

# Puzzle !

## Installation :

- Installer un serveur Tomcat (voir sur internet la procédure)
- Placer le dossier "puzzle\_data" qui contient les images et les masques sur votre disque accessible en écriture.
- Il est possible de configurer l'emplacement du dossier "puzzle\_data" par une variable d'environnement système. De la même manière que l'on crée la variable "JAVA\_HOME" on peut créer une variable "PUZZLE\_HOME" qui contient le chemin du dossier "puzzle\_data".
- Dans le cas où "puzzle\_data" n'est pas configuré par variable d'environnement, le programme demandera le chemin de "puzzle\_data". Dans ce cas là, un fichier de configuration nommé "puzzle.path.txt" sera créé dans le dossier de Tomcat. Il sera possible de modifier manuellement ce fichier.
- Installer le fichier "Puzzle.war" dans Tomcat, en utilisant le manager.
- Lancer le navigateur internet sur la page de l'application (par exemple : <http://localhost:8080/Puzzle/index.html>). Si le dossier "puzzle\_data" n'est pas encore configuré, une page de configuration de ce fichier apparaît, sinon, la page d'authentification apparaît.

## Utilisation :

### Page d'authentification :

Attention : Le système d'authentification est simplifié. Les mots de passe ne sont pas cryptés.

Pour s'authentifier, entrez le nom d'utilisateur et le mot de passe puis cliquer sur Login. Seules les lettres minuscules sont conservées dans le nom d'utilisateur. Tout autre caractère sera supprimé.

Pour créer un nouvel utilisateur, entrez simplement un nom d'utilisateur et un nouveau mot de passe. Si le nom d'utilisateur n'existe pas encore, il sera créé.

### Menu principal :

Il est possible de charger une sauvegarde, démarrer un nouveau puzzle ou modifier les préférences.

### Préférences :

Il est possible de modifier la couleur de fond de jeu, et de passer automatiquement en mode plein écran pendant le jeu. Le mode plein écran fonctionne mal sur une tablette, il faut donc le désactiver sur ce support.

Le bouton Sauvegarder enregistre les modifications, le bouton Retour annule les modifications.

### Nouveau puzzle :

La page de nouveau puzzle demande de choisir une image, puis la découpe utilisée pour ce puzzle. Le démarrage du jeu commence immédiatement après la sélection.

### Charger une sauvegarde :

Le jeu démarrera dans l'état où l'on l'a laissé (sauf voir la suite).

Attention : Les pièces qui n'ont pas été bougées seront repositionnées dans la zone aléatoire. Il convient de laisser cette zone libre tant que toutes les pièces n'ont pas été déplacées au moins une fois.

#### Le jeu du puzzle :

Pour déplacer les pièces, cliquez et glissez les.

Pour faire tourner les pièces, cliquez une fois sur la pièce. Un cercle rouge apparaît. Cliquez et glissez pour faire tourner la pièce marquée du cercle rouge.

Lorsque deux pièces qui peuvent s'emboîter sont placées côte à côte, le jeu fusionnera ces deux pièces et le compteur de pièces diminuera.

Il est possible d'éparpiller les pièces à l'infini vers le bas et vers la droite de la page.

Pour faire défiler l'écran, cliquez et glissez le à un endroit où il n'y a pas de pièce de puzzle.

Toutes les opérations peuvent être réalisées sur un écran tactile de type "Tablette"

Il est possible de zoomer en utilisant les actions du navigateur : Utilisez la touche contrôle combinée à l'action de la molette de la souris, ou pincez si vous disposez d'un écran tactile.

Pour sauvegarder l'état du puzzle en cours, cliquez sur retour.

Installer de nouveaux puzzles.

Copiez des nouvelles images au format png dans le dossier images du dossier puzzle\_data pour pouvoir jouer avec.

Créez de nouveaux masques de découpes au format png et placez les dans le dossier masks du dossier puzzle\_data pour pouvoir jouer avec.

#### Règles :

La résolution d'une nouvelle image doit correspondre exactement à une résolution d'un des masques de découpe pour pouvoir fonctionner.

Pour créer un masque de découpe, créez une image qui va contenir la silhouette des pièces. Les couleurs utilisées ne doivent pas être transparentes mais peuvent être choisies librement. Le nombre de couleurs utilisables n'est pas limité.

Une pièce ne peut pas avoir deux voisines de la même couleur. Le programme risque de ne pas découper ou emboîter toutes les pièces si cette règle n'est pas respectée.

Examinez les masques de découpes fournis.

## **Documentation technique :**

Le projet utilise le paquet RAD (Rapid Application Development). Pour plus d'information sur le paquet RAD, consultez le projet ERP.

#### Les servlets :

Il existe 4 servlets dans ce projet.

La première est la servlet ApplicationRADServlet.

Cette servlet est fournie par le paquet RAD. Elle ouvre la page de login et sauve la fiche en cours dans la session de l'utilisateur.

Les servlets ServletImages et ServletMasks servent à renvoyer des aperçus en taille réduite des images du puzzle et des masques de découpe.

La servlet ServletPieces permet de renvoyer une image d'une pièce du puzzle en cours identifiée par son numéro et préfixé par un code à 4 chiffres constitué du numéro d'image et du numéro de masque. Le préfixe ne sert qu'à différencier les images pour que les pièces d'un précédent puzzle ne restent pas

dans le cache du navigateur. La méthode `ApplicationParams.removeAntiCachePiece()` retire ce code à 4 chiffres pour pouvoir obtenir le numéro de pièce demandé.  
Les pièces sont conservées dans la `FicheHtml` du jeu.

#### La classe FicheHtml.

Cette classe est essentielle au fonctionnement de l'application RAD.  
Il décrit comment remplir une fiche, comment passer d'une fiche à l'autre et s'occupe du branchement du `WebService`.

#### La classe WebServicePuzzle

Cette classe hérite du `WebService` du paquet RAD.  
Elle sert à transmettre les données de l'application au navigateur Web.  
Il aurait été possible de placer ce code dans la classe `FicheHtml`, mais pour plus de clarté cette classe est dissociée.  
Par héritage, cette classe va pouvoir transmettre toutes les données RAD entre le navigateur Web et le serveur.  
Elle surcharge la méthode `writeResponse()` pour pouvoir ajouter des données au `Json RAD`. Ces données seront envoyées du serveur vers le navigateur Web.  
Les données utiles sont la couleur du fond du jeu, le mode plein écran et les informations du puzzle structurées par la variable `puzzleInfo`.  
La méthode `setValues` est surchargée pour pouvoir récupérer les informations fournies par le navigateur Web. Les informations concernent les pièces déplacées et les pièces fusionnées.  
Dans un premier temps, j'avais tenté de transmettre l'intégralité de l'état du puzzle lorsque l'on ferme la fenêtre de jeu. Malheureusement, cette façon de faire génère une l'entête de requête HTTP trop grande. J'ai donc opté pour une transmission d'information à chaque mouvement de pièce.

#### La classe FicheLogin

Cette classe est spéciale puisqu'elle possède deux fonctions :  
Soit elle affiche la page de d'authentification, soit elle affiche la page de configuration du chemin des données.

Partie authentification :

La méthode `createContent()` crée les composants d'authentification.  
Des composants `HtmlTextFields` sont utilisés pour saisir le nom d'utilisateur et le mot de passe.  
Le bouton Login est un composant `HtmlButton`. Il fonctionne comme un bouton Swing. On renseigne donc le code qui sera exécuté lors d'un clic par la méthode `addActionListener`.  
Si l'authentification est réussie, la fiche du menu est créée par les deux instructions suivantes :

```
FicheHtml ficheMenu = fiche.createFiche("Menu");  
new FicheMenu(ficheMenu, FicheLogin.this.validLogin);
```

Les composants de saisie et le bouton de Login seront placés dans une DIV pour permettre une meilleure mise en page. Cette div est matérialisée par le composant `HtmlContainer`.  
Il faut référencer ces composants dans le `WebService` par la méthode `fiche.ajoutComposantWebService()`. En effet le composant `HtmlContainer` ne permet pas d'ajouter des composants en les liant au `WebService`. Toutes les données qui doivent circuler entre le serveur et le navigateur doivent être liées à un `WebService`.

Dans le cas où le chemin des données n'est pas configuré, la méthode `createPathSetup()` est utilisée à la place de `createContent`.

La méthode `createPathSetup()` permet de générer la page qui demande le chemin des données.

Lorsque le chemin est validé, la fiche de validation du chemin FicheValidPath est générée. Au retour de FicheValidPath, si le chemin de configuration est retenu, une nouvelle fiche de Login est créée. Elle remplace la fiche précédente de la fiche FicheValidPath pour que le retour à la fermeture soit redirigé vers cette nouvelle fiche de login.

#### La classe FicheValidPath

Cette classe crée la fiche de validation du chemin des données de l'application.

Sa structure est similaire à toutes les fiches utilisant le paquet RAD :

Le constructeur a besoin d'une FicheHtml pour pouvoir y ajouter les composants.

La méthode createContent crée le contenu de la fiche, à savoir le bouton retour, du texte détaillant ce que l'application a trouvé dans le chemin de configuration fournit, et un bouton de validation qui va sauvegarder le fichier de configuration.

#### La classe FicheMenu

Elle permet de décrire la fiche du menu principal.

Elle contient un bouton retour pour revenir à l'authentification, un bouton btLoad qui permet de charger un nouveau puzzle, un bouton BtPuzzle qui permet d'ouvrir la fiche FicheNouveauPuzzle et pour finir le bouton btPref qui permet l'ouverture de la fiche de préférences.

Le bouton btLoad, dans la méthode fournie à addActionListener va tenter de lire la sauvegarde dans la structure PuzzleInfo. Ensuite, elle utilise la classe PuzzleCutter pour découper le puzzle. Elle va aussi créer une fiche FichePuzzle et lui fournir les données de sauvegarde puzzleInfo ainsi que les pièces découpées puzzleCutter.getPièces().

#### La classe FichePreferences

Permet de modifier les préférences utilisateur.

Les données sont stockées dans la classe Preference. Cette classe sera sérialisée et désérialisée vers un fichier du système.

La méthode createContent crée la saisie de couleur de fond du jeu et la case à cocher pour le mode plein écran. Il y a aussi les boutons Retour et Sauvegarder.

#### La classe FicheNouveauPullze

La méthode createContent crée les composants de la fiche : La liste de toutes les images et la liste de tous les masques de découpe.

Chaque image et masque possède un événement click : selectImage(e) ou selectMask(e)

À l'initialisation, les masques sont invisibles.

Un clic sur une image (méthode SelectImage) va récupérer la taille de l'image dans les variables width et height, puis rendre tous les container à image invisible. Ensuite elle rend visible les images des masques de découpe dont les dimensions correspondent aux variables width et height. Le titre de la page est aussi modifié.

Un clic sur un masque de découpe (méthode selectMask) va instancier un PuzzleCutter sur l'image et le masque sélectionné.

Les données du PuzzleCutter sont alors fournies à une nouvelle FichePuzzle pour que le jeu puisse démarrer.

La fiche précédente de la fiche puzzle est modifiée pour que son bouton retour ouvre la fiche du menu qui est située dans la fiche précédente de la fiche NouveauPuzzle.

#### La classe FichePuzzle

Cette classe crée le contenu qui servira de support au jeu.

La grosse partie du jeu est réalisée avec JavaScript.

La méthode createContent() crée un bouton retour dont l'action sauvegardera les données du jeu en cours. Elle crée aussi les zones de textes qui serviront à afficher le nombre de pièces du puzzle ainsi que le svg qui représente le cercle rouge affiché lors du mode rotation d'une pièce.

Un bouton btLoad est créé. Il ne possède pas d'addListener puisque les données seront fournies par le WebServicePuzzle lorsqu'un événement clic sera reçu. Ce bouton load est caché, et il possède un code JavaScript RadInit d'initialisation. Le RAD appellera la fonction JavaScript initialisation() lorsque cette fiche sera affichée dans le navigateur.

### La classe PuzzleCutter

Cette classe permet de découper une image selon un masque de découpe.

Note : En premier lieu, la découpe du puzzle a été écrite en JavaScript mais le temps d'exécution était long sur mon PC (environs 5 secondes) et très long sur ma tablette. J'ai donc décidé de réaliser la découpe par le serveur pour que le programme soit agréable à utiliser avec la tablette. Les performances s'en sont trouvées améliorées aussi du côté PC, puisque l'algorithme fonctionne plus rapidement en Java qu'en JavaScript.

Les pièces découpées sont stockées dans la variable pieces ou seront sauvegardés dans des fichiers si destinationPath est renseigné. Les pièces ne sont pas sauvegardés en fichiers lors d'une utilisation normale. La classe RunPuzzleCutter permet de tester le PuzzleCutter en fournissant la destinationPath. Dans ce mode de fonctionnement, le masque de découpe est sauvegardé dans l'état après découpe et permet de vérifier que toutes les pièces ont bien été découpées.

Dans le constructeur, l'image et le masque de découpe sont lus. La variable puzzleInfo est créée et on renseigne déjà la largeur et la hauteur de l'image.

Une image workImage est aussi générée, de taille identique à l'image du puzzle. Cette image servira au dessin des pièces découpées.

La méthode cutPieces est appelée après avoir fourni une première pièce à découper dans la liste piecesToSaw.

La méthode cutPieces() va puiser une pièce à découper dans la liste pieceToSaw et appeler la méthode cutPiece. Cela sera répété tant qu'il y aura des pièces à découper. Lors de la découpe d'une pièce, la méthode cutPiece ajoutera de nouvelles pièces à découper dans la liste lorsqu'elle rencontrera une pièce voisine de la pièce à découper. De cette manière, il suffit de renseigner une première pièce à découper à la méthode cutPieces() pour que toutes les autres pièces soient découpées.

Une pièce à découpée est identifiée par les coordonnées d'un de ses pixels et sa couleur. La première pièce à découpée est donc la pièce de coordonnées 0,0 et de couleur mask.getRGB(0,0)

La méthode cutPiece() découpe une pièce décrite dans le paramètre pieceToSaw. Elle renseigne aussi la structure du puzzle en associant chaque pièce découpée à ses pièces voisines. Chaque pièce voisine rencontrée sera ajoutée à la liste des pièces à découper.

Cette méthode utilise l'algorithme de remplissage d'une surface par diffusion. J'ai dû utiliser une pile explicite puisque la version récursive créait un débordement de pile avec JavaScript.

Voici l'algorithme de remplissage d'une surface par diffusion avec une pile :

```

Remplissage(x,y,couleurRemplissage)
Empiler x et y
TanQue la pile n'est pas vide faire :
    Dépiler x et y
    Si x et y sont >= 0 et qu'ils sont < à la taille de l'image alors
        Si la couleur du pixel(x,y) <> couleurRemplissage alors
            Colorier pixel(x,y) avec couleurRemplissage
            Empiler x+1 et y
            Empiler x-1 et y
            Empiler x et y+1
            Empiler x et y-1
        Fin si
    Fin si
Fin TanQue

```

La méthode cutPiece() fait quelque chose de similaire.

Elle va remplir le masque de découpe avec la couleur correspondant au numéro de pièce. La couleur vaudra 1 pour la pièce 1, 2 pour la 2 etc..

Puisque l'image utilise la transparence, les couleurs opaques sont négatives. Donc chaque pixel de couleur négative du masque n'est pas encore découpé et les couleurs positives représentent les numéros de pièce.

En même temps que le masque est rempli avec le numéro de pièce à découper, une copie de chaque pixel de l'image du puzzle est copiée dans la workImage. Toujours dans la boucle TanQue, les bornes de la pièce sont calculés dans xMin, xMax, yMin et yMax. Enfin, toujours dans la boucle TanQue, si le pixel du masque n'est pas de la couleur attendu elle fera appel à la méthode addPieceToSaw() pour ajouter une nouvelle pièce voisine à la liste des pièces à découper (sauf si la couleur est positive et représente un pixel déjà découpé).

Lorsque tous les pixels de la pièce à découper ont été traités par la boucle TanQue la workImage contient la pièce découpée du puzzle positionnée dans le rectangle délimité par xMin, xMax, yMin et yMax. Cette image est sauvegardée dans une nouvelle image de pièce ou dans un fichier grâce à la méthode savePiece(piece).

La avant la boucle TanQue, la méthode cutPièce vide l'image de travail workImage avec workGraphics.

Voisins :

La Méthode addPièceToSaw ajoute son premier voisin puisqu'il est le numéro de la pièce traité au moment de son appel. Le numéro de cette pièce correspond à puzzleInfo.piecesInfo.size() puisque la pièce en cours ajoutera une nouvelle valeur la liste pieceInfo.

La méthode cutPiece commence par tester la couleur du pixel du masque à l'endroit où la nouvelle pièce doit être découpée. Si la couleur est positive, alors c'est une pièce déjà découpée et la couleur représente le numéro de la pièce. Dans ce cas là les nouveaux voisins seront présentés : Le voisin de la pièce à découpé sera ajouté au voisin de la pièce déjà découpée et vice versa. La pièce ne sera pas découpée et l'exécution de la méthode est stoppée par l'instruction return.

[index.html](#)

Le fichier index.html contient seulement une div nommée idRadApplication pour que le RAD puisse fonctionner. Il intègre aussi le script rad.js qui est indispensable aux applications RAD, enfin le script puzzle.js, qui doit être placé après rad.js.

puzzle.js

### Initialisation

Le fichier puzzle.js est destiné à être exécuté pendant la phase de jeu, mais en réalité, il est exécuté dès le début de l'application, à partir de la fenêtre de login.

On peut y trouver la registration au RAD :

```
RegisterRADFillExtra(function(json) {  
    ...  
});  
  
RegisterRADEventExtra(function() {  
    return ...;  
})
```

L'appel à ces fonctions permet au programme de communiquer en utilisant le WebService RAD. RegisterRADFillExtra permet de recevoir des données. Un objet Json est fourni contenant toutes les informations concernant le RAD, mais aussi les informations que le WebServicePuzzle a ajouté. RegisterRADEventExtra permet d'envoyer des données vers le WebServicePuzzle. La fonction fournie doit renvoyer une chaîne de caractères.

Les données reçues concernent les préférences de jeu et les informations décrivant le puzzle. Les données envoyées concernent la pièce qui vient d'être déplacée ou emboîtée. Le tag requestPuzzleInfo est aussi transmis. Il indique au serveur si une réponse doit contenir les informations du puzzle. Il sera mis à 1 lorsque le jeu démarre, lors de son initialisation.

La fonction initialisation() permet d'initialiser le début du jeu. Elle sera appelée par le RAD lors de l'affichage de la FichePuzzle.java. L'instruction de FichePuzzle.java qui déclenche cet appel est la suivante :

```
btLoad.setRadInit("initialisation");
```

L'initialisation va mettre en place les événements souris et touch. Elle va aussi retenir la référence du SVG illustrant le mode rotation ainsi que de la zone de texte affichant le nombre de pièce restantes du puzzle.

Lors de l'initialisation, un appel est fait à RegisterRadDestroy. La fonction passée en paramètre sera appelée lorsque le RAD détruira la page en cours pour en afficher un autre. Cette fonction sera donc exécutée après un clic sur le bouton Retour.

La fonction setinfoPieces() est appelée par RegisterRADFillExtra. Lorsque les informations d'un nouveau puzzle sont reçues, cette fonction va déclencher la création et lecture des images des pièces par un appel à createImages. Un appel à createTempPiece permet de créer la image temporaire qui servira à assembler des pièces.

La fonction createImages() va créer des éléments IMG qui auront pour source l'URL de chaque pièce du puzzle. Un événement onload leur est ajouté pour pouvoir faire appel aux fonctions piecesLoaded() et setPiecesFusionnes() lorsque toutes les images ont été reçues dans le navigateur. Le compteur imagesLoaded permet de savoir si toutes les images ont bien été reçues.

La fonction piecesLoaded va transformer les IMG en CANVAS. En effet, les opérations que demande le programme ne peut pas fonctionner avec des IMG. Pour chaque IMG, Un nouveau canvas est créé, et reçoit une copie du dessin de la pièce de puzzle. La position de la nouvelle pièce est fixée aléatoirement.

La fonction `setPiecesFusionnees()` va parcourir le tableau `piecesFusionnees` provenant d'une partie sauvegardée et va réaliser la fusion de ces pièces par l'appel à la fonction `fusionnePieces()`.

#### Quelques variables globales :

`pieces[]` contient le tableau des canvas des pièces.

`ctxpieces[]` : tableau des contextes des canvas des pièces.

`InfoPieces[]` : tableau de tableau d'informations de pièces.

Exemple `infoPieces[3]` donne le tableau d'information de la pièce numéro 3 à savoir :

`[offsetX,offsetY,[tableau de voisines],angle,x,y]`

où : `offsetX` et `offsetY` sont les coordonnées de la pièce sur le dessin d'origine.

`[tableau de voisines]` est le tableau contenant la liste des pièces emboîtables à la pièce décrite.

`angle` est l'angle de rotation de la pièce sur le jeu.

`x,y` sont les coordonnées de la pièce sur le jeu.

#### Phase de jeu

Pendant la phase de jeu, le programme va réagir aux événements de la souris et au touch.

Les événements vont être :

- `screenDown` lors d'un `mousedown` ou `touchstart`
- `screenMove` lors d'un `mousemove` ou un `touchmove`
- `screenUp` lors d'un `mouseup`, `touchend` ou `touchcancel`

La fonction `screenDown()` commence par un test pour savoir si on est en mode rotation de pièce.

Dans ce cas la variable `inRotation` est modifiée.

La variable `inRotation` vaut 0 si on est pas en mode rotation, 1 lorsque l'on a cliqué sur une pièce, 2 si on passe en mode rotation, 3 si on est en mode rotation et que l'on a cliqué pour commencer une rotation et 4 quand on a commencé à faire tourner la pièce.

La boucle `for` va parcourir chaque pièce du puzzle pour savoir si on a cliqué dessus. Je ne voulais pas utiliser un événement pour chaque pièce puisque je voulais que la pièce cliquée soit la bonne, en tenant compte de la transparence, donc de leur forme.

Il y a un changement de repère qui est effectué. En effet, il faut trouver la couleur du pixel de la pièce sur laquelle on a cliqué en tenant compte de sa position et de sa rotation.

Les variables `x` et `y` contiennent la coordonnée du clic sur la fenêtre.

Les variables `evx` et `evy` sont ensuite positionnées sur la coordonnée du clic, par rapport au centre de rotation de la pièce. `evy` est inversé pour pouvoir correspondre à un axe Y mathématique qui va dans le sens opposé de la coordonnée Y de l'écran. La constante `angle` contient l'angle de rotation de la pièce exprimé en radian. Cet angle est l'angle affiché et il va dans le sens inverse du sens trigonométrique.

Les constantes `revx` et `revy` contiennent les coordonnées du clic dans le repère qui a son origine au centre de la pièce et qui est tourné dans l'angle de rotation de la pièce.

Les variables `evx` et `evy` sont encore calculées pour pouvoir être la coordonnée du clic dans le repère tourné de la pièce dont l'origine est le coin supérieur haut gauche en utilisant un axe Y inversé tel une coordonnée d'écran.

A ce moment là il ne reste qu'à tester si le point de clic de coordonnée `evx`, `evy` est situé dans la pièce et le cas échéant, aller chercher la couleur pour vérifier si il est transparent ou pas. Dans le cas où `alpha=255`, le pixel cliqué n'est pas transparent, la variable `laPièce` garde en mémoire le numéro de la pièce cliquée.

Si une pièce a été cliquée, la variable `movingPièce` est affecté au numéro de la pièce cliquée. Par ailleurs les `zIndex` des pièces sont recalculés pour que la pièce cliquée monte au premier plan.

La fonction `screenMove()` va, selon le mode en cours, déplacer la pièce sélectionnée ou la faire

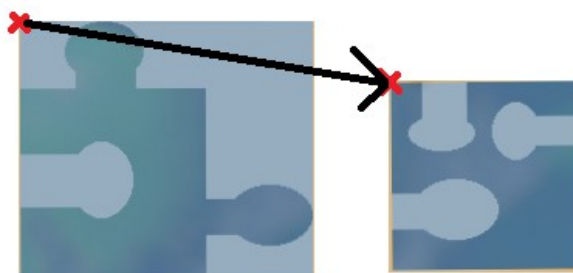


tourner. Si la pièce sélectionnée (variable `movingPiece`) est à -1, alors rien ne se passe. Le scrolling de l'écran est fait uniquement dans l'événement `mouseMove` puisque l'écran tactile réalise déjà par défaut un scrolling.

La fonction `screenUp()` se produit lorsque une pièce a été tournée ou déplacée. (Le premier test sur la variable `movingPiece` sort de la fonction si aucune pièce n'a été sélectionnée). La première partie de la fonction gère la visibilité du cercle rouge concernant la rotation.

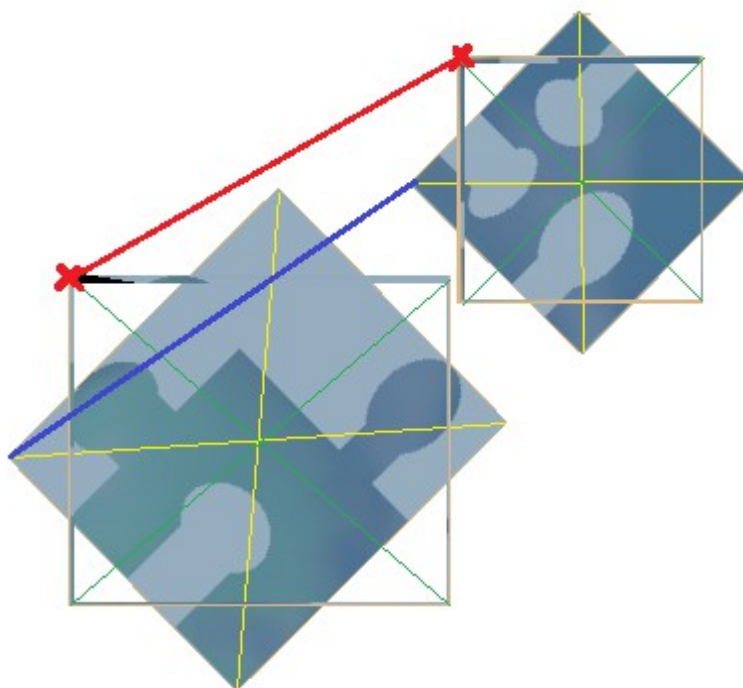
Ensuite il y a une boucle `for` sur les voisins de la pièce déplacée. Le but est de tester si la pièce déplacée est en contact avec une de ses pièces voisine et peut s'emboîter dedans.

La fonction `findNeighborPosition` va calculer la position que devrait avoir une pièce voisine pour pouvoir s'emboîter dans une autre pièce. C'est un calcul assez compliqué :



Quand les pièces sont droites, c'est plutôt simple : Il suffit de comparer le vecteur de la différence des positions des pièces par rapport au vecteur de la différence de leurs offset. L'offset est la position de la pièce dans l'image d'origine, donc si l'écart entre les deux pièces coïncide alors la pièce s'emboîte.

Lorsqu'il y a une rotation, tout se complique :



Le vecteur de la différence des offsets en bleu ne correspond plus au vecteur de la différence des coordonnées en rouge. La rotation de la pièce par rapport à son centre fait déplacer différemment les offsets des pièces selon leur taille. J'ai finalement réussi à trouver une formule qui fonctionne.

## Conclusion :

Je suis plutôt satisfait, puisque le jeu fonctionne bien et il est agréable à jouer.

### Les grosse difficultés rencontrés ont été :

Trouver la liste des voisins emboîtable d'une pièce lors de la découpe. Pour cette raison, dans le masque de découpe, une pièce donnée ne doit pas contenir deux pièces voisine de couleur identique.

Trouver la couleur du pixel de la pièce cliquée pour savoir si elle est transparente ou pas. La difficulté est due à la rotation.

Trouver si une pièce déplacée peut s'emboîter dans une de ses pièces voisines : La rotation complique beaucoup la tâche.

### Problèmes restant :

Sur la tablette, le mode plein écran désactive la possibilité de zoom et de scrolling.

Toujours sur la tablette, Il est possible de scroller vers la gauche lorsque les pièces dépassent l'affichage, mais pour pouvoir scroller vers le bas, il faut agrandir l'écran par la gauche en y glissant une pièce.

Lorsque la session Tomcat expire, le serveur rencontre une erreur null pointer exception. Il faut alors rafraîchir la page du navigateur web avec la touche F5.

J'espère que vous passerez un moment agréable à réaliser des puzzles.