

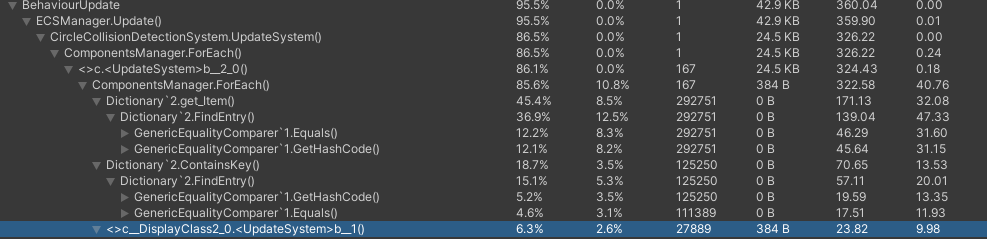
**LOG8715-TP2**

Quoc-Hao Tran (1972967)

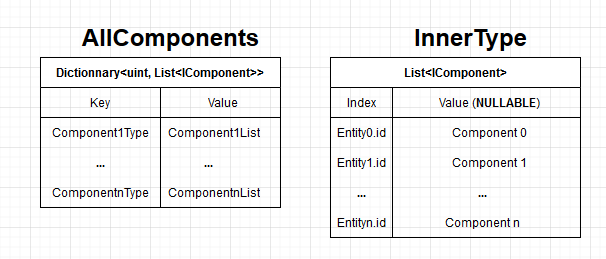
Jeffrey Lavallée (1847777)

1/ Analyse

Voici les résultats après une analyse profonde du code de base :



On remarque que le grand du coût vient du seul système qui utilise une double boucle, le *CircleCollisionDetectionSystem*. On remarque aussi que les coûts sont associés à la recupération des éléments dans les dictionnaires. Pour réduire l’utilisation, on a changé le code pour utiliser une liste comme *innertype* plûtot qu’un dictionnaire. Deux raisons principales ont motivées cette décision : La première est que une liste avec comme index le id d’une entité permet des accès aléatoires instannés. La deuxième raison est que comme les components sont des types simples, donc l’allocation se fait sur le *stack* et non pas sur le *heap*. Cela permet d’itérer sur les éléments de façon séquentielle. C’est un avantage compartivement au dictionnaire qui ne le permet pas. La raison est que lorsqu’on itère des éléments placés sur des addresse séquentielles, cela permet une utilisation optimale de la cache. Clairement cette approche à bien payée car on achève une performance d’environs 140 *fps* pour 950 cercles. De plus, il semblerait que de simple *for loop* augmente la performance par rapport aux *foreach*, donc tout les *foreach* ont été remplacer par des *for loop*. Voici le diagramme de la solution :



En ce qui concerne la maintenabilité, Cette approche est raisonablement maintenable puisqu’il serait facile d’ajouter un nouveau *component* ou un nouveau système sans changer le code du *componentManager*. Un problème possible est que la taille des listes ne peuvent jamais diminuer, elles ne font qu’augmenter puisqu’on ne supprime jamais d’éléments, on ne fait que les assigner à *null* dans la liste. Donc si dans le future on voudrait pouvoir ajouter et retirer des *entity* de façon aléatoire, ce serait un élément à considerer pour ne pas trop utiliser de mémoire.