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 créé la variable "JAVA_HOME" on peut créér une variable "PUZZLE_HOME" qui contient le chemin du dossier "puzzle_data".
- Dans le cas ou "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 conservés 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 mofifier 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 modification, 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 ou 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'on pas été déplacés 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 apparait. Cliquez et glissez pour faire tourner la pièce marquée du cercle rouge.
Lorsque deux pièces qui peuvent s'emboîter son 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 glisser le à un endroit ou il 'n'y a pas de pièce de puzzle.
Toutes les opération peuvent être réalisées sur un écran tactile de type "Tablette"
Il est possible de zommer en utilisant les actions du navigateur : Utilisez la touche contrôle combiné 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 fomat png dans le dossier images du dossier puzzle_data pour pouvoir jouer avec.
Créez de nouveaux masques de découpes au fomat png et placez les dans le dossier masks du dossier puzzle_data pour pouvoir jouer avec.

Astuce : Utilisez le programme Cloanto Personal Paint pour Amiga pour réaliser les masques de découpes :
Installer WinUAE, trouver la ROM 3.0, les ADF du workbench 3.1 et de Cloanto Personal Paint ainsi que le driver Picasso 96 (Picasso96.lha et lha.run pour le désarchiver sur www.Aminet.net). Dans la configuration UAE utilisez un CPU 68040, Chipset AGA, ROM v3.0, 2Mb de chip RAM, 512 MB de 32-bit Chip, RTG board : UAE[ZORROIII] avec VRAM size 1GB.
Pour le Hard drive, ajoutez un Directory de l'ordinateur en tant que disque dur, device name hd0, volume label hd0, bootable.
*Booter sur l'installation du Workbench. Installer sur le disque dur, installer ensuite Picasso96, puis Personal Paint. On peut alors créer une image en 1920*1080 en 16 couleurs et dessiner les masque de découpes avec ce programme. Pour convertir l'image en PNG, utilisez une capture d'écran à coller dans Microsoft Paint pour convertir en format PNG. Pour réaliser la capture d'écran, cachez les menus et le pointeur de la souris dans Personal Paint avec les touches F9 F10 et Suppr(Del) et cachez le pixel de dessin sur un fond de couleur identique.*

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 utilisable 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 servlet 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 chiffre constitué du numéro d'image et du numéro de masque. Le préfixe ne sert qu'a différencier les images pour que les pièces d'un précédent puzzle ne reste pas dans le cache du navigateur. La méthode ApplicationParams.removeAntiCachePiece() retire ce code à 4 chiffre pour pouvoir obtenir le numéro de pièce demandé.
Les pièces sont conservés 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 la 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 envoyés 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 structurés par la variable puzzleInfo.
La méthode setValues est surchargée pour pouvoir récupérer les informations fournies par le navigateur Web. Les informations concerne 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 crée 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.ajoutCompodantWebService(). En effet le composant HtmlContainer ne permet pas d'ajouter des composants en les liants au webService. Toutes les données qui doivent circuler entre le serveur et le navigateur doivent être liées à un webService.

Dans le cas ou 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 crée. 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 créé 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 à besoin d'une FicheHtml pour pouvoir y ajouter les composants.
La méthode createContent créé le contenu de la fiche, à savoir le bouton retour, du texte détaillant ce que l'application à 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 a 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 ficher 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)

A l'initialisation, les masques sont invisibles.
Un clic sur une image (méthode SelectImage) va récupérer la taille de l'image dans les varriables width et height, puis rendre touts 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 fournis 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 que est située dans la fiche précédente de la fiche NouveauPuzzle.

La classe FichePuzzle
Cette classe créé le contenu qui servira de support au jeu.
La grosse partie du jeu est réalisée avec JavaScript.
La méthode createContent() créé un bouton retour dont l'action sauvegardera les données du jeu en cours. Elle créé 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'actionListener 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 coté 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 PuzzleCuter en founissant 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 on bien été découpées.

Dans le constructeur, l'image et le masque de découpe sont lus. La varriable 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 et appelée après avoir fourni une première pièce à découper dans la liste piècesToSaw.

La méthode cutPieces() var puiser une pièce à découper dans la liste pieceToSaw et appeler la méthode cutPiece. Cela sera répété tant qu'il y aura des pièce 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 cutPièces() pour que toutes les autres pièces soient découpées.
Une pièce à découpée est identifiée par les coordonnées d'un de ses pixel et sa couleur. La première pièce à découpée est donc la pièce de coordonnée 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 découpées à ses pièces voisines. Chaque pièce voisine rencontrée sera ajouté à la liste des pièces à découper.
Cette méthode utiliser l'algorithme de remplissage d'une surface par diffusion. J'ai du 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 et 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) 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 founit contenant toutes les informations concernant le RAD, mais aussi les informations que le WebServicePuzzle à 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 reçus, 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 et crée, et reçoi 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 vaux 0 si on est pas en mode rotation, 1lorsque 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 cliqué 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 radiant. 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 calculée 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 que a 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 ou alpha=255, le pixel cliqué n'est pas transparent, la variable laPièce garde en mémoire le numéro de la pièce cliqué.

Si une pièce à 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 cliqué monte au premier plan.

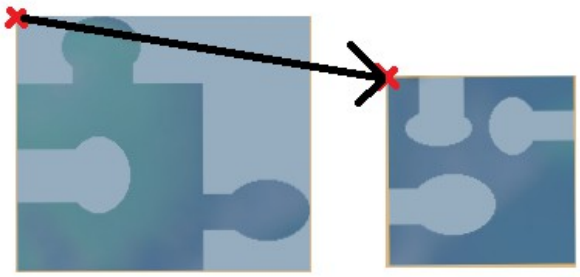
Astuce : Trouver la formule de rotation :
Pour faire une rotation d'un point P autour de l'origine, Il faut multiplier ce point par un nombre complexe de norme 1
Les nombres réels sont les nombres de tous les jours, tels que 1, 5, 2.5, Pi, Racine de 2, 2/3.
Un nombre imaginaire est un nombre réel que l'on multiplie par i.
Qu'est ce que i ?
*i est un nombre imaginaire, tout ce que l'on sait sur lui c'est que i*i = -1 (i au carré = -1)*
Un nombre complexe est la somme d'un nombre réel et d'un nombre imaginaire.
On peut représenter un nombre complexe sur un graphique. La partie réelle représente l'axe des X et la partie imaginaire représente l'axe des Y.
Le point de coordonnée x=2 et y=3 est donc représenté par le nombre complexe 2+3i.
*Un point P(x,y) s'écrit alors x+y*i.*
Pour faire tourner un point d'un angle a autour de l'origine, on le multiplie par un complexe de norme 1 et d'angle a.
*Ce point de rotation (R) à pour coordonnée x = cos(a) et y = sin(a). Le nombre complexe est donc cos(a) + i*sin(a)*
*Pour faire tourner le point P d'un angle a, il faut donc faire la multiplication P*R:*
*soit : (x+y*i) * (cos(a) + i*sin(a)).*
*= x*cos(a) + x*i*sin(a) +y*i*cos(a) + y*i*i*sin(a)*
*Je met i en facteur et i*i = -1 donc :*
*= x*cos(a) + i*(x*sin(a) +y*cos(a)) - y*sin(a)*

La rotation P'(x',y') d'un point P(x,y) par un angle a vaut donc :
*x' = x*cos(a) - y*sin(a)*
*y' = x*sin(a) + y*cos(a)*

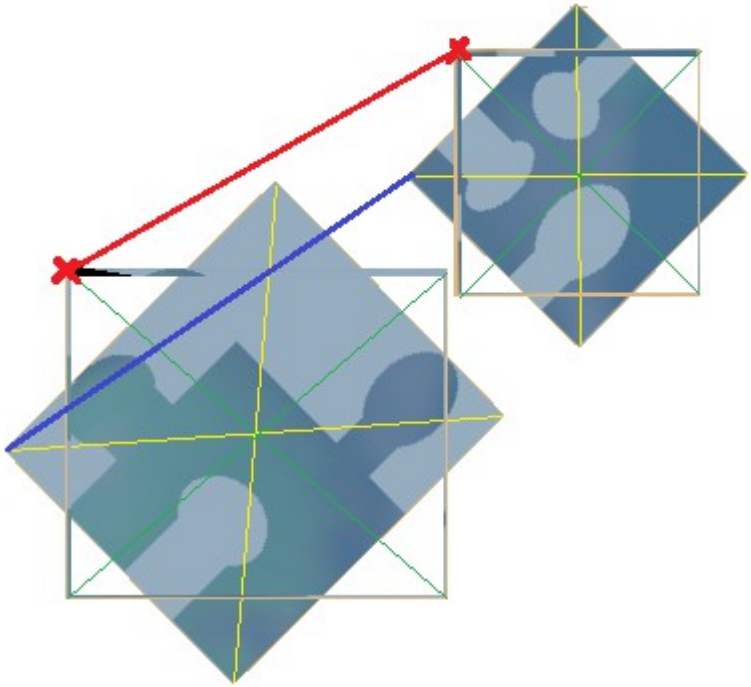
La fonction screenMove() va, selon le mode en cours, déplacer la pièce sélectionnée ou la faire tourner. Si la pièce sélectionnée (variable movingPiece) est à -1, alors rien ne se passe. Le scrooling de l'écran est fait uniquement dans l'événement mouseMove puisque l'écran tactile réalise déjà par défaut un scrooling.

La fonction screenUp() se produit lorsque une pièce à été tournée ou déplacée. (Le premier test sur la variable movingPièce 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 findNeigborPosition 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 déplace différemment les offsets des pièce 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 étés :

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 scrooling.
Toujours sur la tablette, Il est possible de scrooler vers la gauche lorsque les pièces dépassent l'affichage, mais pour pouvoir scrooler 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.

Amélioration des versions :

Correction de fautes de français dans la documentation et ajout de la formule de rotation et utilisation de Cloanto Personal Paint.
Correction d'un bug : Le jeu ne prenait pas en compte la couleur de fond personnalisée lors d'un nouveau puzzle.
Ajout d'un test lors de la découpe du puzzle. Si une pièce est trop petite, un message d'avertissement est affiché. Ceci permet de vérifier qu'une découpe de puzzle ne possède pas de pixels isolés.
Ajout du masque de découpe circulaire.