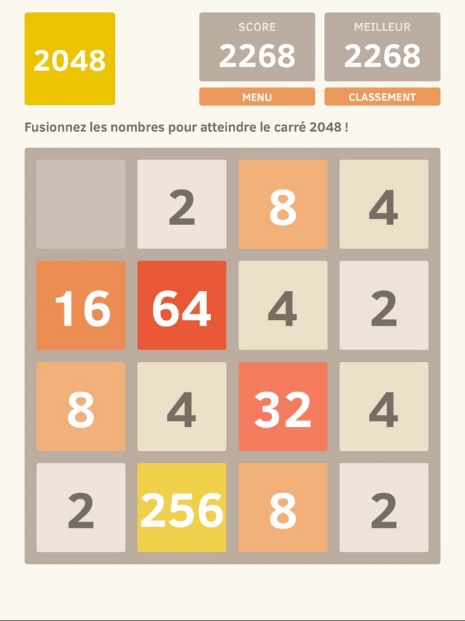
Introduction et cahier des charges fonctionnel.

Le jeu du 2048 se présente de la manière suivante :

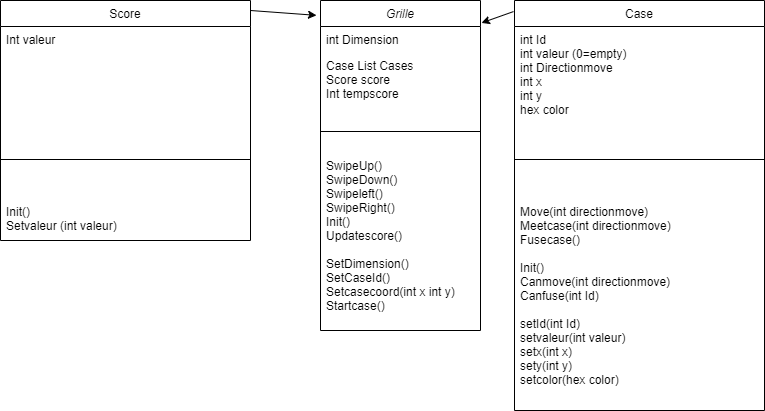


Un tableau de nombres comporte des cases remplies (ou non) par des nombres. Ces nombres sont des multiples de 2. Il est possible de faire grandir ces nombres en faisant glisser l’intégralité du tableau vers la droite, la gauche, le haut ou le bas. Lorsque deux cases se rencontrent pendant un glissement :

* Si elles contiennent le même nombre : elles fusionnent pour devenir une case de coordonnée variante selon la direction de glissement, de nombre égal à la multiplication du nombre des cases par deux.
* Si elles ne contiennent pas le même nombre : elles agissent comme des objets physiques, et sont en butée l’une sur l’autre.
* Si une case rencontre une bordure du tableau, alors elle ne bouge pas plus selon la direction de glissement (il faudra donc faire un test avant avec un self.canmove dans la direction du mouvement)

Par ailleurs, le joueur gagne lorsqu’une case atteint la valeur 2048. (on peut continuer après sur la véritable application). Un compteur de score tient également compte du score du joueur (le score est calculé de la manière suivante : pour chaque nouvelle fusion de cases, la valeur de la nouvelle case est ajoutée au score.)

On peut envisager de nouvelles fonctionnalités, mais à partir de ces fonctionnalités de base, on peut établir les classes et fonctions basiques qui vont permettre à la fois de créer les fonctionnalités du jeu, mais également les liens entre les objets et l’interface graphique du jeu (Qml).



Le diagramme de classes utilisé pour le projet est situé ci-dessus. Une explication sommaire de chaque fonction est présentée ci-dessous : Pour chaque classe, un constructeur et un destructeur sont implémentées.

Pour la classe grille :

* Un attribut dimension donne la taille de la grille, qui est une matrice.
* L’attribut Cases est une liste contenant des objets de type case.
* L’attribut score est un objet de type score, distinct pour faciliter l’affichage.
* L’attribut Tempscore est un score temporaire qui est ajouté au vrai score à la fin de chaque tour, et qui est donc remis à zéro à chaque tour.

Les fonctions Swipe correspondent au mouvement des cases après un input par le joueur. A la fin de chaque fonction swipe, qui fait appel aux fonctions de la classe case pour chaque case, et qui contient tout un tour dans chaque éxécution, on fait appel à la fonction UpdateScore et la fonction UpdateDisplay, en lien avec QML, pour mettre à jour l’affichage La fonction UpdateBoard vient avant ces deux fonctions pour mettre à jour la liste des cases et changer celles-ci dans la matrice-même. Nous discuteront de l’animation des cases plus tard si le temps nous le permet. Actuellement, il est uniquement prévu de mettre à jour les cases à la fin de chaque tour. Il faudra également penser à ajouter une fonction qui parcourt la liste des cases à la fin de chaque tour, détermine lesquelles sont vides, et en choisit une au hasard pour ajouter un 2 ou un 4, comme dans le vrai jeu.

Concernant les attributs et fonctions de la classe case :

* Chaque objet case possède une coordonnée X et une coordonnée Y, donnant sa place dans la matrice.
* L’Id correspond à l’attribut qui permettra de tester les cases entre elles. C’est l’identifiant qui permet de répertorier chaque case.
* La valeur de la case correspond à sa valeur dans le jeu. Il faudra penser à ajouter une fonction WIN qui détecte lorsque la valeur change en 2048, et change un attribut de la classe grille pour qu’une fonction victoire se déclenche.
* Directionmove est une valeur qui détermine dans quelle direction l’utilisateur a glissé, et servira à chaque fonction test de la partie mouvement de la case
* Color est la couleur de la case, qui varie uniquement en fonction de sa valeur. Toutes les cases de même valeur sont de même couleur.

Concernant les fonctions. Voici la boucle qui se met en place à chaque itération de la fonction swipe.

On parcourt la grille en commençant par le côté de la direction vers laquelle glisse l’utilisateur, comme dans le vrai 2048. Pas besoin de regarder si une case est collée contre le mur. En effet, si elle toutes les cases tout à droite (par exemple), de valeur nulle ou non, sont collées contre le mur, et leur valeur ne changera pas. On peut donc directement commencer par une case à une case de distance du mur ‘destination’. Ensuite, on regarde si elle rencontre une case sur son trajet. Si non, alors on supprime la case et on en créé une de même valeur collée au mur. Si oui, alors on regarde si elle peut fusionner avec la case en question (la fonction qui regarde si elle rencontre une case retourne l’ID de la case si elle en rencontre une, 0 sinon.). Si non, alors on la laisse collée à cette case en modifiant la valeur des coordonnées pour qu’elle soit à côté. Si oui, alors la fonction Fuse supprime les deux cases fusionnées, créé une nouvelle case de valeur double à l’endroit de la case rencontrée. On ne réitère pas le test sur cette nouvelle case. Une fois la nouvelle case créée, on déplace toutes les cases dans la direction inverse de la direction de mouvement vers la direction de mouvement pour faire le mouvement correspondant à la fusion de la case. Une fois que toutes les cases de la matrice sont testées, on sort de la fonction Swipe, et on peut tout mettre à jour.

On illustre ici un cas d’utilisation, avec toutes les étapes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **4** | **4** | **4** | **4** |
| **2** | **2** | **4** | **4** | **4** |
| **3** |  | **2** | **4** | **4** |
| **4** |  |  |  | **4** |
|  | **A** | **B** | **C** | **D** |

L’utilisateur fait glisser le tableau vers la droite.

On regarde si la case 1,C peut bouger. Oui. Si elle rencontre une case sur son trajet. Oui. Si elle peut fusionner avec. Oui. On supprime les cases 1,C et 1,D, et on créé une nouvelle case sur 1,D, de valeur 8. On déplace les cases 1,A, 1,B, vers la droite.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **0** | **4** | **4** | **8** |
| **2** | **2** | **4** | **4** | **4** |
| **3** | **0** | **2** | **4** | **4** |
| **4** | **2** | **0** | **0** | **4** |
|  | **A** | **B** | **C** | **D** |

Il ne sert à rien de re-tester la case 1,C . on teste la case 1,B, on voit qu’elle peut fusionner avec 1,C, elle fusionne.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **0** | **0** | **8** | **8** |
| **2** | **2** | **4** | **4** | **4** |
| **3** | **0** | **2** | **4** | **4** |
| **4** | **2** | **0** | **0** | **4** |
|  | **A** | **B** | **C** | **D** |

On passe ensuite à la ligne suivante, les cases 2,C , 2,D sont fusionnées, les case 2,A, 2,B sont déplacées vers la droite. On ne re-teste pas la case 2,C. on teste la case 2-B, elle ne peut pas fusionner. On obtient le tableau suivant.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **0** | **0** | **8** | **8** |
| **2** | **0** | **2** | **4** | **8** |
| **3** | **0** | **2** | **4** | **4** |
| **4** | **2** | **0** | **0** | **4** |
|  | **A** | **B** | **C** | **D** |

On fait de même pour les lignes suivantes, et on voit que le tableau obtenu à la fin est bien le bon tableau.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **0** | **0** | **8** | **8** |
| **2** | **0** | **2** | **4** | **8** |
| **3** | **0** | **0** | **2** | **8** |
| **4** | **0** | **0** | **2** | **4** |
|  | **A** | **B** | **C** | **D** |