Projet démo PCII

Introduction

Voici la documentation du projet de démo dans le cadre du projet PCII

Vous trouverez 7 sections dans la documentation qui suit:

1. Document d’analyse
2. Plan de développement
3. Conception
4. Résultats
5. Documentation utilisateur
6. Documentation développeur
7. Conclusion et Perspectives
8. Document d’analyse :

Analyse globale :

Nous devons réaliser une Interface graphique ce qui inclut l’ovale, la ligne brisée, un menu, le score et un fond animé, couplé de la génération et du défilement continu de la ligne brisée ainsi que le mouvement contrôlé de l’ovale.

Analyse détaillée :

Pour réaliser ce qui a été présenté dans l’analyse globale nous réaliserons de manière plus détaillée les objectifs énoncés ci-dessous :

* Dessin de l’ovale, difficulté basse priorité 1
* Montée de l’ovale lors d’un click, difficulté moyenne, priorité 1
* Descente permanente de l’ovale, difficulté basse, priorité 1
* Rajout de vitesse, difficulté moyenne, priorité 2
* Dessin de la ligne brisée, difficulté basse priorité 1
* Génération de la ligne brisée, difficulté moyenne priorité 1
* Défilement de la ligne brisée, difficulté moyenne priorité 1
* Affichage menu, difficulté simple priorité 2
* Bouton d’accès au jeu, difficulté simple priorité 2
* Score du jeu, difficulté simple priorité 2
* Affichage de l’image, difficulté simple priorité 2
* Défilement perpétuel de l’image, difficulté moyenne priorité 2

1. Plan de développement

Tâches :

Voici les différentes tâches à réaliser pour accomplir les objectifs spécifiés dans l’analyse détaillée, section 1.

* Apprentissage Swing : 1h
* Apprentissage threads : 30 min
* Dessin de l’ovale : 15 min
* Mouvement de l’ovale lors d’un click : 15min
* Thread de mouvement pour la descente de l’ovale : 15 min
* Rajout de vitesse : 30 min
* Rédaction document analyse et conception : 2h
* Réorganisation du code en packages : 45 min
* Génération de la ligne brisée : 30 min
* Dessin de la ligne brisée : 15 min
* Défilement de la ligne brisée : 45 min
* Affichage menu : 15 min
* Buuton d’accès au jeu : 15 min
* Affichage image de fond : 10 min
* Défilement perpétuel de l’image de fond : 30 min

Ressources :

1 seule

Diagramme de Gant :

Voici l’organisation détaillée du travail pour la seule ressource disponible :



1. Conception :

Conception générale :

Patron MVC :

Nous allons utiliser le patron modèle, vue, contrôleur. Ce modèle offre une simplicité dans la réalisation et la compréhension du projet et il est également très usité en dehors de ce projet. Pour ces raisons et aux vues des nombreux avantages qu’il présente nous avons décidé de l’utiliser. Voici un diagramme représentant le modèle dans notre projet :

**Vue**

Dessin ovale

Dessin ligne

Affichage menu

Affichage image fond

**Contrôleur**

Clic souris

Thread chute

Thread repaint

Thread ligne

Button d’accès

Défilement image

**Modèle**

Vitesse

Hauteur

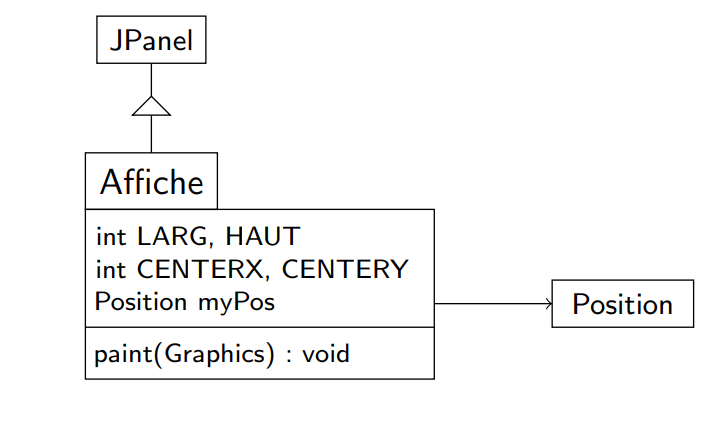
Position image

Ligne

Conception détaillée :

Dessin de l’ovale :

Pour le dessin de l’ovale, il n’y a qu’une classe d’affichage dans laquelle on dessine l’ovale en question



Mouvement de l’ovale lors du clic souris :

* Programmation événementielle avec la classe MouseAdapter
* Vitesse
* Lorsque l’on clique sur la souris on fait un appel à jump qui change la position de l’ovale

MouseAdapter

ReactionClic

mouseClicked(MouseEvent): void

Position

int vitesse

int JHEIGHT

jump(int): void

get(): int

Chute Ovale :

* Thread (et constante pour la vitesse)
* Hauteur
* Le thread appelle la fonction move avec un délais, la fonction move modifie la position de l’ovale dans la direction opposée à la fonction jump sauf quand l’ovale a atteint le bas du cadre dans ce cas on n’appelle pas la fonction move.

Thread

Descendre

int DELAY

run(): void

Position

int height, vitesse

int JHEIGHT

get(): int

move(): void

Affichage :

* Une classe avec un Jpanel qui représente l’affichage du jeu
* Un thread qui actualise ce Jpanel pour prendre en compte les changements

repaint(): void

revalidate(): void

Affichage

JPanel

int DELAY

run(): void

sleep(int): void

Redessine

Thread

Génération ligne :

Algorithme de génération :

Entrée : XMIN, XMAX, YMIN, YMAX, XDEPART, XFIN, YDEPART, LARG\_FENETRE

(YMIN >= bas de la fenêtre, YMAX <= haut de la fenêtre)

Sortie : Liste(Point)

L = [(0-XDEPART, YDEPART), (XDEPART, YDEPART)]

X = XDEPART + XMAX

Tant que X <= LARG\_FENETRE + XFIN

Last\_point = Get\_last\_point()

Y\_last\_point = Last\_point.y

Xr = Random(XMIN, XMAX)

Faire :

Ratio = Random (0, 10)

Yr = Y\_last\_point + (Ratio-5)\*10

Tant que Yr > YMAX ou Yr < YMIN

X = X + Xr

L += [(X, Yr)]

Défilement et affichage de la ligne :

* Position réelle des points avec le Array points
* Position affichée donnée par l’appel à la méthode get\_points()
* Position affichée = Position réelle – avancement (classe position)
* Mise à jour constante d’avancement avec le thread Avance
* Algorithme de génération et destruction des points, le thread redessine appelle la méthode get\_points() il n’y a donc pas besoin de boucle while dans l’algorithme ci-dessous :

If points2[0].x < 0 – XDEPART then Remove(Points[0])

If points2[Size(points2) – 1].x < LARG\_FENETRE + XFIN then

Yr = 0

X = Get\_X(points, Size(points) – 1)

Last\_point = Get\_last\_point()

Y\_last\_point = Last\_point.y

Xr = Random(XMIN, XMAX)

Faire :

Ratio = Random (0, 10)

Yr = Y\_last\_point + (Ratio-5)\*10

Tant que Yr > YMAX ou Yr < YMIN

X = X + Xr

L += [(X, Yr)]

Parcours

ArrayList<Points> points

generate\_points(): void

get\_points(): ArrayList<Points>

Position

Int avancement

avance(): void

Thread

Avance

int DELAY

run(): void

sleep(int): void

Affichage du menu :

* Une classe menu qui hérite de Jpanel et qui possède un bouton. Lorsque l’on clique sur ce bouton on crée une instance de la classe jeu détaillée dans la section suivante et on lance la méthode Jeu.game sur la JFrame courante.

JFrame

JPanel

Menu

JButton myButton

Classe jeu :

* La méthode game remplace le Jpanel menu par le Jpanel du jeu. Elle crée et elle lance tous les threads associés au jeu. Elle crée aussi les instances des classes utiles au jeu.

Jeu

game(JFrame): void

Affichage de l’image :

* Classe de la position de l’image (il y a en réalité deux images dont les positions sont distinctes)
* L’image en tant que telle est ouverte et conservée dans la classe affichage deux fois
* Algorithme du changement d’images pour le défilement dans la fonction drawImages (il n’y a pas de boucle car cet algorithme est appelé par un thread) :

Entrée: image1, image2, ImagePosition, FWIDTH

If (ImagePosition.get\_pos2() + image2.width() == FWIDTH)

Then ImagePosition.set1(FWIDTH)

If (ImagePosition.get\_pos1() + image1.width() == FWIDTH)

Then ImagePosition.set2(FWIDTH)

ImagePosition

Int pos1, pos2

get\_pos1() : int

get\_pos2() : int

set\_pos1(int) : void

set\_pos1(int) : void

JPanel

Affichage

drawImages(Graphics):void

Défilement perpétuel de l’image :

* Il y a deux images
* Le thread appelle move sur les deux positions, ce qui fait bouger les images constamment. On échange les deux images pour qu’il y en ait toujours une dans le cadre avec l’algorithme décrit ci-dessus
* Les deux images sont parfaitement côte-à-côte, elles échangent juste de place

ImagePosition

Int pos1, pos2

move\_pos1() : void

move\_pos2() : void

Thread

Imagechange

run() : void

1. Résultats :

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

Une image contenant capture d’écran, texte, Tracé, diagramme

Description générée automatiquement

1. Documentation utilisateur :

* Prérequis : Java avec un IDE
* Mode d’emploi : Importez le projet dans votre IDE, sélectionnez la classe Main à la racine du projet puis cliquez sur « Run ».
* Cliquez sur le bouton « Cliquez pour commencer »
* Cliquez sur la fenêtre pour faire monter l’ovale

1. Documentation développeur

* La classe principale est main, elle crée une JFrame et affiche le menu dans celui-ci
* Les bibliothèques externes utilisées sont swing, AWT et thread
* Pour continuer le développement de la collision entre la ligne et l’ovale il faudra utiliser les classes Position et Parcours.

1. Conclusion et perspectives

* Le travail est plutôt satisfaisant du fait que l’on arrive à obtenir un jeu visuellement agréable et sympathique, dont les performances sont plutôt correctes par rapport à nos attentes. Par ailleurs le jeu est presque jouable malgré le fait qu’il manque quelques fonctionnalités. Dans l’ensemble j’ai pris du plaisir à coder et à voir les résultats apparaître au fur et à mesure.
* Il existe la possibilité de finir le jeu et d’implémenter les collisions pour que le joueur puisse perdre