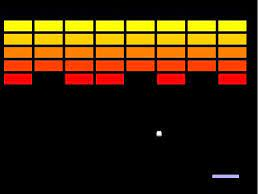
**Conception d’un jeu casse brique utilisant la programmation objet**



Sommaire :

1) Introduction

2) Principe du jeu

3) Analyse descendante

a) Analyse de niveau 0

b) Analyse de niveau 1

c) Analyse de niveau 2

4) Analyse objet

5) Références

6) Programme et documentation

7) Conclusion

1. Introduction

Nous voulons coder un jeu du style “casse-briques” et son interface en C++. Nous devons être capable de manipuler la raquette à partir du clavier (touches ← et →), la disparition des briques ainsi que de gérer les conditions de victoire et de défaite.

1. Principe du jeu

Le casse-briques est un jeu vidéo d’arcade apparu en 1975 avec le jeu Breakout. Il est directement inspiré d’un autre jeu d’arcade, Pong.

Le principe général est de détruire un ensemble de briques se trouvant dans un niveau pour accéder au niveau suivant grâce à une (ou plusieurs) balles.

Traditionnellement, le joueur contrôle une raquette pouvant seulement être déplacée horizontalement au bas de l'écran, et le but est d'empêcher la balle de toucher la bordure du bas de l’écran en faisant rebondir la balle sur la raquette.

Si la balle touche la raquette, alors elle est renvoyée en direction des briques.

Sinon, le joueur perd la balle (et la partie si c’était la dernière qui lui restait).

Dans des versions plus élaborées de ce jeu, on peut trouver l’ajout de bonus ou malus, une vitesse de balle qui augmente constamment ou encore plusieurs balles.

1. Analyse descendante

Analyse de niveau 0 :

Logigramme :

Initialisation

Déroulement de jeu

Affichage

Fin

Début

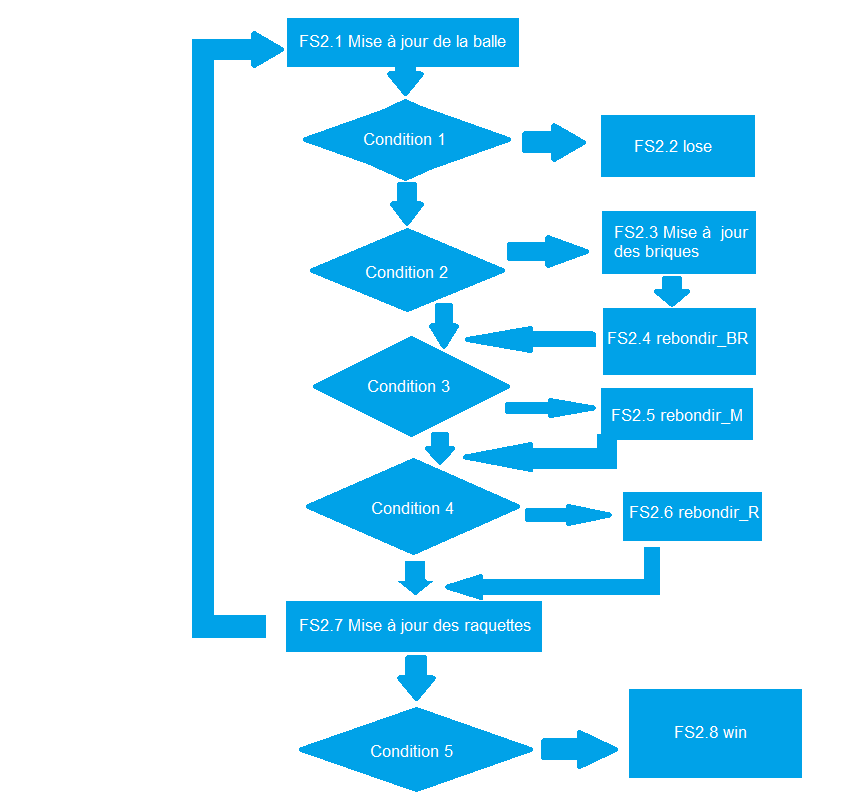
FP.1 Initialisation : Initialisation d’une grille contenant la balle, raquette, et les briques.

FP.2 Déroulement de jeu : Regroupe toutes les fonctions qui constituant le jeu.

FP.3 Affichage : Permet de mettre à jour notre grille avec ses éléments

Analyse de niveau 1 :

Logigramme :



Condition 1 : Si la balle touche le bas de l’écran

Pseudo-code : Si Pos\_B\_Y <= Ymin

Condition 2 : Si la balle touche une brique

Pseudo-code : Si (0 < Pos\_B\_X - Pos\_BR\_X < larg\_BR) ET (0 < Pos\_B\_Y- Pos\_BR\_Y < haut\_BR)

Condition 3 : Si la balle touche un mur

Pseudo-code : Si Pos\_B\_Y == Ymax OU Pos\_B\_X == Xmin OU Pos\_B\_X == Xmax

Condition 4 : Si la balle touche une raquette

Pseudo-code : Si (0 < Pos\_B\_X - Pos\_R\_X < larg\_R) ET (0 < Pos\_B\_Y == Pos\_BR\_Y)

Condition 5 : Si le nombre de brique est nul

Pseudo-code: Si (Nb\_BR == 0)

Analyse de niveau 2 :

Pour FP1 Initialisation

FS.1.1 Initialisation :

|  |  |
| --- | --- |
| Variable  Lose=False  Win=False  Xmin  Xmax  Ymin  Ymax  Pos\_B\_X  Pos\_B\_Y  Pos\_R\_X  Pos\_R\_Y  Larg\_R  Pos\_BR\_X [vecteur]  Pos\_BR\_Y [vecteur]  Visible [vecteur]  Larg\_BR  Haut\_BR  Nb\_BR  Dir =1 // 1 = nord-est  // 2 = nord-ouest  // 3 = sud-est  // 4 = sud-ouest | Description  État du jeu (défaite)  État du jeu (victoire)  Coordonnée X minimale de la fenêtre  Coordonnée X maximale de la fenêtre  Coordonnée Y minimale de la fenêtre  Coordonnée Y maximale de la fenêtre  Coordonnée X de la balle  Coordonnée Y de la balle  Coordonnée X de la raquette  Coordonnée Y de la raquette  Largeur de la raquette  Coordonnée X d’une brique  Coordonnée Y d’une brique  Condition de visibilité d’une brique (bool)  Largeur d’une brique  Hauteur d’une brique  Nombre de brique visible  Direction que prend la balle en fonction de son état : 1,2,3 ou 4. |

Pour FP2 déroulement de jeu :

FS.2.1 mises à jour de la balle

FS.2.2 lose

FS.2.3 mises à jour des briques

FS.2.4 rebondir\_BR

FS.2.5 rebondir\_M

FS.2.6 rebondir\_R

FS.2.7 mises à jour de la raquette

FS.2.8 win

FS.2.1 mises à jour de la balle

Input [Pos\_B\_X , Pos\_B\_Y, dir]

Output [Pos\_B\_X , Pos\_B\_Y]

Algo :

Si dir ==1

Pos\_B\_X = Pos\_B\_X +1

Pos\_B\_Y = Pos\_B\_Y +1

Finsi

Si dir ==2

Pos\_B\_X = Pos\_B\_X -1

Pos\_B\_Y = Pos\_B\_Y +1

Finsi

Si dir ==3

Pos\_B\_X = Pos\_B\_X +1

Pos\_B\_Y = Pos\_B\_Y -1

Finsi

Si dir ==4

Pos\_B\_X = Pos\_B\_X -1

Pos\_B\_Y = Pos\_B\_Y -1

Finsi

FS.2.2 lose

Input :[Pos\_B\_Y, lose]

Output : [lose]

Algo :

Si (Pos\_B\_Y<= Ymin)

Lose=1

Finsi

FS.2.3 mise à jour des briques

Input [Pos\_B\_X , Pos\_B\_Y, Pos\_BR\_X , Pos\_BR\_Y, Visible]

Output [Visible]

Algo :

Pour i dans briques

Si Visible[i]=true

Si (0<Pos\_B\_X- Pos\_BR\_X[i]<larg) et (0<Pos\_B\_Y- Pos\_BR\_Y[i]<haut)

Visible[i]=0

Rebondir\_BR(Pos\_B\_X, Pos\_B\_Y, Pos\_BR\_X, Pos\_BR\_y, dir)

Finsi

Finsi

Fin pour

FS.2.4 rebondir\_BR

Input :[Pos\_B\_X , Pos\_B\_Y, Pos\_BR\_X , Pos\_BR\_Y, dir]

Output : [dir]

Algo :

Si dir==1

Si Pos\_B\_X == Pos\_BR\_X

Dir=2

Finsi

Si Pos\_B\_Y == Pos\_BR\_Y

Dir=3

Finsi

Finsi

Si dir==2

Si Pos\_B\_X == Pos\_BR\_X +larg\_BR

Dir=1

Finsi

Si Pos\_B\_Y == Pos\_BR\_Y

Dir=4

Finsi

Finsi

Si dir==3

Si Pos\_B\_X == Pos\_BR\_X

Dir=4

Finsi

Si Pos\_B\_Y == Pos\_BR\_Y+haut

Dir=1

Finsi

Finsi

Si dir==4

Si Pos\_B\_X == Pos\_BR\_X +larg\_BR

Dir=3

Finsi

Si Pos\_B\_Y == Pos\_BR\_Y +haut\_BR

Dir=2

Finsi

Finsi

FS.2.5 rebondir\_M

Input :[Pos\_B\_X , Pos\_B\_Y, dir]

Output : [dir]

Algo :

Si dir==1

Si Pos\_B\_X == Xmax

Dir=2

Finsi

Si Pos\_B\_Y == Ymax

Dir=3

Finsi

Finsi

Si dir==2

Si Pos\_B\_X == Xmin

Dir=1

Finsi

Si Pos\_B\_Y == Ymax

Dir=4

Finsi

Finsi

Si dir==3

Si Pos\_B\_X == Xmax

Dir=4

Finsi

Finsi

Si dir==4

Si Pos\_B\_X == Xmin

Dir=3

Finsi

Finsi

FS.2.6 rebondir\_R

Input :[Pos\_B\_X , Pos\_B\_Y, Pos\_R\_X , Pos\_R\_Y,dir]

Output : [dir]

Algo :

Si (0<Pos\_B\_X- Pos\_R\_X<larg\_R) et (0<Pos\_B\_Y= Pos\_BR\_Y)

Si dir==3

Dir=1

Finsi

Si Dir=4

Dir=2

Finsi

Finsi

FS.2.7 mise à jour raquette

Input : [ Pos\_R\_X , Pos\_R\_Y]

Output : [Pos\_R\_X , Pos\_R\_Y]

Algo :

Si appui()==’d’

Pos\_R\_X = Pos\_R\_X +1

Si Pos\_R\_X >Xmax- larg\_R

Pos\_R\_X =Xmax-larg\_R

Finsi

Finsi

Si appui()==’g’

Pos\_R\_X = Pos\_R\_X -1

Si Pos\_R\_X >Xmin

Pos\_R\_X =Xmin

Finsi

Finsi

FS.2.8 win

Input :[Nb\_BR, win]

Output : [win]

Algo :

Si (Nb\_BR==0)

Win=1

Finsi

1. Analyse objet

**Quelle modélisation objet pouvons-nous envisager par rapport aux interactions à gérer dans un casse-briques (interactions entre les trois éléments que sont la raquette, la balle et le mur de briques).**

Nous pouvons envisager la modélisation objet suivante :

Classe Brique :

Attributs :

Pos\_BR\_X

Pos\_BR\_Y

Visible

Larg\_BR

Haut\_BR

Nb\_BR

Méthodes :

FS2.3 : Mise à jour brique

Classe Balles :

Attributs :

Pos\_B\_X

Pos\_B\_Y

Dir

Méthodes :

FS2.1 : Mis à jour balle

FS2.4 : Rebondir\_BR

FS2.5 : Rebondir\_M

FS2.6 : Rebondir\_R

Classe Raquette :

Attributs :

Pos\_R\_X

Pos\_R\_Y

Larg\_R

Méthodes :

FS2.7 : Mise à jour raquette

Classe Affichage :

Attributs :

Lose

Win

Xmin

Xmax

Ymin

Ymax

Méthodes :

FS2.2 : lose

FS2.8 : win

**Quel diagramme de classes suggérez-vous pour l’implémentation objet du jeu ? Vous ferez apparaître les relations de dérivation et de composition**

**Quelles interfaces publiques proposez-vous dans vos classes pour gérer les interactions raquette balle et balle-brique ?**

1. Références

https://fr.wikipedia.org/wiki/Diagramme\_de\_classes

1. Programme et documentation
2. Conclusion

Ce projet nous a permis de mettre en œuvre une analyse descendante pour la première fois et nous a également permis de travailler sur un programme de manière autonome, presque comme en situation concrète contrairement aux TP, très guidé. Comme nous avons fait une analyse descendante plus que complète, nous n’avons pas rencontré de difficultés particulières lors de l’implémentation du programme en C++.