LOG8715- Architecture de jeu vidéo

Rapport TP1 :

Conception ECS

Présenté à

Samuel Bellomo

&

Keven Chaussé

Réalisé par

Équipe 4

Emmanuel Doré-Walsh	1954113
Alex Hua	1994253
Vincent Beiglig	2161304

Dans l'architecture ECS qui a été construite, les différentes composantes sont des structures contenant le strict minimum de variables afin d'améliorer la maintenabilité. La composante Protection en possède deux, car celles-ci sont étroitement liées. Les composantes choisies, ainsi que les variables qui se trouvent dans leur structure peuvent être visualisées au diagramme 1.

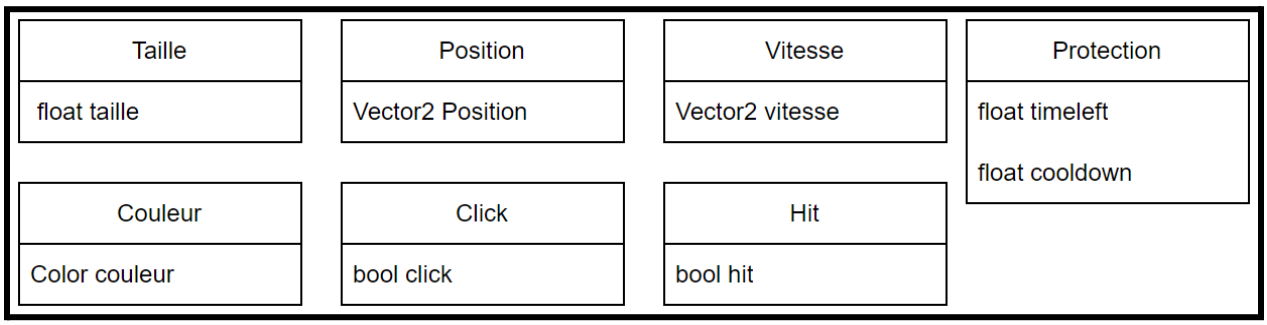

Diagramme 1 : Structure des composantes et leurs variables

Pour avoir un répertoire des composantes existantes, une classe publique statique englobe des conteneurs de type dictionnaire pour chaque composante où chaque clé représente une entité différente. De cette façon, on peut itérer sur les composantes souhaitées de manière efficace sans *cache miss*, car le dictionnaire complet de chaque composante est à un endroit dans la mémoire. Les différents dictionnaires peuvent être visualisés au diagramme 2.

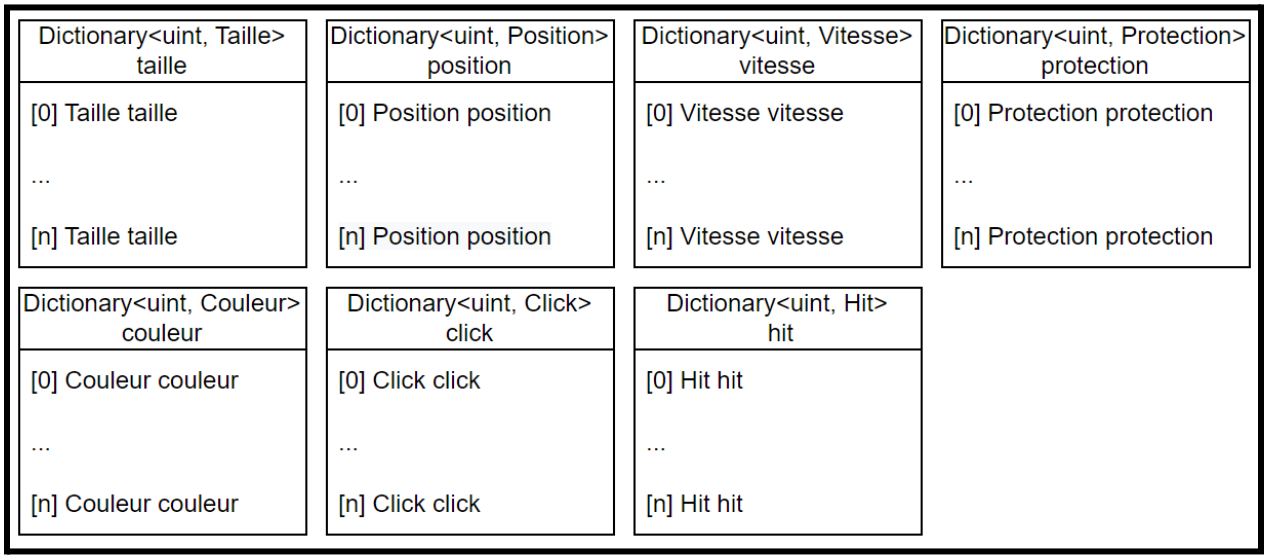

Diagramme 2 : Dictionnaires de composantes utilisés

Les comportements des cercles ont été séparés dans les systèmes suivants : StartUp(), Move(), Bounce(), WallBounce(), Explosion(), ClickCircle(), Destroy(), ColorSetter(), Protect(),

ReverseTime(). FasterPace(). Le choix des séparations a été fait pour réduire l'interdépendance entre les systèmes afin de réduire la complexité du code et améliorer la maintenabilité. Le système StartUp s'occupe de démarrer la simulation en suivant la logique décrite au diagramme 3.

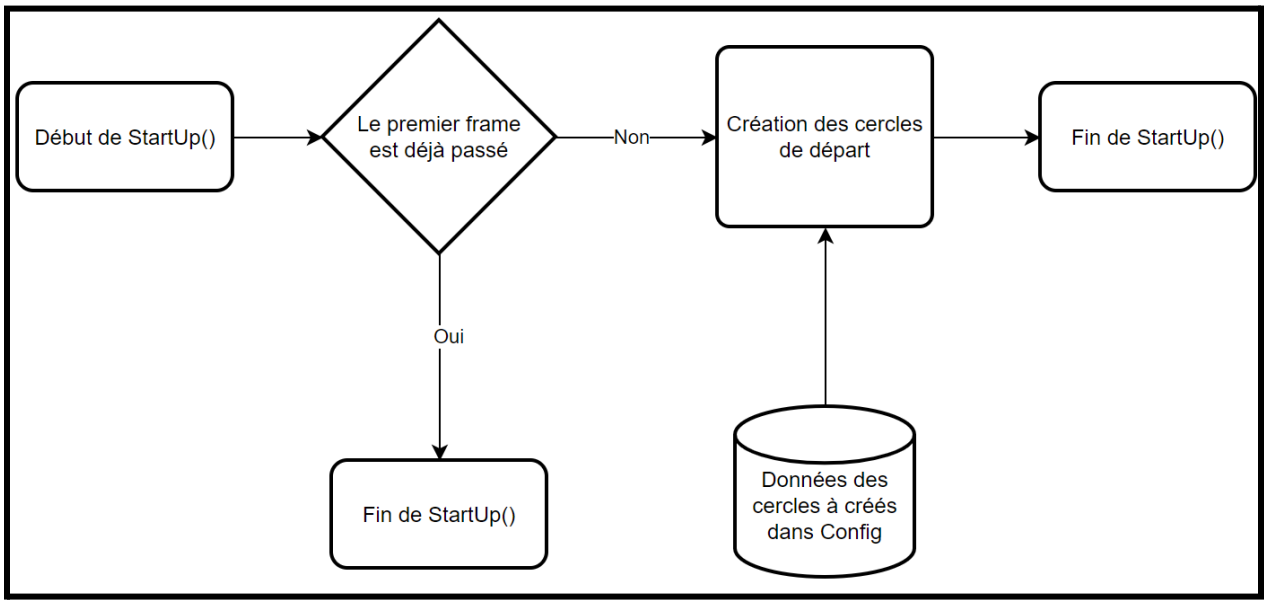

Diagramme 3 : Logique du système StartUp()

Le système Move() pour sa part s'occupe du déplacement des cercles en ajoutant à la position le temps écoulé depuis la dernière *frame* multiplié par la vitesse. Bounce() s'occupe de gérer les collisions entre cercles en fonction de l'état de ceux-ci tel que décrit au diagramme 4.

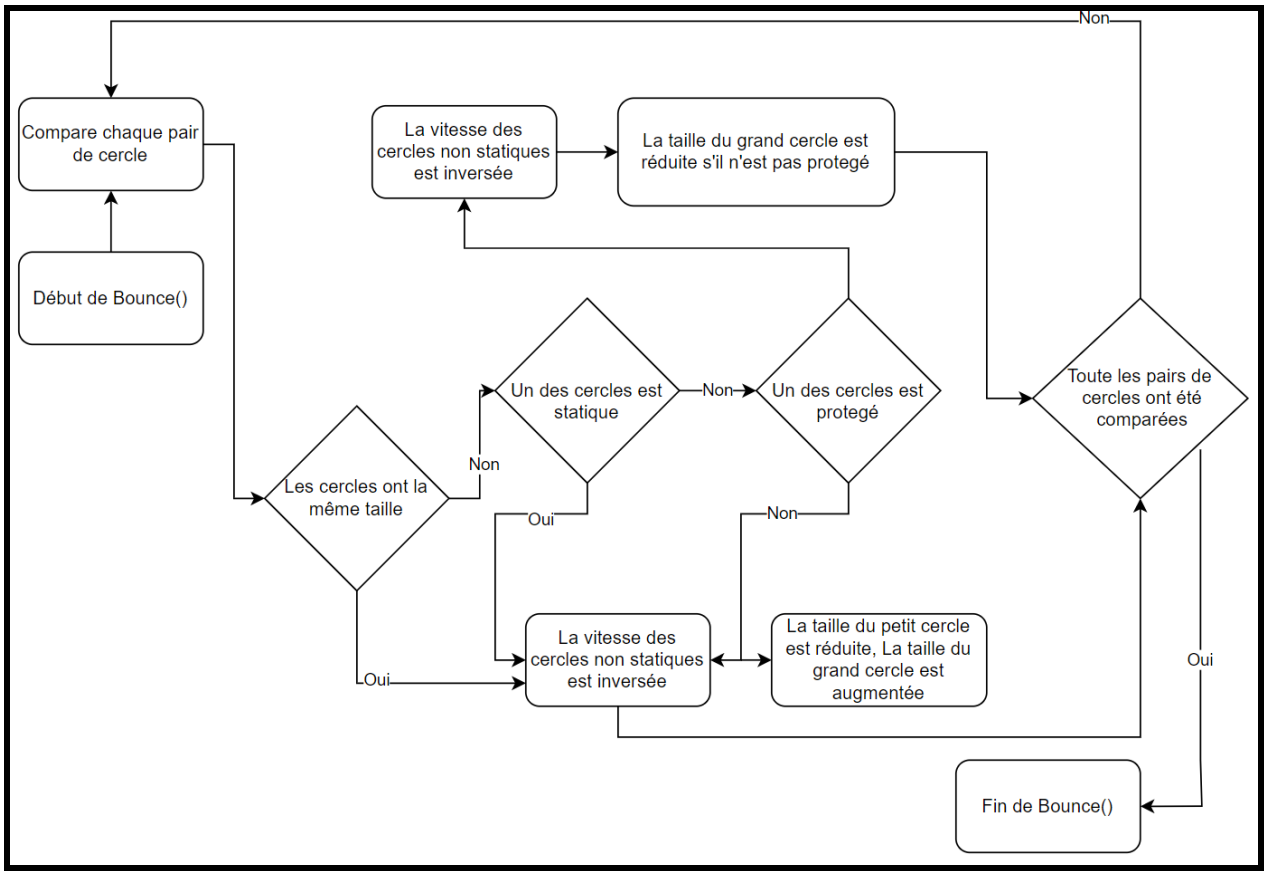

Diagramme 4 : Logique du système Bounce()

Ensuite, le système WallBounce() s'occupe de faire rebondir tous les cercles sur le contour de la scène en utilisant les dimensions de celles-ci. Explosion() s'occupe de faire exploser les cercles en deux cercles plus petits en suivant la logique du diagramme 5. Alors que ClickCircle() fait exploser les cercles dynamiques lorsqu'on clique sur ceux-ci ou les détruit s'il on une taille de 1.

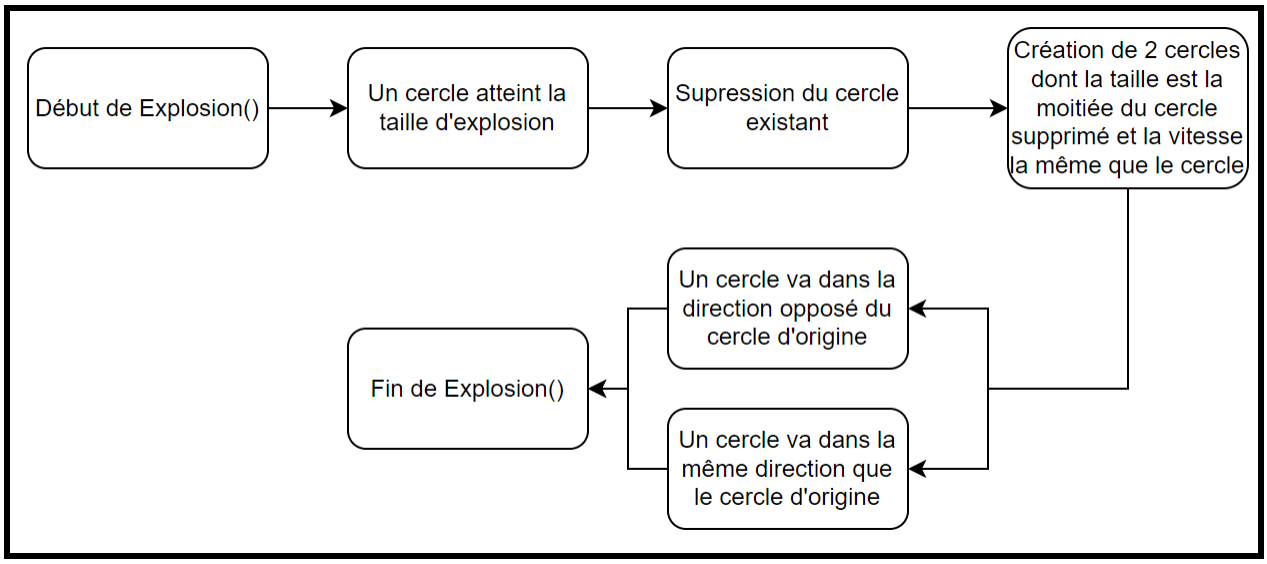

Diagramme 5 : Logique du système Explosion()

Destroy() s'occupe de détruire les cercles dont la taille est nulle. ColorSetter() s'occupe de donner une couleur aux cercles a la fin de chaque frame en suivant la logique décrite au diagramme 6.

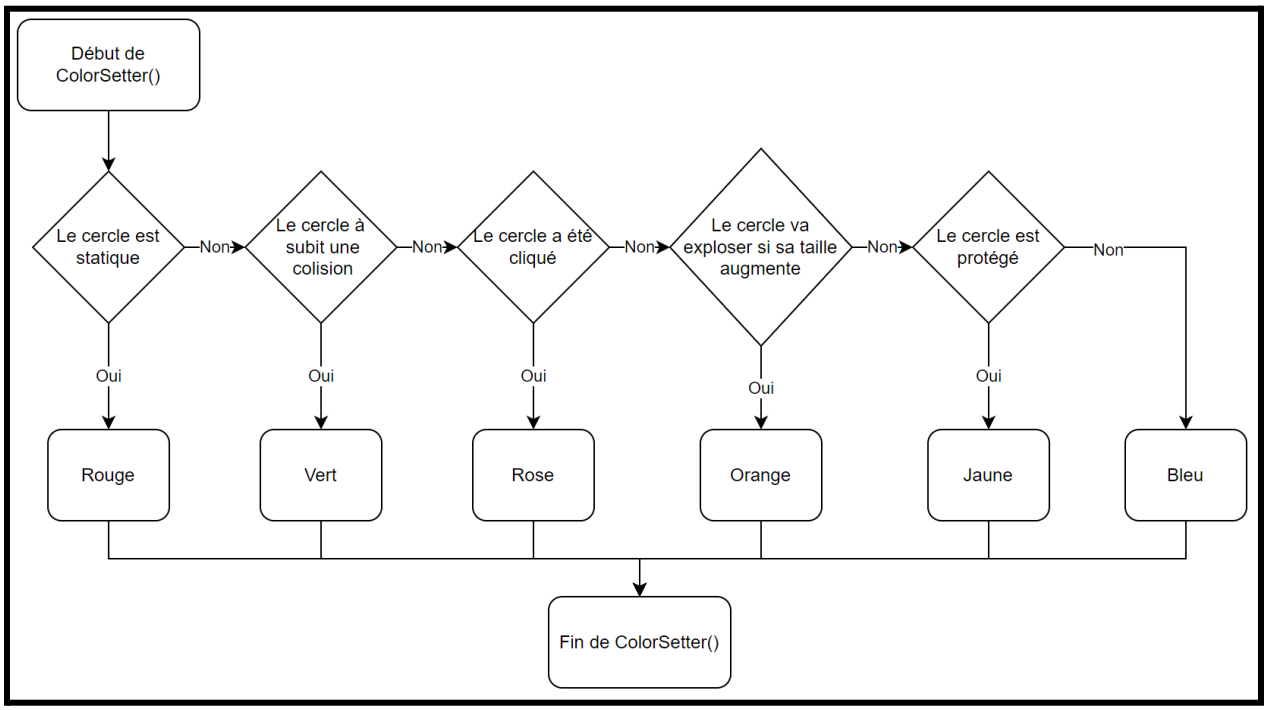

Diagramme 6 : Logique du système ColorSetter()

Protect() gère le temps de *cooldown* et le temps restant de protection, ainsi que l'attribution de la protection selon la logique du diagramme 7. Toutes les couleurs sont attribuées dans ce système pour éviter des problèmes avec l'ordre de priorité.

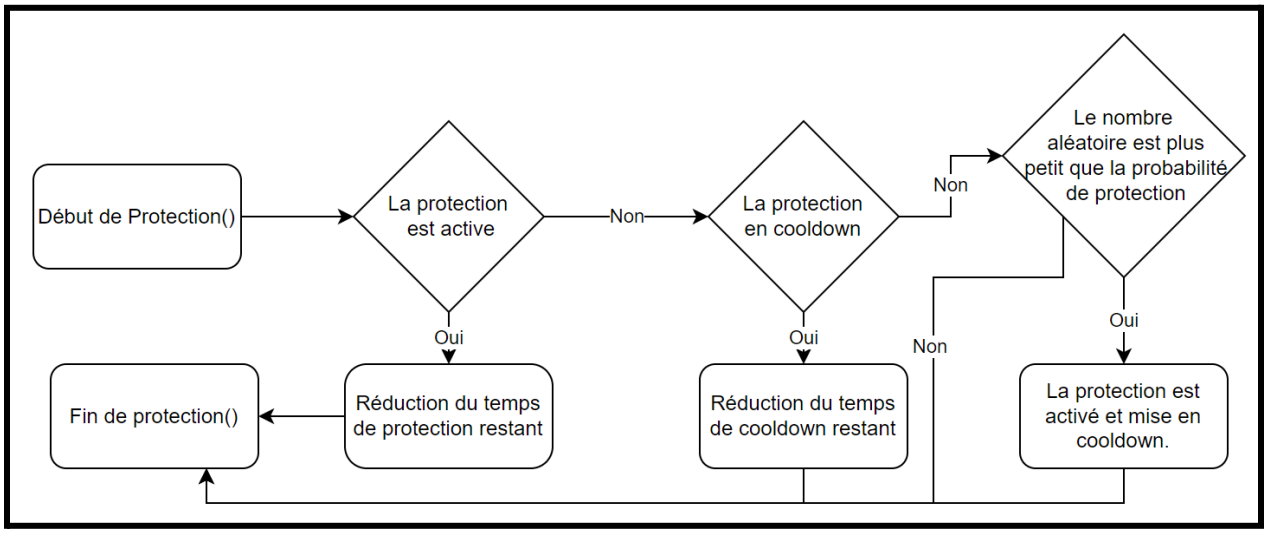

Diagramme 7 : Logique du système Protect()

Le système ReverseTime() s'occupe du retour en arrière de 3 secondes selon la logique du diagramme 8.

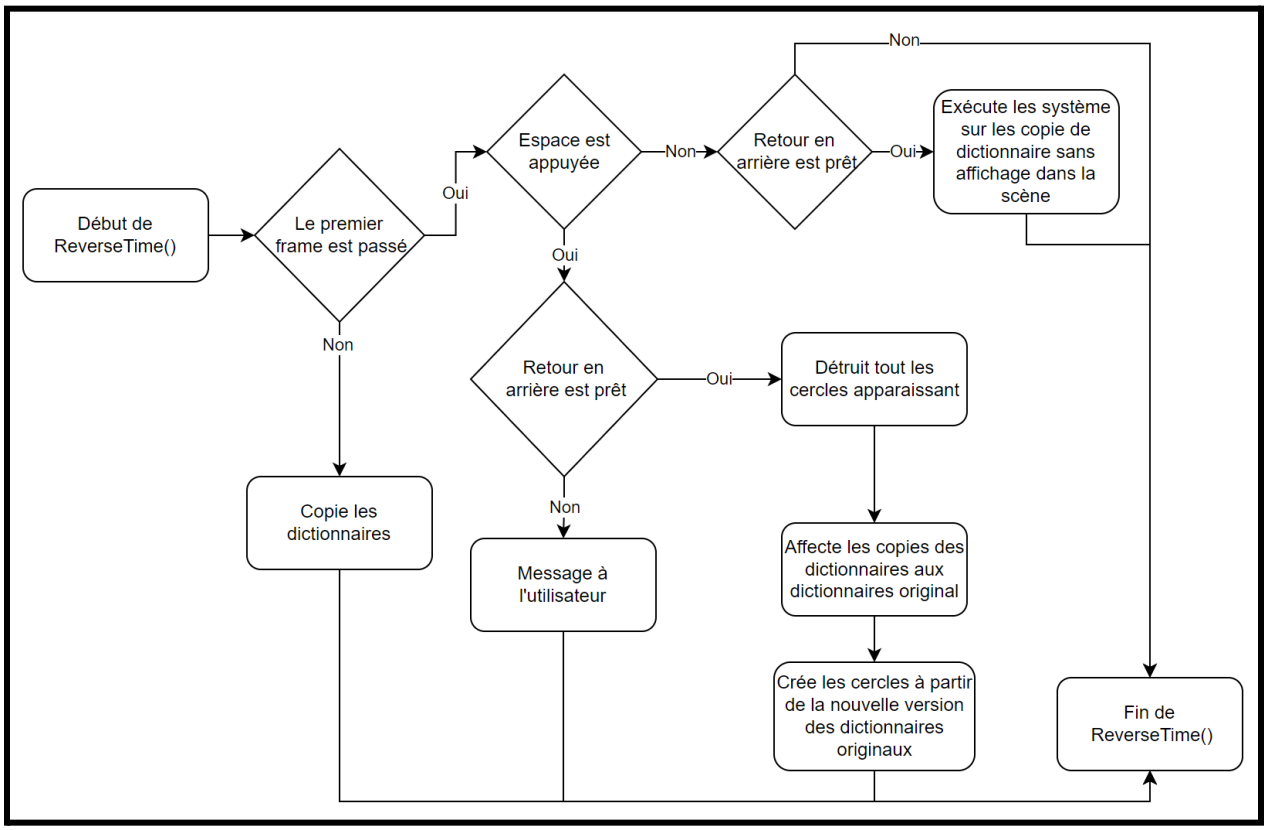

Diagramme 8 : Logique du système ReverseTime()

Finalement le Système FasterPace() quadruple le nombre de simulations effectuées pour les cercles se trouvant du côté gauche de la scène en appelant tous les autres systèmes après avoir modifié les dictionnaires existant pour qu'ils ne contiennent que les cercles d'intérêt. Ce système n'a malheureusement pas été implémenté, puisqu'on avait de la difficulté à donner un time scale à nos gameObject circle .