l'activité du menu.

JouerClickListener.java

JouerClickListener
-activity: Activity
+JouerClickListener(MenuActivity) +onClick(View v): void

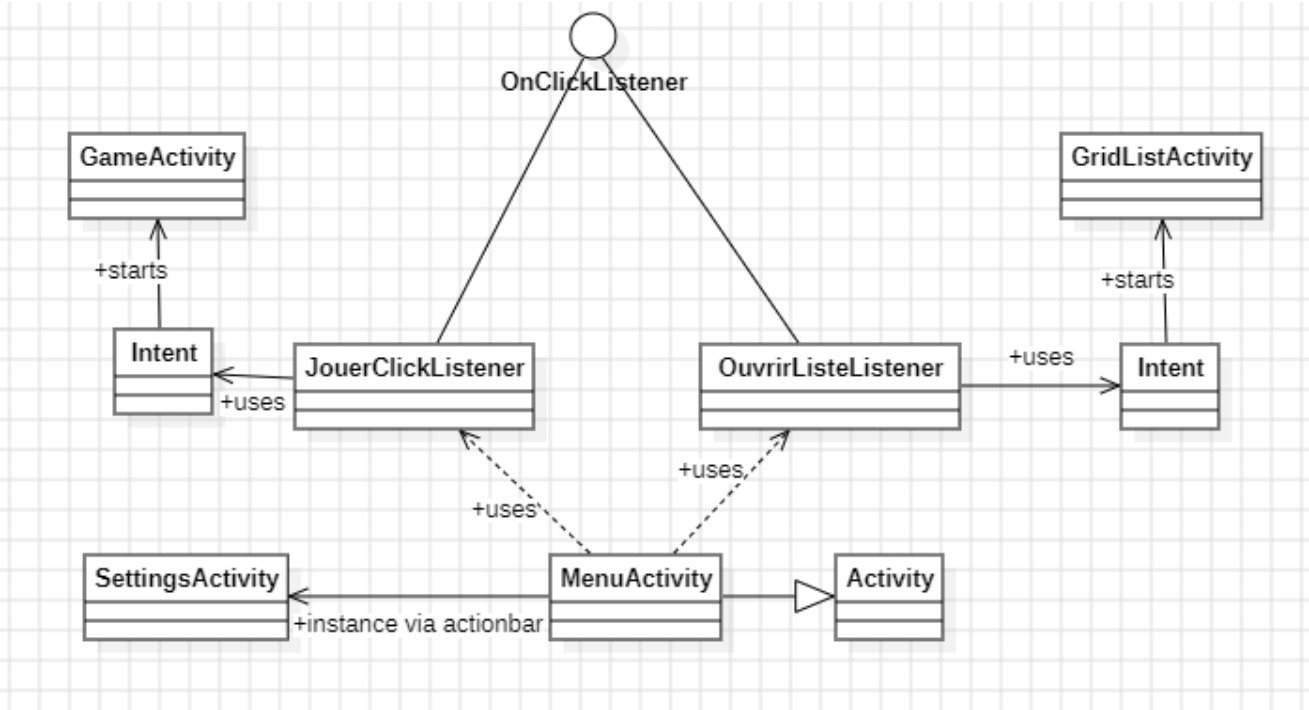
Gère les clicks du bouton jouer du menu.

OuvrirListeListener.java

OuvrirListeListener
-activity: Activity
+onClick(View v): void

Gère les clicks du bouton de choix des grilles.

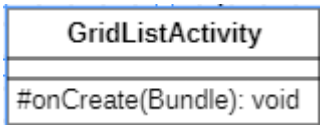
STRUCTURE DU PROGRAMME



Le menu est lancé, il contient 2 boutons, un pour jouer qui va lancer l'activité de jeu et un autre qui va permettre d'aller sur une activité pour choisir sa grille de jeu. On utilise un actionbar pour les options.

Activité de la liste de grille :

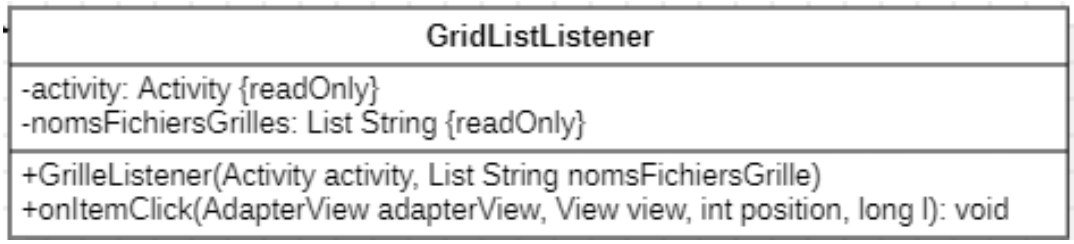
GridListActivity.java



Cette activité gère le choix de la grille de jeu.

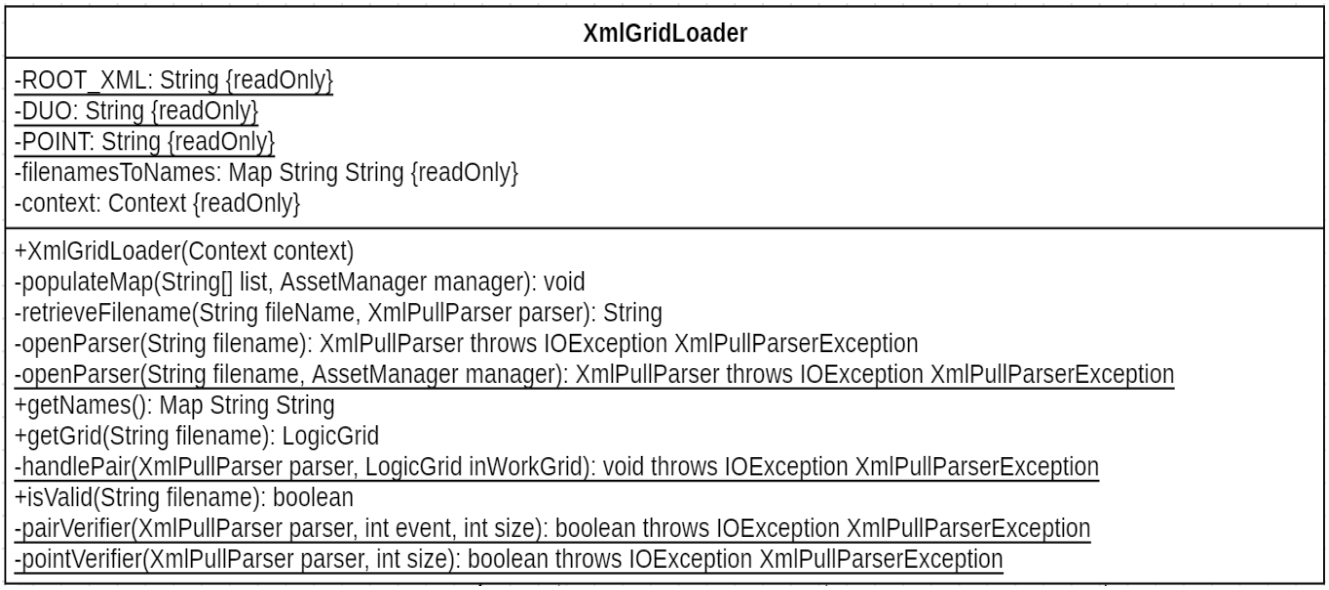
STRUCTURE DU PROGRAMME

GridListListener.java



Gère les clicks sur les éléments de la liste de grilles.

XmlGridLoader.java



Gère la lecture et le stockage des grilles et de leurs données à partir de leurs fichiers puzzle xml. Il utilise un XmlPullParser.

STRUCTURE DU PROGRAMME

LogicGrid.java

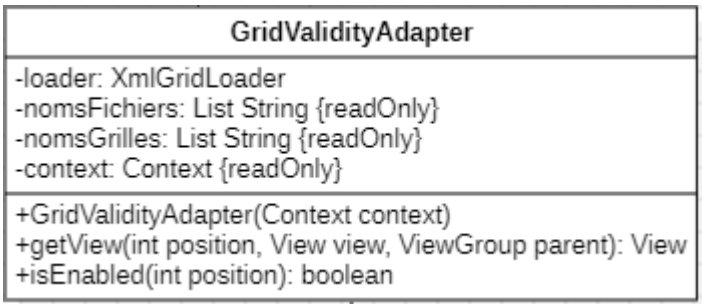
LogicGrid
-name: String {readOnly} -size: int {readOnly} -internalGrid: Case[**] -colorHelper: ColorHelper {readOnly} -inConstructionLine: Line -pointNumber: int +appliedLines: List Line {readOnly}
+LogicGrid(int size, String name, Context context) +setNewPair(int x1, int y1, int x2, inty2): void +getPointNumber(): int +getInternalGrid(): Case[**] +getSize(): int +equals(Object o): boolean <<override>> +isVictory(): boolean +startLine(int x, int y): boolean +addPointToLine(int x, int y): boolean +cancelLine(): void -applyLine(): void +getName(): String +removeLine(int x, int y): boolean +isCaseEmpty(int x, int y): boolean +getLines(): List Line +getCurrentLine(): Line -applyRestoredLine(Line l): void

Note : la notation [**] représente un tableau à double entrée

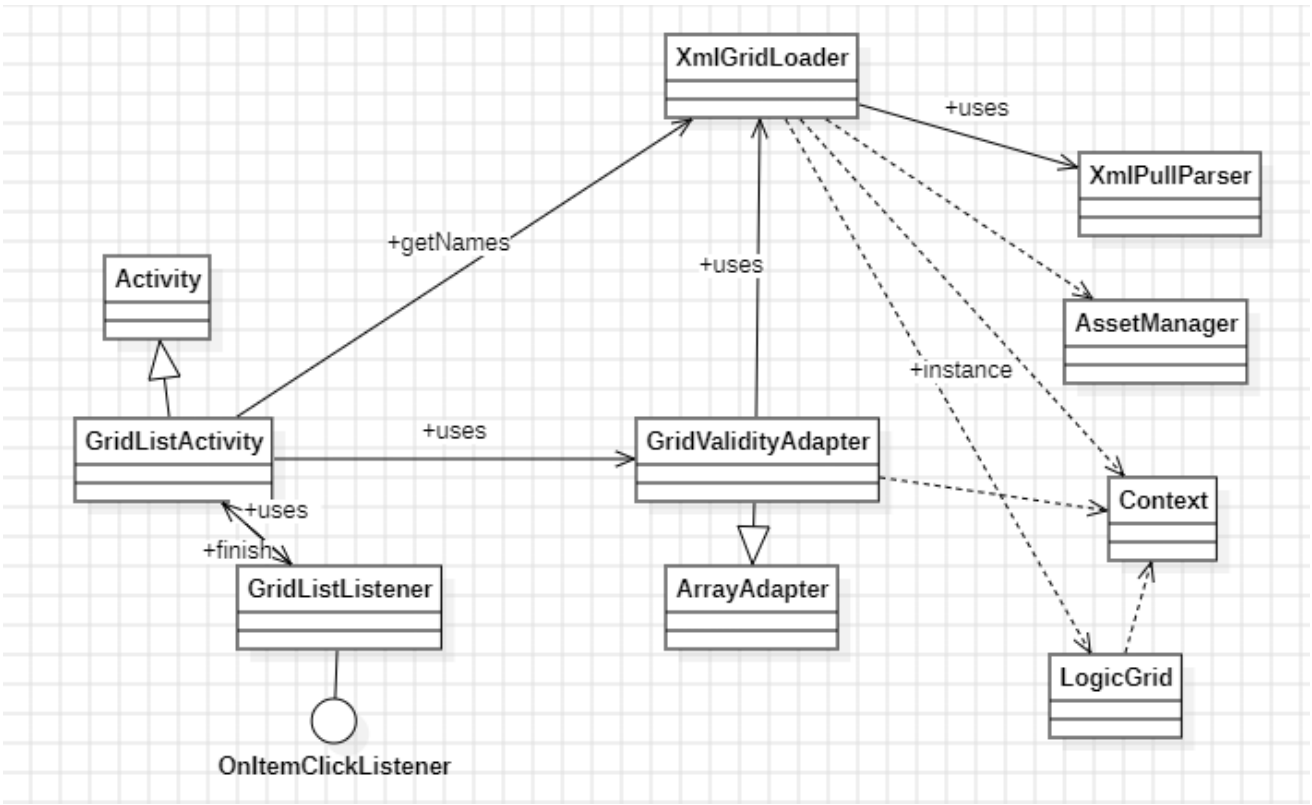
Représente la logique du jeu.

STRUCTURE DU PROGRAMME

GridValidityAdapter.java



Permet de vérifier si une grille de la liste est valide pour le jeu.



La liste récupère les noms des grilles grâce à XmlGridLoader, GridValidityAdapter vérifie la validité de chaque grille (rend la grille insélectionnable si non valide), puis on récupère la sélection du joueur via GridListListener et le tout est instancié dans LogicGrid.

STRUCTURE DU PROGRAMME

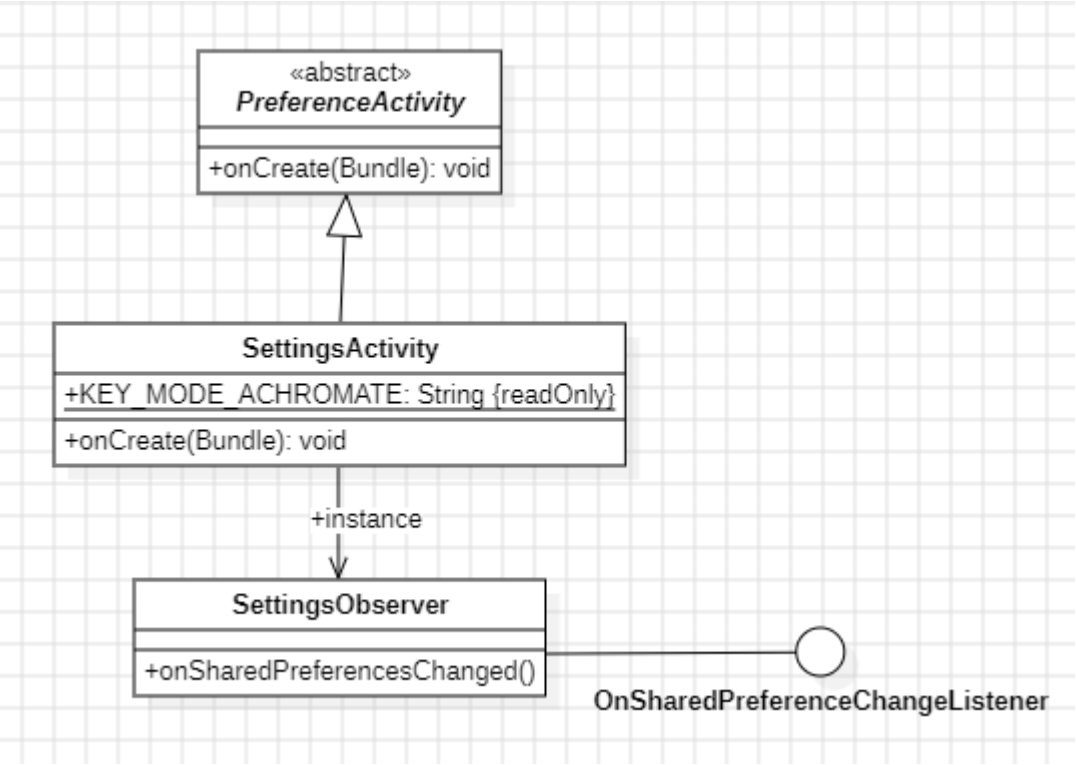
Activité des options :

SettingsActivity.java

Cette activité gère l'option achromate du jeu en héritant de PreferenceActivity.

SettingsObserver.java

Surveille les changements de l'option.



Comme dit dans SettingsActivity, cela gère l'activation du mode achromate.

STRUCTURE DU PROGRAMME

Activité du jeu :

GameActivity.java

GameActivity
-grid: LogicGrid -view: ViewGrid -filename: String -grayscale: boolean
#onCreate(Bundle): void #onSaveInstanceState(Bundle): void #onRestoreInstanceState(Bundle): void

Cette classe gère l'activité du jeu.

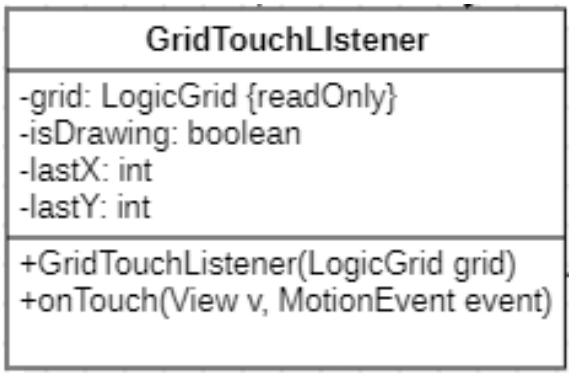
ViewGrid.java

ViewGrid
-grid: LogicGrid -grayscale: boolean -gridSize: int -gtl: GridTouchListener -sc: SegmentCalculator {readOnly} -defaultPaint: Paint {readOnly} -changingPaint: Paint {readOnly}
+ViewGrid(Context context, AttributeSet attrs) +setGrid(LogicGrid grid): void #onSizeChanged(int w, int h, int oldw, int oldh): void <<override>> +setGrayScale(boolean value): boolean #onMeasure(int widthMeasureSpec, int heightMeasureSpec): void <<override>> #onDraw(Canvas canvas): void <<override>> +drawLines(Canvas canvas): void +drawGrid(Canvas canvas): void +victory(): void

Cette classe représente la vue du jeu.

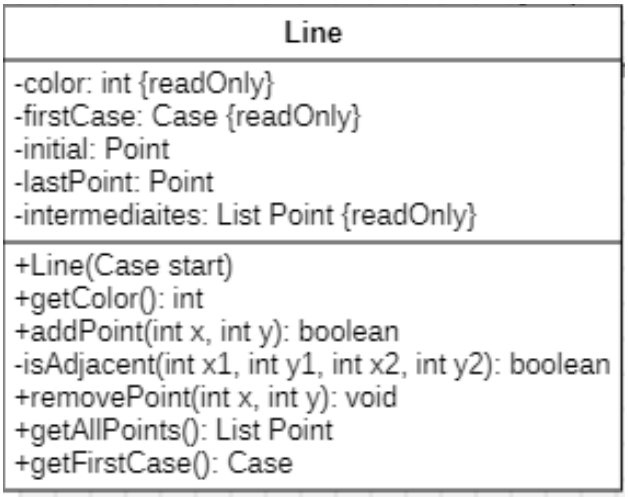
STRUCTURE DU PROGRAMME

GridTouchListener.java



Permet de gérer les interactions tactiles de la vue du jeu.

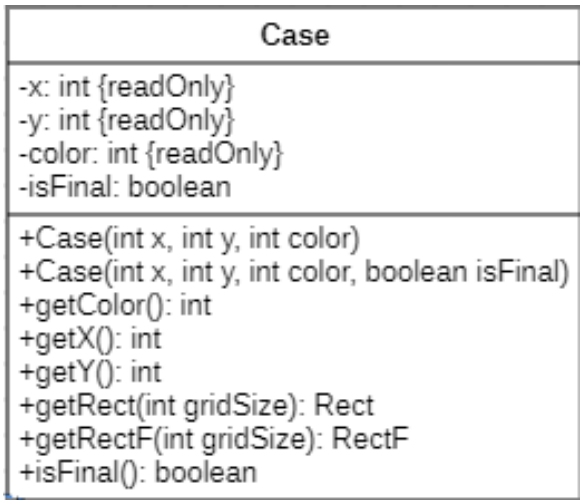
Line.java



Représente une ligne entre deux paires.

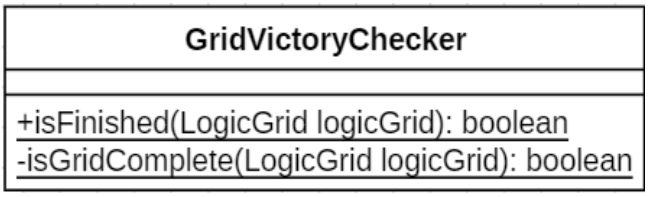
STRUCTURE DU PROGRAMME

Case.java



Représente une case du puzzle. Case possède deux constructeurs, un pour les cases intermédiaires et un pour les cases correspondant à un point d'une paire.

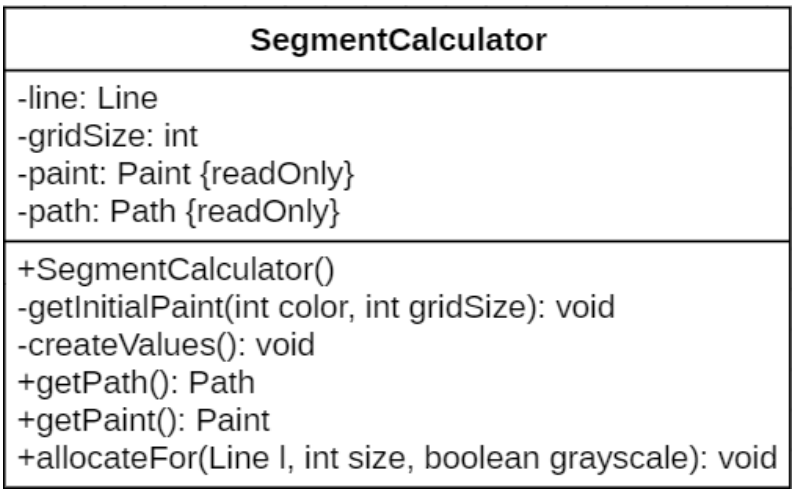
GridVictoryChecker.java



Vérifie si la partie est finie.

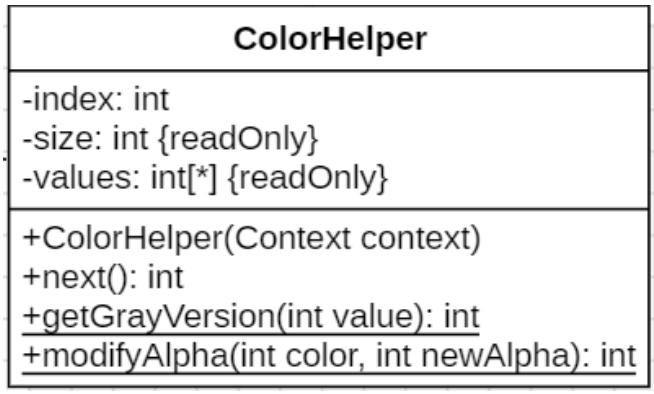
STRUCTURE DU PROGRAMME

SegmentCalculator.java



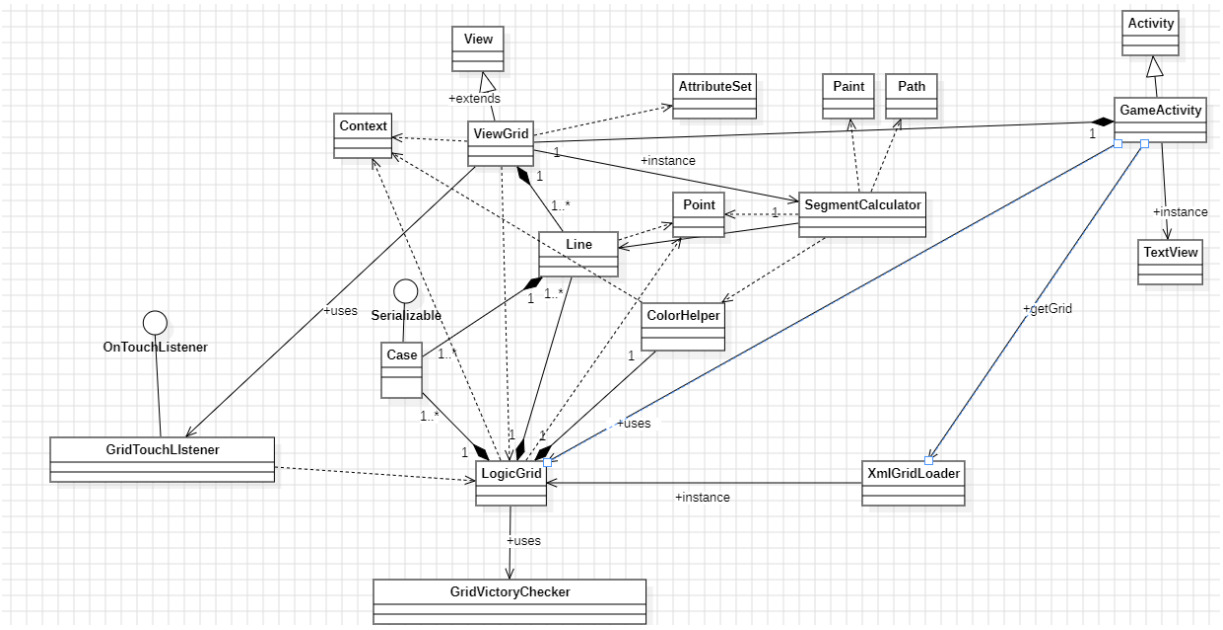
Transforme les données d'une Line en données exploitables pour ViewGrid.

ColorHelper.java



Gère l'utilisation des couleurs de la grille en utilisant le fichier source colors.xml. Nous avons limité le nombre de couleurs utilisées pour les paires dans colors.xml car pour une question technique, d'autres couleurs sont utilisées mais pour le dessin du reste de la vue.

STRUCTURE DU PROGRAMME



Le jeu récupère la grille de jeu grâce à un XmlGridLoader qui instancie la logique du jeu. Le jeu contient la vue du jeu, ViewGrid, qui contient des Line qui contiennent des Case. Elle dépend de la logique du jeu LogicGrid qui la met à jour grâce aux interactions tactiles détectées par GridTouchListener. La vue utilise un SegmentCalculator pour réprésenter les Line du jeu, qui dépend d'un ColorHelper pour gérer les couleurs de la Line.

LogicGrid utilise un GridVictoryChecker pour vérifier si la partie est bien finie, elle est composée des Case et Line du jeu réprésentées dans la vue. Elle possède aussi un ColorHelper pour les couleurs.

IV. Algorithme de résolution

L'algorithme de vérification de résolution du puzzle se décompose en 2 parties :

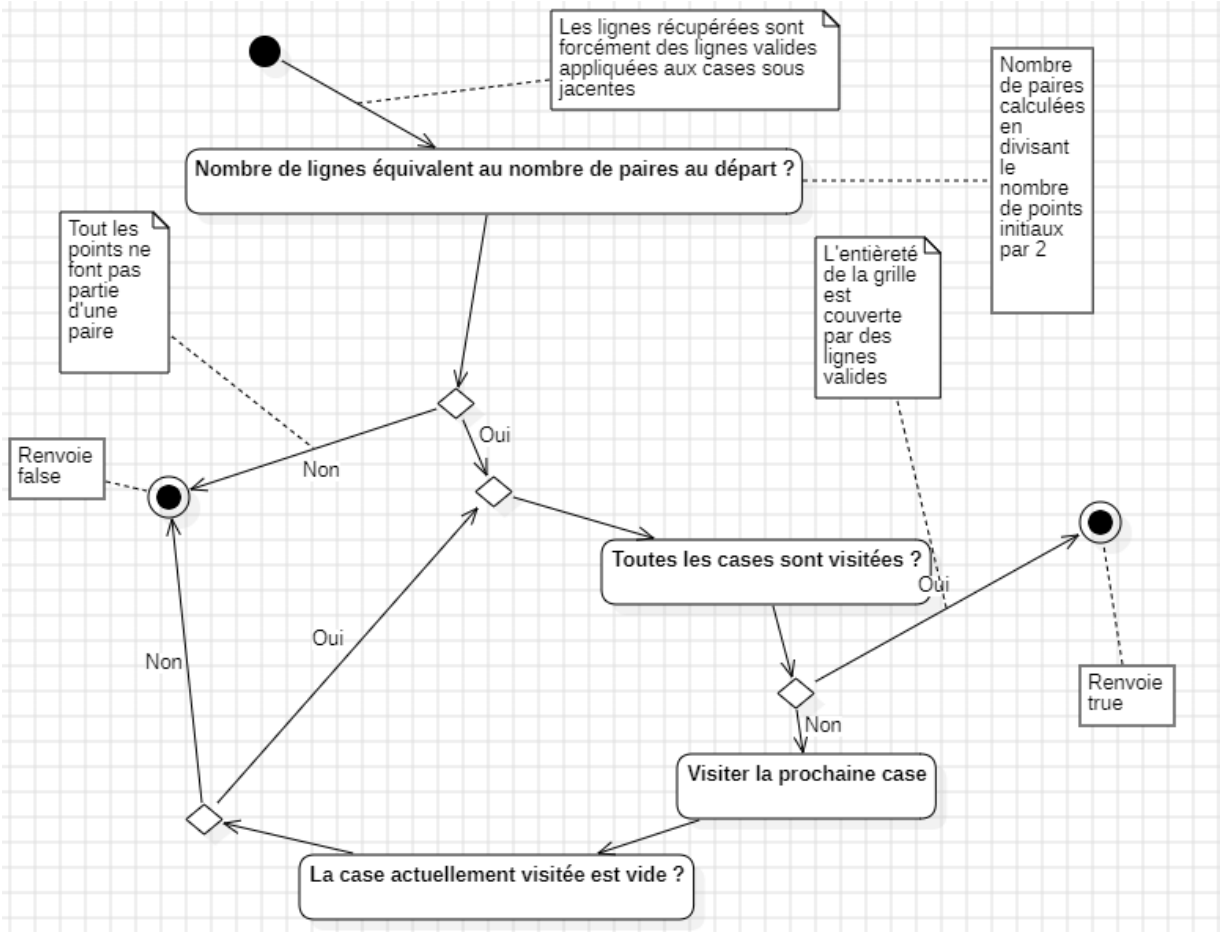
- La vérification que le nombre de paires équivaut à celui de lignes,
- la vérification que toutes les cases soient recouvertes.

Ces 2 conditions sont nécessaires pour que l'algorithme soit considéré comme résolu. La logique inhérente est la suivante :

Les lignes récupérées sont forcément valides reliant une paire entre elle et respectant toutes les conditions. Si le nombre de lignes équivaut à celui de paires (que l'on calcule en divisant le nombre de points par 2) alors on sait que chaque paire est reliée entre elle de manière correcte. Puis si lors du parcours complet de la grille, on détecte une grille vide (i.e. null à la place d'un objet Case), on sait que bien que chaque paire soit complète, la grille n'est pas recouverte. Si ces conditions sont remplies, on sait que la grille est recouverte de lignes tracées entre chaque paire sans case orpheline vu que chaque ligne part forcément d'un point et se termine au point de l'autre extrémité de la paire.

ALGORITHME DE RESOLUTION

Algorigramme de la vérification de résolution du puzzle



V. Conclusion

Nell Telechea :

Ce projet était intéressant car il changeait des autres, après avoir développé des projets pour ordinateur, développer un projet en Android. La représentation du projet en diagramme de classe UML différait par rapport à un projet en java de par la présence plus importante de classes Android dont héritait une majorité de classes ou en dépendait. Le travail d'équipe s'est bien déroulé grâce aux tâches bien réparties et une communication efficace à travers différents moyens de communication.

Benjamin Bribant :

J'ai beaucoup aimé ce projet car j'ai toujours voulu développer un petit jeu ou une petite application sur téléphone, ce qu'il m'a permis de le faire. Ce projet changeait des autres car nous étions en groupe de trois au lieu d'un groupe de deux comme habituellement, ce qui a permis d'avoir moins de tâches par personne et de les répartir plus facilement. Cela nous a permis d'être efficaces et nous n'avons pas rencontré beaucoup de problèmes.

Maxence Raymond :

Cette SAE m'aura permis de découvrir l'utilisation d'un parser. Son appréhension dans le cadre de la SAE s'est avérée ardu comme en témoigne les nombreux changements apportés. Ce travail est un bon complément à la SAE Dorfromantik nous permettant de voir à la fois le développement sous Android et avec Swing pour le bureau. Cette SAE aura néanmoins été agréable via une communication régulière dans un groupé dédié, une liste concrète de points à régler et un report du travail accompli. Cela a permis d'obtenir une application fonctionnelle avec un

CONCLUSION

rendu légèrement moins élaboré que prévu via notamment des difficultés techniques pour la réalisation.