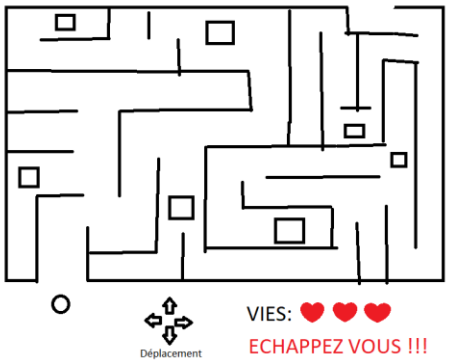
Projet du jeu LabyLu

Nom du projet : Labyrinthe

Numéro du projet : 4

Nom de l'élève : Lucie LE FUR

Dessin à main levée l'allure du logiciel (graphique / texte)



Le rond = Le Joueur

Carré = méchants

Explication détaillant le fonctionnement de l'application

Le jeu consiste à ce que le joueur s'échappe du labyrinthe en évitant les méchants en mouvement en un minimum de temps qui est de 60s.

Le joueur a 3 vies s'il rentre en collision avec un méchant il perd une vie et revient au début.

Dans le labyrinthe il existe des sabliers, ses sabliers permettent au joueur de lui donner du temps supplémentaire qui est de 10s. Si le joueur rencontre un sablier, il disparaît.

Un sablier sera considéré comme un mur (obstacle) pour un méchant.

Le joueur perd si :

- Il n'a plus de vie
- il n'a pas réussi à sortir à temps

Si le joueur perd, on lui indique qu'il a perdu et le programme s'arrête.

Dans le jeu, on indiquera le nombre de vie du joueur et le temps qu'il lui reste pour sortir du labyrinthe.

Exemples concrets sur le fonctionnement de l'application :

- Exemple N°1:

Le joueur se trouve dans le labyrinthe, il ne lui reste qu'une vie. Un méchant le touche. Il a donc perdu toute ses vie, le programme indique au joueur qu'il a perdu et le jeu s'arrête.

PERDU !

- Exemple N°2:

Le joueur se trouve dans le labyrinthe, il ne lui reste plus de temps pour sortir. Il a donc perdu il n'a plus de temps, le programme indique au joueur qu'il a perdu et le jeu s'arrête.

PERDU !

- Exemple N°3:

Le joueur se trouve à la sortie du labyrinthe. Un message s'affiche pour le féliciter d'avoir gagné car ce jeu est très dur. Le jeu s'arrête.

GAGNER !

- Exemple N°4:

Le joueur est dans le labyrinthe. Il touche un sablier, 10s lui sont rajouter à son temps pour sortir du labyrinthe.

- Exemple N°5:

Le joueur est dans le labyrinthe. Il ne trouve pas la sortie car c'est un jeu très complexe. Il peut donc appuyez sur "Q" pour quitter le programme, on quitte le programme.

Contraintes du cahier des charges à respecter

Pour le joueur :

- Le joueur ne peut pas traverser les murs, il ne peut se déplacer qu'à l'intérieur des couloirs du labyrinthe
- Le joueur s'il est touché par un méchant perd instantanément une vie et revient au début
- Le joueur s'il touche un sablier, du temps (10s) est rajouté et le sablier disparaît
- Si le joueur ne possède plus de vie, on lui dit qu'il a perdu et le jeu s'arrête
- Si le joueur n'a plus de temps pour sortir, on lui dit qu'il a perdu et le jeu s'arrête

- Si le joueur atteint la sortie un message s'affiche pour dire qu'il a gagné et le jeu s'arrête

Pour un méchant :

- Le méchant ne peut pas traverser les murs, la sortie et les sabliers, il ne peut se déplacer qu'à l'intérieur des couloirs du labyrinthe
- Le méchant se déplace aléatoirement dans le labyrinthe, il doit prendre des choix de direction au hasard en vérifiant si la direction est possible

En appuyant sur la touche « Q », on doit arrêter le programme.

Le labyrinthe sera mis dans un fichier texte labyrinthe.txt :

- Les murs seront renseignés par des « - »
- Les couloirs seront renseignés par des espaces
- Le joueur au départ sera renseigné par un « J »
- Les méchants au départ seront renseignés par des « M »
- Les sabliers pour le temps supplémentaires seront renseignés par des « T »
- La sortie sera renseigné par un « S »

Les classes pour le jeu :

- **Classe et algorithme de la classe Labyrinthe :**
  La classe permet de générer le labyrinthe : murs, couloirs, sabliers, porte de sortie
  Elle gère les collisions avec un de ses sabliers et détecte le collision avec la sortie

Classe Labyrinthe
Attributs : **privée** fenetre : objet objet de la fenêtre pygame **privée** nomFichierImgMur : chaînes de caractères nom du fichier de l'image pour le mur **privée** nomFichierImgCouloir : chaîne de caractères nom du fichier de l'image pour le couloir **privée** nomFichierImgSablier : chaîne de caractères nom du fichier de l'image pour le sablier **privée** nomFichierImgSortie : chaîne de caractères nom du fichier de l'image pour la sortie **publique** positionsBlocMurs : liste de positions pour les surfaces des murs

publique positionsSabliers : liste de positions pour les surfaces des sabliers

publique positionSortie : position pour la surface de la sortie

Méthodes :

publique constructeur (fenetre, nomFichierImgMur, nomFichierImgCouloir,
nomFichierImgSablier, nomFichierImgSortie)
fenetre ← fenetre
nomFichierImgMur ← nomFichierImgMur
nomFichierImgCouloir ← nomFichierImgSablier
nomFichierImgSortie ← nomFichierImgSortie
nomFichierImgSablier ← nomFichierImgSablier
positionsBlocMurs ← []
positionsSabliers ← []

procédure publique genererLabyrinthe(tableauLabyrinthe)
posY ← 0
Pour ligne **de** tableauLabyrinthe
posX ← 0
Pour colonne **de** ligne
Si colonne = «COULOIR » **Alors**
couloir ← chargerImage(nomFichierImgCouloir)
mettreImageSurFenetreAPosition(fenetre, couloir, (posX, posY))
Sinon si colonne = «MUR» **Alors**
mur ← chargerImage(nomFichierImgCouloir)
positionMur ←determinerPositionSurface(mur)
positionMur.positionGauche ← posX
positionMur.positionHaut ← posY
mettreImageSurFenetreAPosition(fenetre, mur, positionMur)
Ajouter positionMur dans la liste positionsBlocMurs
Sinon si colonne = «SORTIE» **Alors**
sortie ← chargerImage(nomFichierImgSortie)
positionSortie ←determinerPositionSurface(sortie)
positionSortie.positionGauche ← posX
positionSortie.positionHaut ← posY
mettreImageSurFenetreAPosition(fenetre, sortie, positionSortie)
Sinon si colonne = «SABLIER» **Alors**
sablier ← chargerImage(nomFichierImgSablier)
positionSablier ←determinerPositionSurface(sortie)
positionSablier.positionGauche ← posX
positionSablier.positionHaut ← posY
mettreImageSurFenetreAPosition(fenetre, sablier, positionSablier)
Ajouter positionSablier dans la liste positionsSabliers
Fin Si

posX ← posX + 32
Fin Pour
posY ← posY + 32

```
        Fin Pour

    fonction publique detecterCollisionSortie(ciblePosition) → boolean
        Retourner siCollisionEntre(positionSortie, ciblePosition)

    fonction publique detecterCollisionSablier(ciblePosition) → boolean
        collisionSablier ← False
        indiceSablier ← 0

        Tant que collisionSablier = False Et indiceSablier < Taille(positionsSabliers)
            Si siCollisionEntre(positionsSabliers[indiceSablier], ciblePosition) Alors
                collisionSablier ← True
            Sinon
                indiceSablier ← indiceSablier + 1
            Fin si
        Fin Tant que

        Si collisionSablier = True Alors
            couloir ← chargerImage(nomFichierImgCouloir)
            mettreImageSurFenetreAPosition(fenetre, couloir, positionsSabliers[indiceSablier])
            Supprimer l'élément à l'indice indiceSablier dans la liste positionsSabliers

        Retourner collisionSablier
```

- **Classe et algorithme de la classe Mechant :**

  La classe permet de gérer un méchant dans le labyrinthe
  Elle permet d'afficher le méchant, de gérer le déplacement aléatoire du méchant en connaissant les obstacles du labyrinthe et de détecter une collision avec ce méchant

Classe Mechant
Attributs : **privée** fenetre : objet objet de la fenêtre pygame **privée** imageMechant: objet pour l'image du méchant **privée** positionMechant : position de la surface du méchant **privée** directionMechant : caractère « B » : Direction vers le bas « H » : Direction vers le haut « D » : Direction vers la droite « G » : Direction vers la gauche
Méthodes :

```
publique constructeur (fenetre, nomFichierImgMechant, colonne, ligne)
    fenetre ← fenetre
    imageMechant ←  chargerImage(nomFichierImgMechant)
    positionMechant←determinerPositionSurface(imageMechant)
    positionMechant.positionGauche ← colonne * 32
    positionMechant.positionHaut ← ligne * 32
    mettreImageSurFenetreAPosition(fenetre, imageMechant, positionMechant)
    directionMechant ← «B»

 fonction privée collisionObstacle(tableauPositionsObstacles, positionMechant) → boolean
    collisionObstacle ← False
    indiceObstacle ← 0

    Tant que collisionObstacle = False Et indiceObstacle < Taille(tableauPositionsObstacles)
       Si siCollisionEntre(tableauPositionsObstacles[indiceObstacle], positionMechant) Alors
          collisionObstacle ← True
       Sinon
          indiceObstacle ← indiceObstacle + 1
       Fin si
    Fin Tant que
    Retourner  collisionObstacle

 fonction privée sensInverseDirection(direction) → Caractère
        Si direction  = «D»  Alors
          Retourner «G»
        Sinon si direction  = «G»  Alors
          Retourner «D»

        Sinon si direction  = «H»  Alors
          Retourner «B»
        Sinon si direction  = «B»  Alors
          Retourner «H»

procédure publique gererDeplacement (tableauPositionsObstacles)
    listeDirectionsPossibleMechant ←[]

    positionDeplacementDroite ←positionPourDeplacementDroite(positionMechant, 1px)
    Si (directionMechant <> «G» Et collisionObstacle(tableauPositionsObstacles,
                                                     positionDeplacementDroite) = False Alors
        Ajouter «D» dans la liste listeDirectionsPossibleMechant
    Fin Si

    positionDeplacementGauche←positionPourDeplacementGauche(positionMechant, 1px)
    Si (directionMechant <> «D» Et collisionObstacle(tableauPositionsObstacles,
                                                     positionDeplacementGauche) = False Alors
        Ajouter «G» dans la liste listeDirectionsPossibleMechant
    Fin Si

    positionDeplacementHaut←positionPourDeplacementHaut(positionMechant, 1px)
```

```
        Si (directionMechant <> «B» Et collisionObstacle(tableauPositionsObstacles,
                                                positionDeplacementHaut) = False Alors
            Ajouter «H» dans la liste listeDirectionsPossibleMechant
        Fin Si

        positionDeplacementBas←positionPourDeplacementBas(positionMechant, 1px)
        Si (directionMechant <> «H» Et collisionObstacle(tableauPositionsObstacles,
                                                positionDeplacementBas) = False Alors
            Ajouter «B» dans la liste listeDirectionsPossibleMechant
        Fin Si

        Si taille(listeDirectionsPossibleMechant) = 0 Alors
          directionMechant ← sensInverseDirection(directionMechant)
        Sinon
          directionMechant ←choisirAleatoirementDansListe(directionMechant)
        Fin Si

         Si directionMechant = «D»  Alors
            positionMechant = positionPourDeplacementDroite(positionMechant, 1px)
         Sinon si directionMechant = «G»  Alors
            positionMechant = positionPourDeplacementGauche(positionMechant, 1px)
         Sinon si directionMechant = «H»  Alors
            positionMechant = positionPourDeplacementHaut(positionMechant, 1px)
         Sinon si directionMechant = «B»  Alors
            positionMechant = positionPourDeplacementBas(positionMechant, 1px)
         Fin si

        mettreImageSurFenetreAPosition(fenetre, imageMechant, positionMechant)

    fonction publique detecterCollision(ciblePosition) → boolean
        Retourner siCollisionEntre(positionMechant,  ciblePosition)
```

- **Classe et algorithme de la classe Joueur :**

  La classe permet de gérer le joueur dans le labyrinthe

  Elle permet d'afficher le joueur, de gérer son déplacement en connaissant les obstacles

Classe Joueur
Attributs : **privée** fenetre : objet objet de la fenêtre pygame **privée** imageJoueur: objet pour l'image du joueur **privée** imageJoueurVide: objet pour l'image vide pour effacer le joueur

privée positionDepart: position de la surface pour le départ du joueur

publique positionJoueur: position de la surface le joueur

Méthodes :

publique constructeur (fenetre, nomFichierImgJoueur, nomFichierImgJoueurVide, colonneJoueur, ligneJoueur)
fenetre ← fenetre
imageJoueur← chargerImage(nomFichierImgJoueur)
imageJoueurVide ← chargerImage(nomFichierImgJoueurVide)
positionDepart←determinerPositionSurface(imageJoueur)
positionDepart.positionGauche ← colonneJoueur * 32
positionDepart.positionHaut ← ligneJoueur * 32

procédure publique afficherDepartJoueur ()
mettreImageSurFenetreAPosition(fenetre, imageJoueurVide, positionJoueur)
positionJoueur ← positionDepart
mettreImageSurFenetreAPosition(fenetre, imageJoueur, positionJoueur)

procédure publique deplacerHaut(tableauPositionsObstacles)
nouvellePosition← positionPourDeplacementHaut(positionJoueur, 2px)
deplacerSiPossible(nouvellePosition, tableauPositionsObstacles)

procédure publique deplacerHaut(tableauPositionsObstacles)
nouvellePosition← positionPourDeplacementBas(positionJoueur, 2px)
deplacerSiPossible(nouvellePosition, tableauPositionsObstacles)

procédure publique deplacerDroite(tableauPositionsObstacles)
nouvellePosition← positionPourDeplacementDroite(positionJoueur, 2px)
deplacerSiPossible(nouvellePosition, tableauPositionsObstacles)

procédure publique deplacerGauche(tableauPositionsObstacles)
nouvellePosition← positionPourDeplacementGauche(positionJoueur, 2px)
deplacerSiPossible(nouvellePosition, tableauPositionsObstacles)

procédure privée deplacerSiPossible (nouvellePositionJoueur, tableauPositionsObstacles)
collisionObstacle ← False
indiceObstacle ← 0

Si siCollisionEntre(tableauPositionsObstacles[indiceObstacle], nouvellePositionJoueur) **Alors**
collisionObstacle ← True
Sinon
indiceObstacle ← indiceObstacle + 1
Fin si

Si collisionObstacle == False **Alors**
mettreImageSurFenetreAPosition(fenetre, imageJoueurVide, positionJoueur)
positionJoueur ← nouvellePositionJoueur

```
            mettreImageSurFenetreAPosition(fenetre, imageJoueur, positionJoueur)
        Fin si
```

- **Algorithme du programme :**

Le programme permet de créer une fenêtre

Lire le fichier texte labyrinthe.txt» et générer ensuite le labyrinthe (murs, couloirs, sabliers et porte)

Mettre le joueur dans le labyrinthe et pouvoir déplacer le joueur

Mettre les méchants dans le labyrinthe et déplacer les méchants

Gérer les collisions (porte, sabliers, méchants)

Début programme

fenetre ← CréationFenetreTailleAvecTitre(1100 * 670, « LE LABYRINTHE !!! »)

tableauLabyrinthe ←[]

listeMechants ← []

nombreVie ← 3

labyrinthe ← Labyrinthe(fenetre,"mur.png", "caseblanche.png", "sablier.png", "sortie.png")

debutProgrammeEnMs ← RecupererEnMsTempsDebutProgramme()

fichierLabyrinthe ← ouvrirFichier(«labyrinthe.txt», «r»)

indiceLigne ←0

Pour ligne **de** fichierLabyrinthe

ligneLabyrinthe ← [];

indiceColonne ← 0

Pour colonne **de** ligne:

Si colonne = «-» **Alors**

Ajouter «MUR» **dans la liste** ligneLabyrinthe

Sinon si colonne = «S» **Alors**

Ajouter «SORTIE» **dans la liste** ligneLabyrinthe

Sinon si colonne = «T» **Alors**

Ajouter «SABLIER» **dans la liste** ligneLabyrinthe

Sinon

Ajouter «COULOIR» **dans la liste** ligneLabyrinthe

Fin si

```
            Si colonne = «M» Alors
                mechant = Mechant(fenetre, "mechant.png", indiceLigne, indiceColonne)
                Ajouter mechant dans la liste listeMechants
            Sinon si colonne = «J» Alors
                joueur = Joueur(fenetre, "joueur.png", indiceLigne, indiceColonne)
            Fin si
            indiceColonne ← indiceColonne + 1
        Fin Pour
        Ajouter ligneLabyrinthe dans la liste tableauLabyrinthe
        indiceLigne ← indiceLigne + 1
    Fin Pour
Fermeture fichier  fichierLabyrinthe
labyrinthe. genererLabyrinthe(tableauLabyrinthe)
joueur. afficherDepartJoueur() ;
partieTermine ← False
partieQuitter ← False
tempsMaximum ← 60
Tant que partieTermine = False
    AfficherTexteAPosition(« LabyLu » , 747, 100)
    secondesRestantes ← tempsMaximum - (RecupererEnMsTempsDebutProgramme() -  start
    debutProgrammeEnMs) / 1000)
    Si secondesRestantes = 0 Alors
        AfficherTexteAPosition(« Perdu !!! » , 747, 100)
        partieTermine ← True
    Fin Si
    keys  ← touchePresse()
    Si  touchePresse = HAUT Alors
        joueur.deplacerGauche(labyrinthe.positionsBlocMurs)
    Sinon si touchePresse = DROITE Alors
        joueur.deplacerDroite(labyrinthe.positionsBlocMurs )
```

```
Sinon si touchePresse = HAUT Alors
   joueur.deplacerHaut(labyrinthe.positionsBlocMurs )
Sinon si touchePresse = BAS Alors
   joueur.deplacerBas(labyrinthe.positionsBlocMurs )
Sinon si touchePresse = Q Alors
    partieTermine ← True
    partieQuitter ← True
Fin si
Si labyrinthe.detecterCollisionSortie(joueur.positionJoueur) = True Alors
     AfficherTexteAPosition(« Gagné bravo !!! » , 750, 500)
     partieTermine ← True
 Fin si
 Si labyrinthe.detecterCollisionSablier(joueur.positionJoueur) = True Alors
    tempsMaximum ← tempsMaximum + 10
 Fin si
Pour mechant de listeMechants
    mechant.gererDeplacement(labyrinthe.positionsBlocMurs + labyrinthe.positionsSabliers +
                    [labyrinthe.positionSortie])
     collisionAvecMechant ← mechant.detecterCollision(joueur.positionJoueur)
     Si collisionAvecMechant = True Alors
       nombreVie ← nombreVie – 1
       Si nombreVie = 0 Alors
          AfficherTexteAPosition(« Perdu !!! » , 750, 500)
          partieTermine ← True
       Sinon
           joueur.afficherDepartJoueur()
       Fin Si
Fin Pour
AfficherTexteAPosition(« Vie : » + nombreVie , 750, 300)
AfficherTexteAPosition(« Reste: » + secondesRestantes + « s », 750, 300)
```

```
        RaffraichirEcran()
    Fin Tant que
    Si partieQuitter = True Alors
        QuitterProgramme()
    Fin Si
Fin programme
```