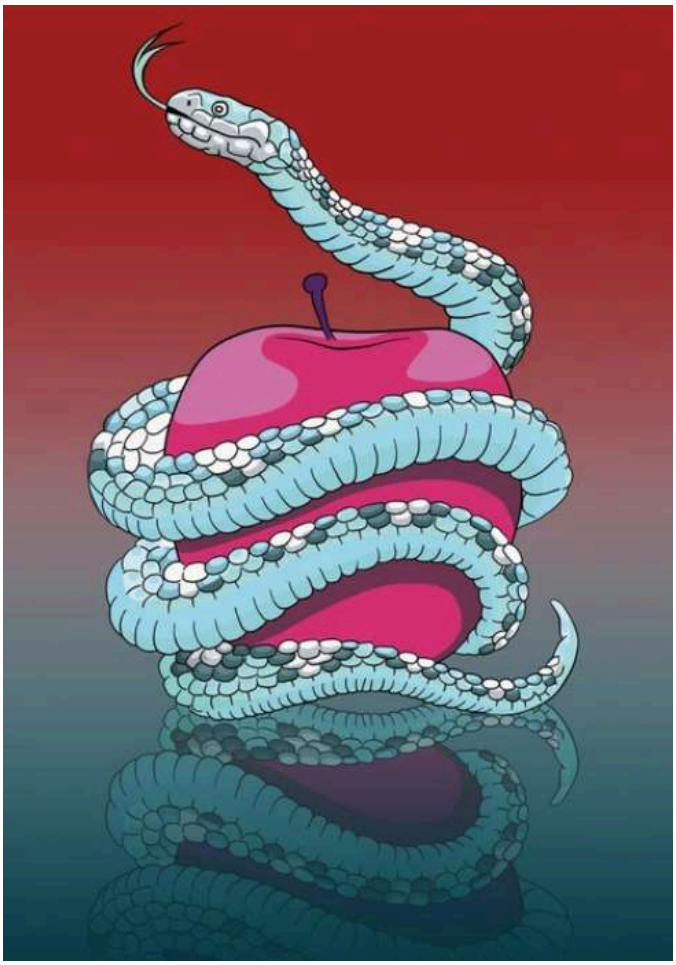
Favre Sylvain - BTS SIO 1 - 20 Janvier 2025

Codage C - Cpp

Livre Blanc : Jeux Basiques et Levels Of Snake



SOMMAIRE

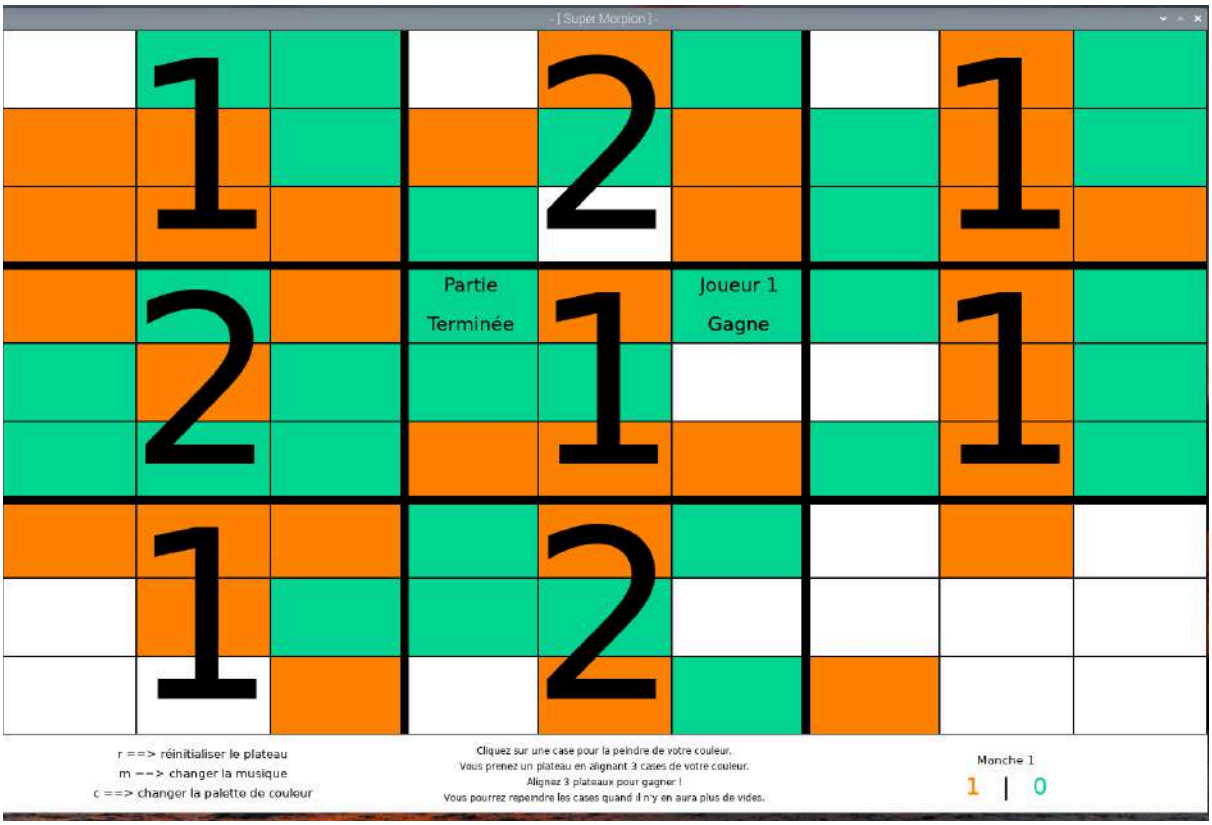
1 Présentation des différents jeux

1.1 Morpion

Un jeu de morpion basique, ou les joueurs doivent aligner trois cases de leur couleur respective. Plusieurs palettes de couleur sont disponibles, ainsi que plusieurs musiques, et la capacité de faire commencer une partie par un joueur différent a chaque fois. Recommencer le jeu avec n quand la partie n'est pas terminée compte pour un abandon et donne un point à l'adversaire.

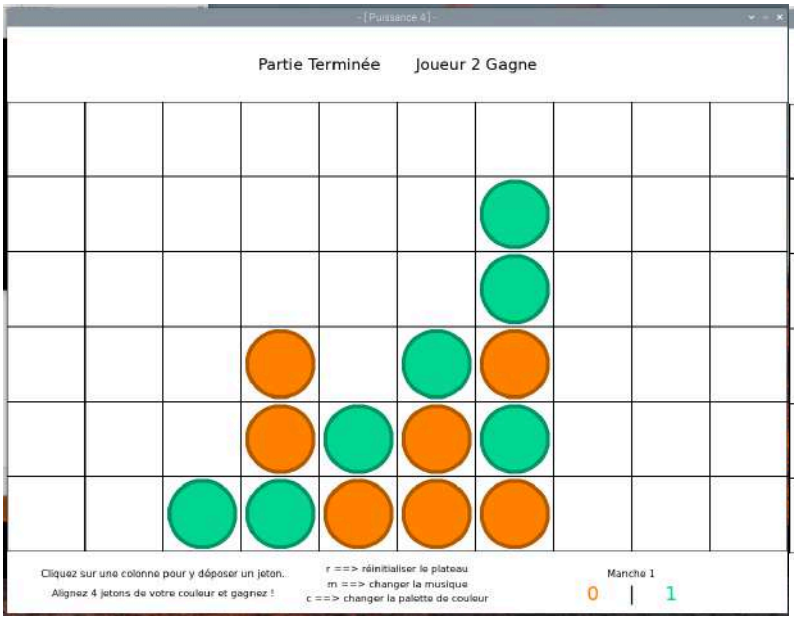
1.2 Super-Morpion

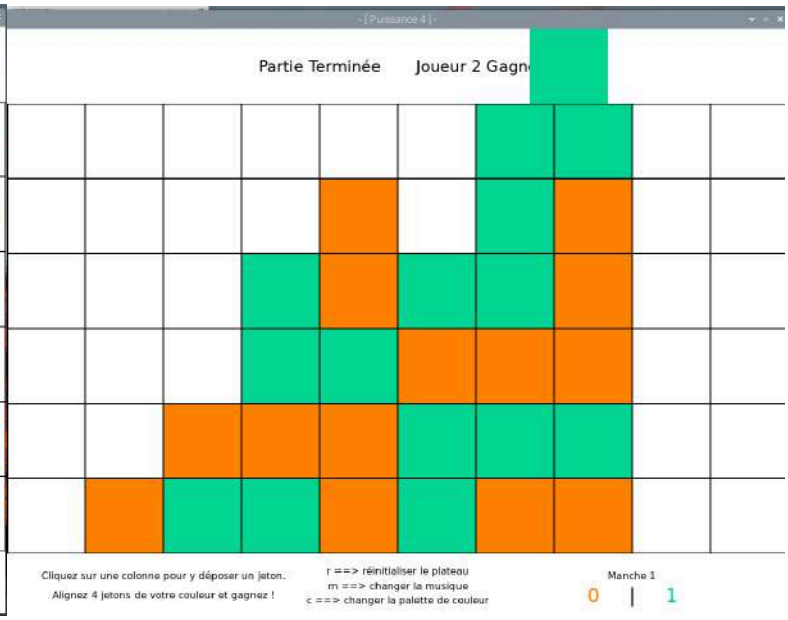
Un jeu de super-morpion adapté. Le grand plateau de jeu est composé de 9 plateaux normaux de 3 par 3. Lorsqu'un joueur aligne trois cases de sa couleur sur l'un des plateaux normaux, il le gagne. La partie se finit lorsqu'un joueur parvient à gagner 3 plateaux alignés. Basé sur le morpion de base, les mêmes palettes de couleur sont disponibles, dont une palette de couleur invisible qui ne fait apparaitre les jetons que lorsque la manche est terminée. Si toutes les cases sont possédées, les joueurs gagnent le droit de capturer n'importe quelle case. De cette manière, il ne peut pas y avoir d'égalité.



1.3 Puissance 4 (Rectangles / Jetons)

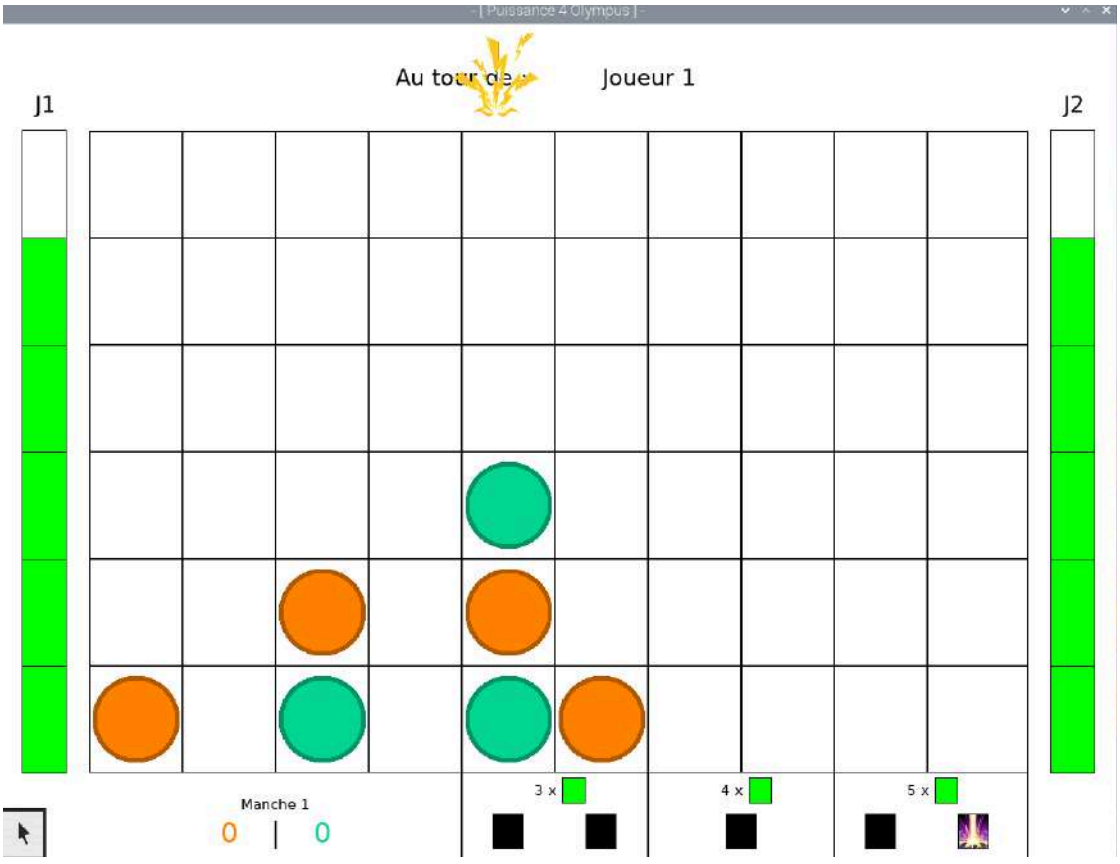
Un jeu de puissance 4 ou les pièces des joueurs sont des rectangles (dans la première version) et des véritables jetons (dans la deuxième version). Les joueurs posent à tour de rôle leur pièces en haut du plateau de jeu, et les pièces tombent vers le bas jusqu'à tomber sur le bas du plateau, ou sur une autre pièce. Le premier joueur qui réussit à aligner 4 pièces de sa couleur remporte la manche. Comme pour les autres jeux, plusieurs palettes de couleur sont disponibles. Il est impossible de poser un jeton dans une colonne déjà remplie. Enfin, un jeton apparaît au-dessus de la souris pour montrer dans quelle colonne il viendrait se mettre si le joueur venait à cliquer. Selon le style d'affichage, le jeton suit complètement la souris, ou se cantonne aux colonnes du plateau de jeu.





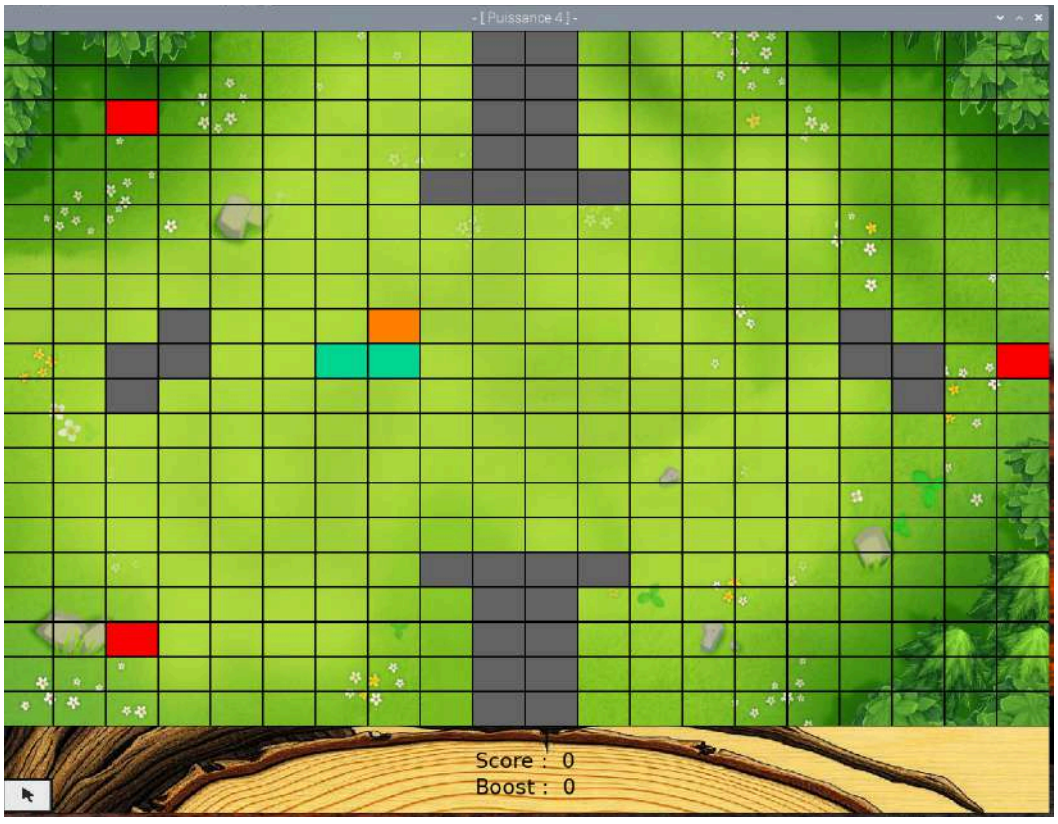
1.4 Puissance 4 Olympus

Un jeu de puissance 4 un peu différent. Comme toujours, il faut aligner 4 jetons de sa couleur. Cependant, à chaque fois qu'un joueur pose un jeton , il gagne une barre d'énergie. Les énergies des joueurs sont affichées sur les côtés du plateau de jeu, et il ne peut y avoir plus de 6 énergies pour chaque joueur. Les barres d'énergies peuvent être dépensées dans l'utilisation de sorts affichés plus bas. Les sorts en noirs ne peuvent être utilisés. Un bouton en bas à gauche permet d'afficher les règles du jeu et de présenter les différents sorts.
La touche LeftShift fait passer le jeu en mode **overdrive,** avec énergie max à 999 et 2 barres d'énergies gagnées par tour.



1.5 Snake

Un jeu de snake, ou le serpent est représenté par une suite de rectangles colorés. La tête a une couleur différente du corps, et le snake avance tout seul dans la dernière direction indiquée par le joueur avant le mouvement. Lorsque le snake mange un fruit représenté par des cases de couleurs rouges (fraises) ou jaunes (banane), sa taille ainsi que sa vitesse augmente. Il existe aussi des rochers gris , et si le snake rentre en collision avec l'un d'entre eux, il meurt. De la même manière, si le snake rentre en collision avec son corps, il meurt. Les fraises apparaissent automatiquement à un emplacement libre du tableau, et ont 30% de chance de réapparaître sous forme de banane. Les bananes donnent plus de points que les fraises lorsqu'elles sont mangées, mais elles deviennent des fraises au bout de quelques dizaines de déplacements.



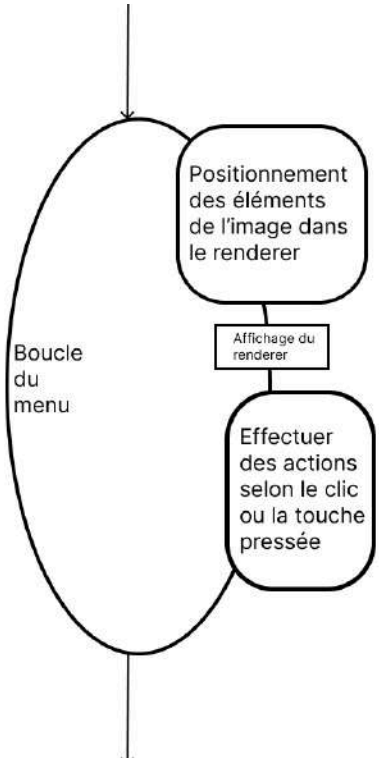
1.6 Levels of Snake

Levels of Snake est une version améliorée de snake. Le but n'est plus de manger le plus de fruit dans un plateau déterminé, mais d'avancer le plus loin possible dans une trentaine de niveaux. De nouvelles mécaniques de jeux sont introduites au fur et à mesure que le jeu avance, et il est possible de passer d'un niveau à l'autre en mangeant 33 fruits. On introduit ainsi de nouveaux fruits, de nouveaux obstacles, un timer, de la téléportation, des vies, des niveaux de difficulté qui influent sur le score, ainsi qu'un tutoriel de départ sur plusieurs menus. Levels of Snake est ainsi une expérience vidéoludique complète avec un début et une fin, du level design et des mécaniques exaltantes qui le fait passer de simple projet de programmation à véritable jeu vidéo. Et c'est pour cette raison que nous allons l'étudier plus en profondeur dans ce livre blanc.

2 - Levels of Snake

2.1 Menu Principal et Tutoriel

2.1.1 Structure d'un menu

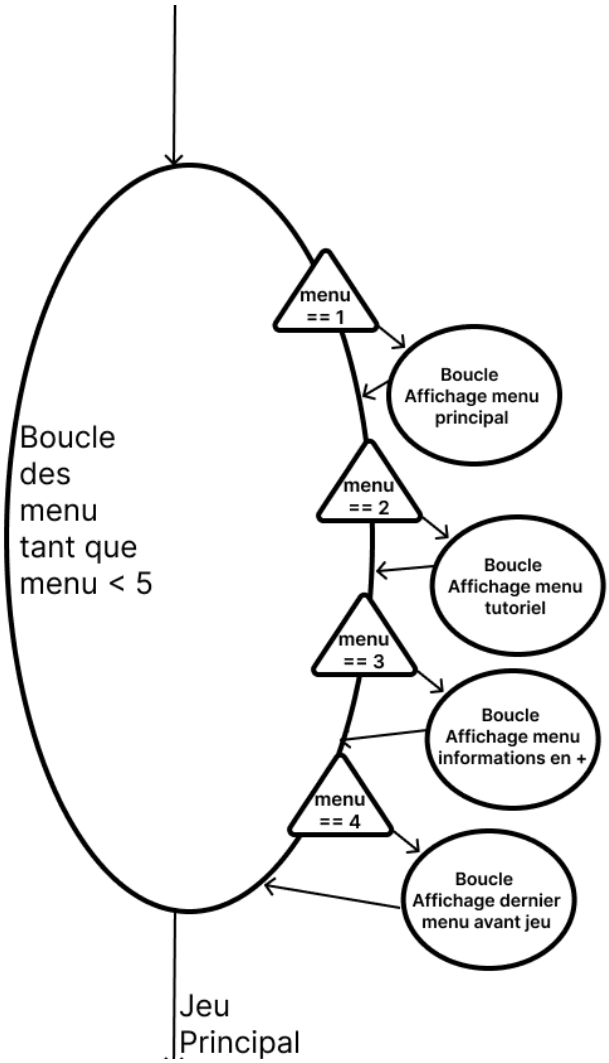


Pour chaque élément de l'image, on calcule sa position x et y, sa largeur, sa longueur et sa couleur en format rgba, que l'on insère dans une structure rectangle ou texte, puis on utilise cette structure pour dessiner la texture dans le renderer. Pour les textes, la largeur et la longueur est stockée dans une structure SDL_Rect elle-même stockée dans une structure texte, et calculée à partir de la police d'écriture et du texte que l'on veut écrire.

Pour les actions, on utilise SDL_PollEvent et on regarde la touche pressée. selon la touche, on peut changer certaines choses dans le menu. Dans le cas d'un bouton, une structure bouton est utilisée pour récupérer la taille et position des rectangles associés lors du positionnement des éléments de l'image dans le renderer. Ensuite, lors du clic, on regarde si la souris se trouve dans la zone du bouton, et on fait les actions en conséquence (changer de menu, choisir la difficultée, etc).

2.1 Menu Principal et Tutoriel

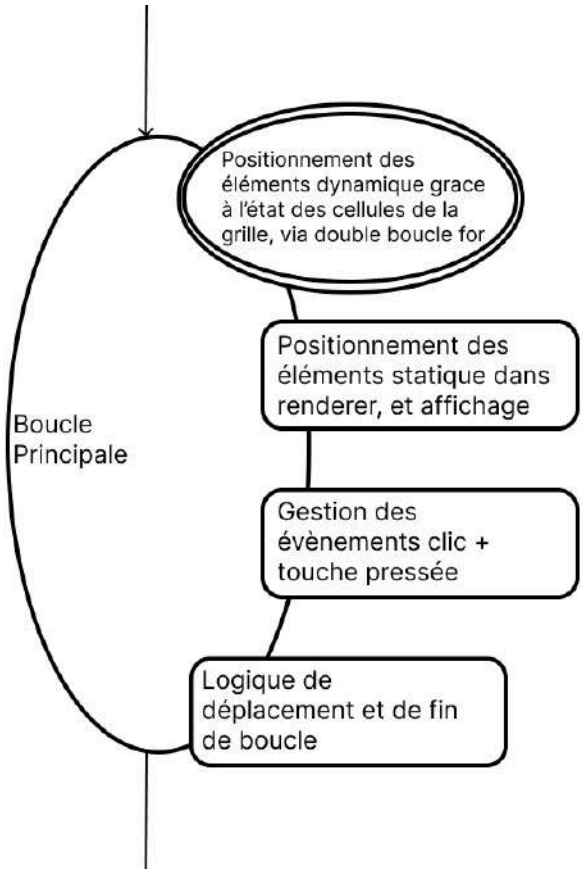
2.1.2 Navigation dans les menus



La navigation dans les menus repose sur une variable menu qui va indiquer quelle boucle de menu utiliser. Par exemple, si menu est égal à 1, on va utiliser la première boucle et afficher le menu principal. A l'intérieur des menus, selon le bouton sur lequel on appuie, menu va augmenter ou diminuer de 1, et on va sortir du menu pour aller utiliser une autre boucle. Il est important de noter que le menu principal ne comprend que des boutons de sélection de difficulté, qui incrémentent menu de 1 lorsque l'on clique dessus. De cette manière, menu ne peut jamais être égal à zéro.

2.2 Boucle Principale

2.2.1 Affichage



L'affichage est similaire aux autres menus, sauf pour les éléments dynamiques. Grâce à une double boucle for, on itère dans toute les cases d'un tableau 2D représentant le plateau lui-même et les objets qui y sont placés. Selon le numéro stocké dans ces cases, on va afficher un rectangle de différentes couleurs. Selon la position de ces cases, la position du rectangle va aussi changer. IL existe aussi une autre grille dont on parlera plus tard, appelée grille de texture, et qui est aussi utilisée lors de l'affichage pour les objets qui existent à un plan z plus élevé , et qui ne doivent pas disparaître lorsque le snake passe dessus (si on l'avait mis sur la même grille du bas, lorsque le snake serait passé dessus, la valeur de la case aurait changée pour représenter la tête du snake, et la valeur de la case aurait été perdue.

2.2 Boucle Principale

2.2.2 Au regard des touches

Le jeu possède un grand nombre de touches qui permettent de faire un grand nombre d'action. Le tout est expliqué à la fois dans les menus de tutoriel, dans la section boîte à outil, et dans le menu d'aide accéder en cliquant sur le bouton d'aide en bas à gauche pendant le jeu principal. Le principe de ces fonctionnalités est d'aider le joueur a customiser son expérience de jeu pour qu'elle corresponde le plus possible à ses attentes.

Il existe aussi d'autres touches qui permettent de faire des actions de débogage, comme un genre de backdoor dans le système:

- LeftShift : revient un niveau en arrière
- RightShift : avance d'un niveau
- u : lors d'un game over (non final) , change le message affiché

Le reste des touches est expliqué dans l'annexe.

Il est important de noter que, pour le déplacement qui utilise les touches zqsd, on appelle pas le déplacement du snake directement après. Par exemple, appuyer sur z ne fait pas se déplacer le snake vers le haut, il ne fait que mettre à jour la variable direction_actuelle vers le haut, variable qui sera utilisée au prochain déplacement quand il sera fait. A ce titre, le déplacement ne prend en compte que la dernière touche pressée, à l'inverse du véritable snake qui possède un buffer pour les déplacements plus compliqués (les touches pressées sont alors gardées en mémoire et utilisées automatiquement pour le premier déplacement, ce qui n'est pas le cas ici.)

2.3 Logique de jeu

2.3.1 Déplacement normal

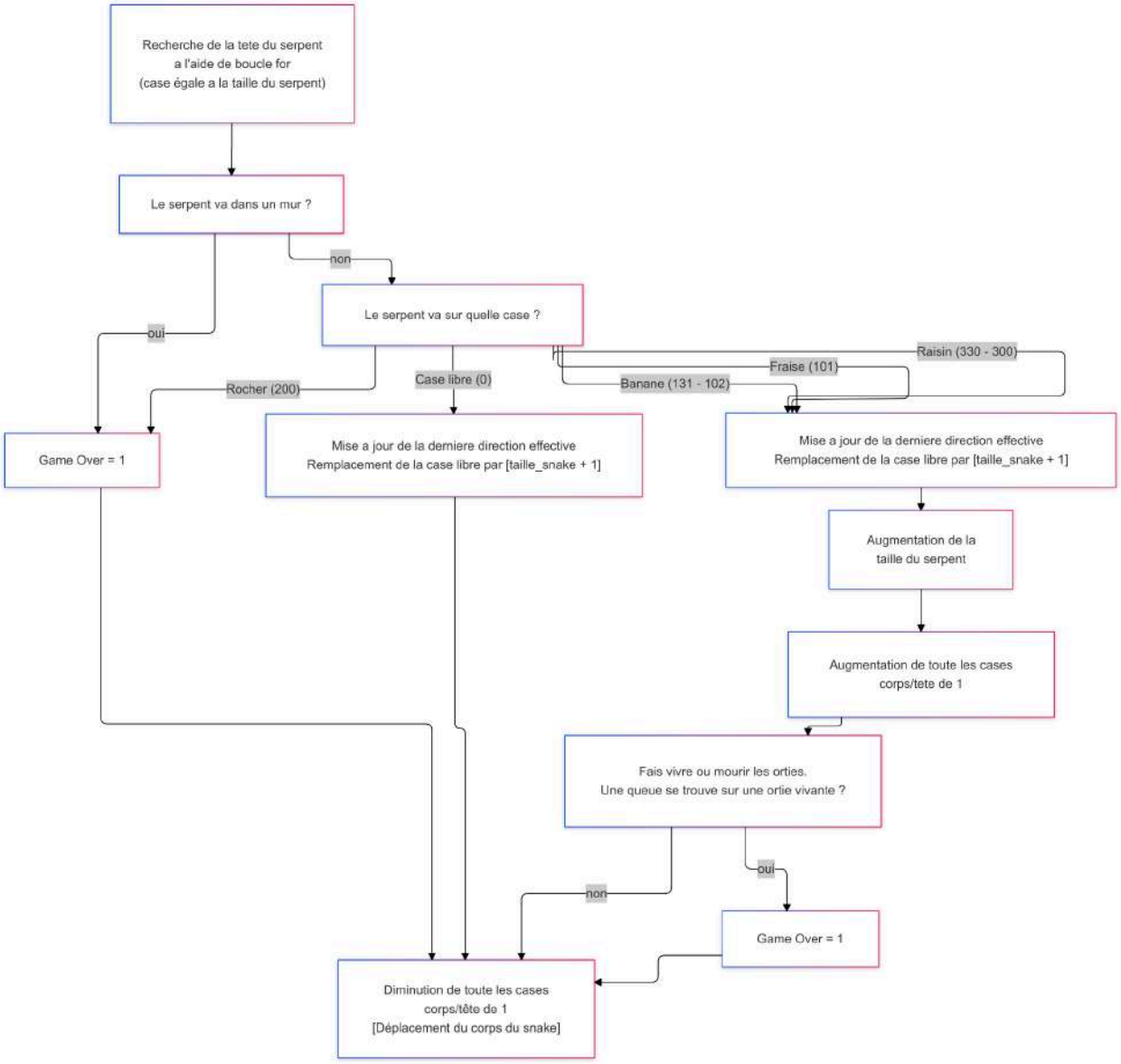
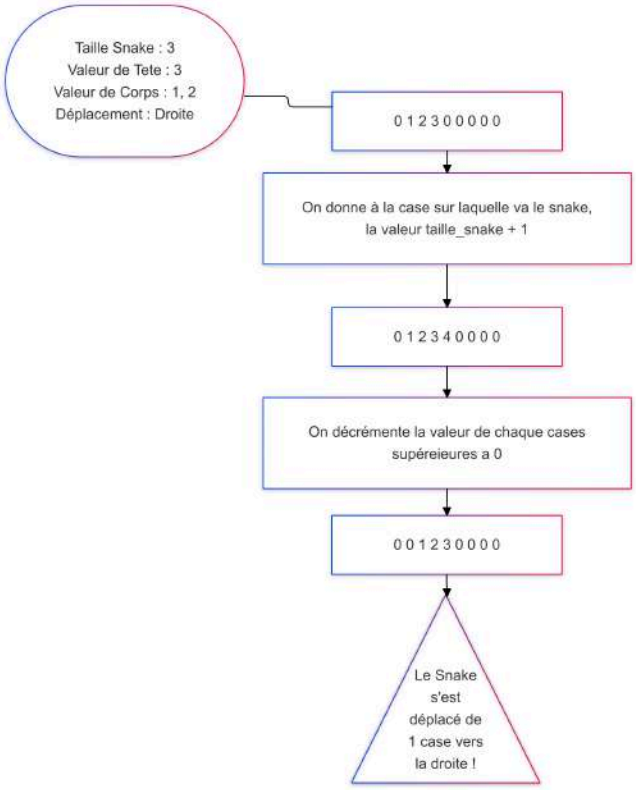


Diagramme schématique expliquant dans les grandes lignes le principe de déplacement et de nourriture. Ne comprend pas le système de téléportation, qui s'effectue entre la Recherche de la tête du serpent et le test de "serpent va il dans un mur". S' il y a une téléportation, le reste du déplacement est ignoré.

2.3 Logique de jeu

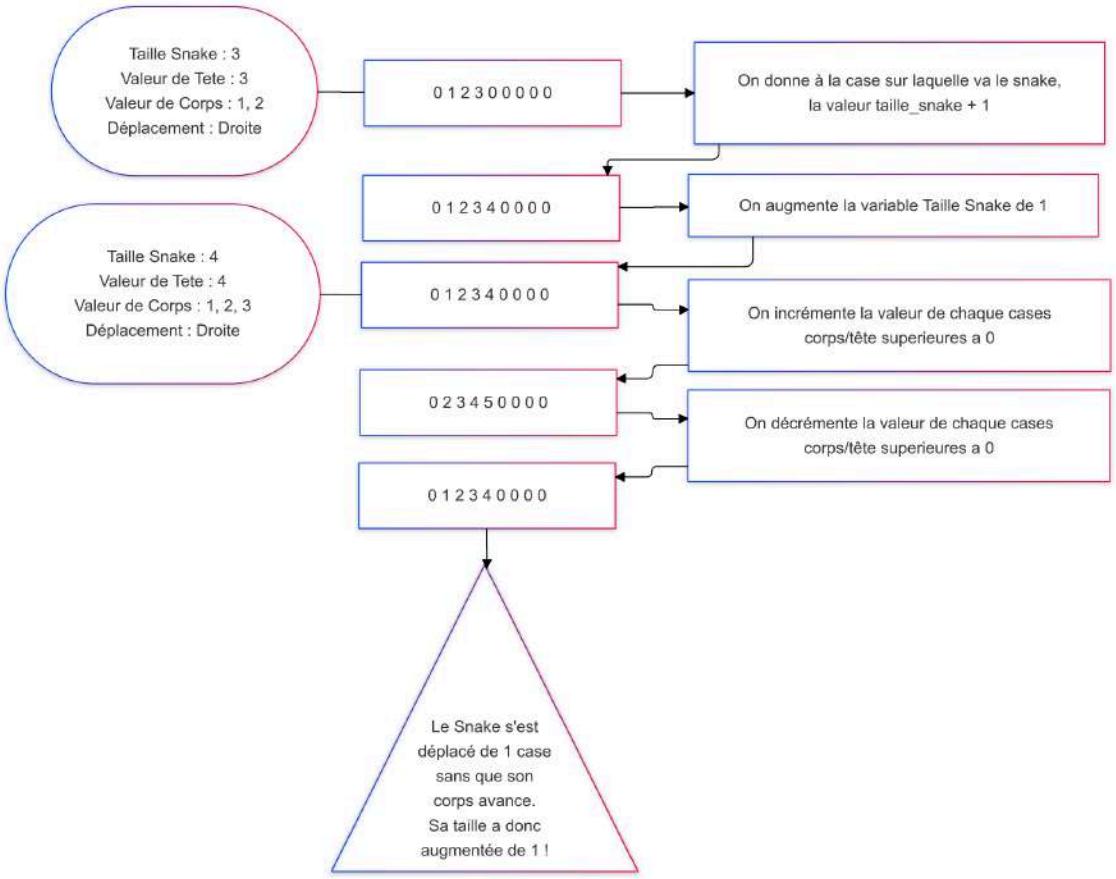
2.3.1 Déplacement normal (suite)



Dans les grandes lignes, le déplacement du snake se base sur un déplacement de la tête, qui possède une certaine valeur (égale à la taille du serpent), sur une grille. Ces déplacements laissent derrière eux des traces qui sont graduellement décrémentées avec chaque déplacement, et qui représentent le corps du snake.
Pour laisser cette trace, on donne à la case sur laquelle on va la valeur de la tête, incrémentée de 1. Cette incrémentation est alors contrebalancée par la décrémentation subséquente du déplacement, et on a alors une tête qui bouge d'un espace.
Si l'espace sur lequel la tête se déplace est occupé par un nombre qui ne signifie ni un fruit, ni un espace vide, c'est donc un obstacle (rocher ou corps du snake). Dans ce cas, le déplacement ne se fait pas, la décrémentation liée au déplacement se fait et efface alors la tête, et une variable game_over passe sur True, arrêtant ainsi la partie.

2.3 Logique de jeu

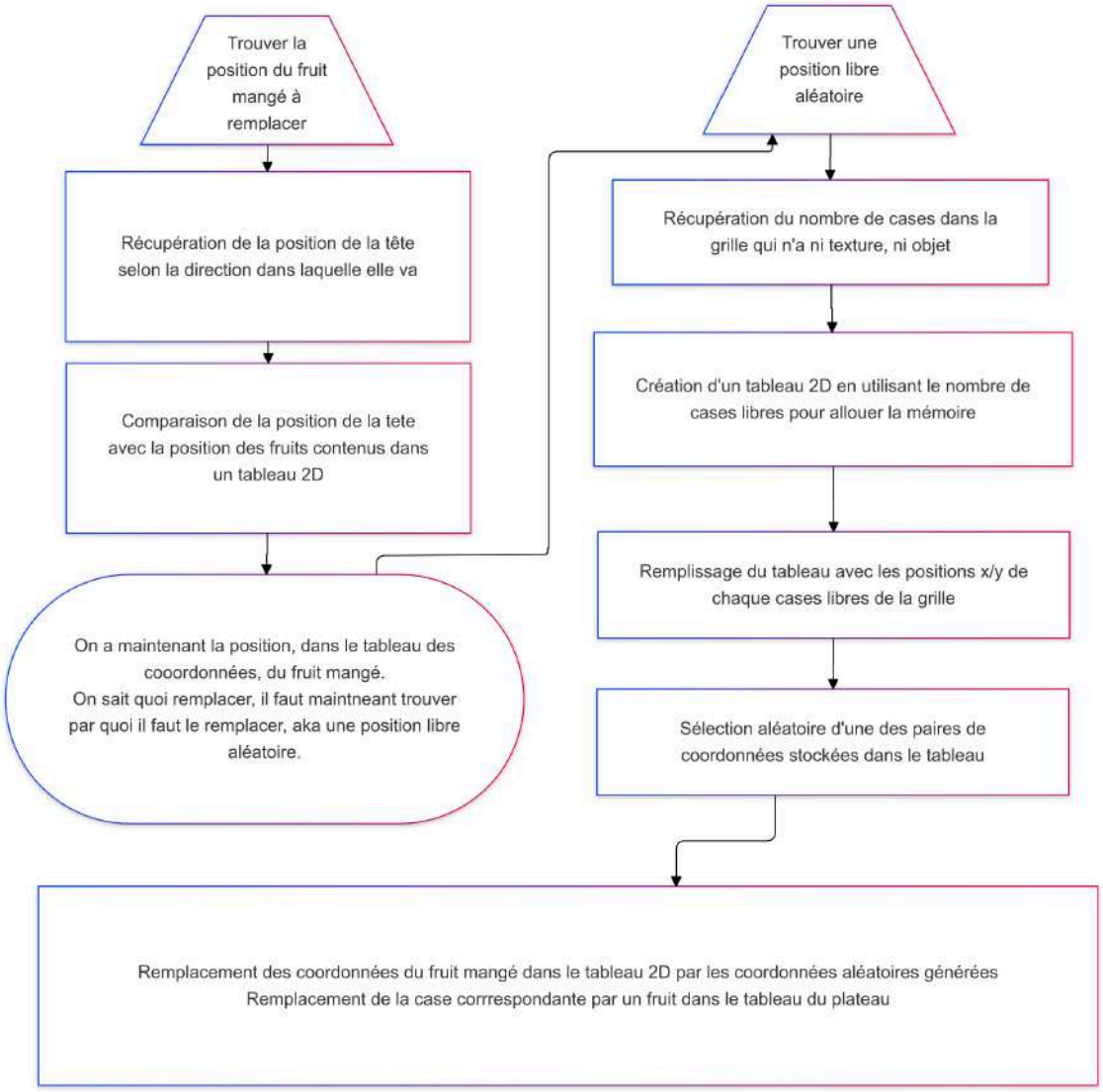
2.3.2 Déplacement sur fruit



Le déplacement du snake sur le fruit se résume à un déplacement normal et à une incrémentation de chaque partie du snake pour contrecarrer la décrémentation liée à son déplacement. C'est comme un déplacement normal ou on place la valeur de la tête +1 sur la case de déplacement, mais on empêche la diminution de toutes les cases de snake pour avoir l'illusion que la tête avance, mais pas le corps, augmentant ainsi la taille du snake de 1.
Cette logique est valable pour le déplacement sur une fraise, une banane ou du raisin, avec seulement le nombre de points ajoutés au score qui change.
Pour le raisin, une autre logique est ajoutée, sont on parlera plus tard.

2.3 Logique de jeu

2.3.2.1 Réapparition des fruits



Lors de l'ingestion d'un fruit, une variable "fruit_mange" devient True. La logique ci-dessus est alors enclenchée afin de trouver, dans un tableau contenant la position de tous les fruits sur le plateau, la position du fruit à remplacer. Ensuite, on récupère une position libre aléatoire dans la grille, et l'utilise pour remplacer les coordonnées du fruit à remplacer dans son tableau 2D. Enfin, on utilise aussi ces coordonnées pour placer le fruit sur la grille de jeu.

2.3 Logique de jeu

2.3.3 Méchanique du raisin

Le raisin est un fruit particulier qui disparaît au bout d'un certain temps. Pour représenter ce concept, on utilise la même logique que la fraise, a ceci près que le nombre qui représente un raisin n'est pas un seul nombre, mais une gamme allant de 300 à 330. A chaque décrémentation en fin de déplacement de snake, on décrémente aussi toute les cases qui possèdent une valeur entre 300 et 330, et lorsqu' une case possède la valeur 300, on considère que c'est un raisin qui doit disparaître. Dans ce cas, on fait appel à la logique de réapparition des fruits pour éliminer les coordonnées du raisin à remplacer, stockées dans un tableau 2D, et les remplacer par les coordonnées d'une autre case libre. Pas besoin de récupérer les coordonnées de la tête, car on possède directement les coordonnées de la case contenant le raisin mort (case étant égale à 300).
En clair, le raisin est déposé sur le plateau de jeu avec une valeur de 330, qui diminue avec chaque déplacement du snake, et se fait remplacer lorsque sa valeur atteint 300.

À des fins de compréhension des mécanique à travers le game design, les composantes rgb de la couleur du raisin dépendent de sa valeur selon la fonction suivante :

```
// Plus x est proche de 300, plus le facteur est faible (donc la couleur devient plus foncée)
float facteur = ((330.0f - tableau_etat_cellule[pos_y][pos_x]) / (330.0f - 300.0f)); // facteur entre 0 et 1
// Appliquer un amortissement pour que le facteur ne descende jamais en dessous de 0.2
facteur = 0.45f + (0.8f * exp(-facteur * 1))
```

On a alors un raisin dont la couleur violette s'assombrit avec le temps jusqu'a devenir proche du noir avant de mourir.

2.3 Logique de jeu

2.3.4 Déplacement sur orties

Les orties sont des obstacles à tangibilité variable. A la manière d'un nombre binaire ou d'une porte, elles peuvent être sous deux formes : mortes ou vivantes. Les orties mortes ont une couleur verte plus foncée que leur homologue vivante, et le snake peut les traverser. Les orties vivantes agissent comme un obstacle, et provoquent un game over si on tente de les traverser. L'état des orties, mortes ou vivantes, traversables ou non, changent à chaque fois que le snake mange un fruit. Il est important de noter que si un bout du snake se trouve sur une ortie morte au moment où cette dernière devient vivante, cela provoque aussi un game over.

Au niveau de la logique, on itère simplement sur la grille avec une double boucle for , et on change les cases à valeur 400 (orties mortes) en 500 (orties vivantes), et vice versa. On vérifie aussi qu'une case contenant un bout du snake ne se trouve pas au même endroit qu'une case contenant une sortie vivante.

Les orties sont un élément différent de ceux que l'on a rencontrés jusqu'à maintenant. A la différence des fruits qui disparaissent quand on passe dessus, ou des roches qui ne bougent pas quand on rentre dedans , les orties sont un élément que l'on peut traverser sans effacer sa position. En utilisant la logique vue précédemment, on voit que le serpent remplace les valeurs des cases sur lesquelles il passe. En clair, il est absolument nécessaire de développer une autre logique pour permettre d'avoir un objet ayant un effet tangible sur le snake et le tableau de jeu, sans que son existence ne soit liée à la grille.
Cette logique, c'est la grille de texture.

2.3 Logique de jeu

2.3.4.1 Grille de Texture

On peut voir la grille de texture comme un deuxième plateau de jeu se trouvant à une élévation supérieure. L'affichage de cette grille se fait après l'affichage du plateau de jeu, comme pour recouvrir le tout. La grille n'est responsable que des objets qui peuvent être traversés sans changer leur position. Il y a, pour l'instant, 3 types d'objets sur la grille de texture :

- Les orties vivantes ou mortes (400 ou 500)
- Les trous d'eau (600 a 759)
- Les caches invisibles (999)

L'utilisation d'une telle solution pour ces objets nécessite la mise à jour des logiques précédentes ou qu'elle prenne aussi en compte la grille de texture. Cela veut dire :

- Lors de la récupération des cases vides pour la sélection d'une nouvelle position de fruit, il faut aussi regarder si il n'y a pas d'objets sur la grille de texture (pour éviter qu'une fraise apparaisse sur une ortie par exemple)
- Lors de l'affichage, il faut regarder s' il n'y a pas d'objet qu'il faudrait afficher et qui se trouve sur la grille de texture .
- Lors de la vérification de "snake sur une ortie morte devenant vivante", on regarde si une case ortie vivante contenue sur la grille de texture n'a pas de bout de snake sur la grille de plateau.

On se met alors à jongler entre les deux grilles pour appliquer certaines logiques.

2.3 Logique de jeu

2.3.5 Déplacement sur Trou d'eau

Les trous d'eau sont une version remaniée d'un téléporteur. Lorsqu'on le prend dans le bon sens, représenté par une flèche noire sur le dessus, il permet au snake de se déplacer au trou d'eau correspondant. La logique implémentée prend aussi en compte un changement de direction du snake à la sortie du trou d'eau correspondant. Il est important de noter que le jeu considère "trou d'eau pris dans le bon sens" comme étant "la tête du snake est sur la case du trou d'eau, et la direction de déplacement est la même que la direction du trou d'eau". Il n'est ainsi pas nécessaire de rentrer sur la case téléporteur avec la bonne direction, juste d'aller dans la bonne direction quand le snake est sur la case trou d'eau.

En termes de logique, on regarde si la tête du snake se trouve sur une case trou d'eau stockée dans la grille de téléporteur. Ensuite, selon le numéro de la case, on obtient les informations suivantes :

- Dans quel sens le snake doit aller pour que la téléportation soit effective
- Quel sens aura le snake a la sortie du téléporteur

On a alors plus qu'à comparer la direction de déplacement avec la direction à prendre, et si il y a match, on change la direction de déplacement avec celle que le snake est censé avoir a la sortie. Enfin, on cherche le trou d'eau correspondant et on applique la valeur de la tête à sa position.

Pour trouver cette sortie, on regarde toutes les cases de la grille de texture, et on cherche celle qui possède la même unité que la case trou d'eau "entrée", mais qui a une position différente de la case trou d'eau "entrée".

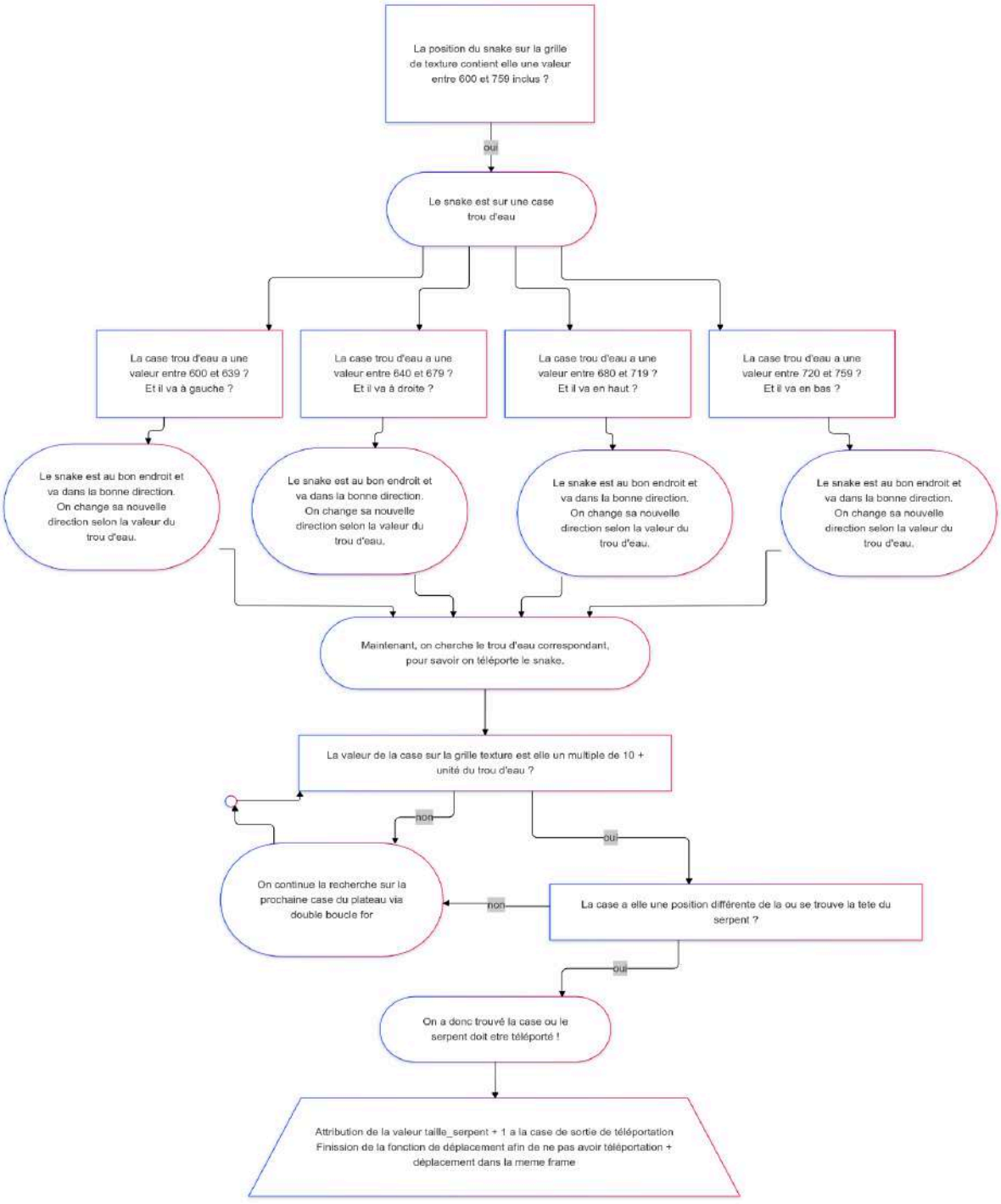
En clair, un trou d'eau avec une valeur 614 ne peut mener qu'à un trou d'eau avec une valeur 604, 624, 634, 644, 654, 664, 674, 684, 694, 704, 714, 724, 734, 744 ou 754. Cette logique apporte avec elle deux restrictions :

- Il ne peut y avoir que 10 paires de téléporteurs (reliés entre eux par les unités 0,1,2,3,4,5,6,7,8 et 9).
- On ne peut pas relier deux trous d'eau avec les mêmes "directions à prendre" et "direction que le snake aura à la sortie" .

Les valeurs de trou d'eau correspondantes sont en annexe.

2.3 Logique de jeu

2.3.5 Déplacement sur Trou d'eau (suite)



2.4 Éditeur de niveaux

2.4.1 Template

Le jeu possède 33 niveaux. Il a donc fallu intégrer une certaine logique pour pouvoir les créer de manière plus simple. Pour cela, on sépare les données du niveau en 8 informations stockées de différentes manières :

- Une grille de 20 par 20 contenant les objets réels de chaque niveau (rochers [200], tête du serpent [3]).
- Une grille de 20 par 20 correspondant à la grille de texture de chaque niveau (trou d'eau, orties, caches invisibles).
- Un tableau de 3 par 2 contenant les coordonnées y:x de 3 fraises. Ces coordonnées sont utilisées à la génération du niveau, mais une fois les fraises mangées elles réapparaissent sur n'importe quelle case vide. Les fraises et raisins ne sont donc pas stockés directement sur la grille de plateau.
- Un tableau de 3 par 2 contenant les coordonnées y:x de 3 raisins.
- Une variable d'activation des fraises. Si elle est sur 0, alors les fraises ne se généreront pas dans le niveau. Cela évite d'avoir des fraises qui apparaissent dans un niveau réservé aux raisins.
- Une variable d'activation des raisins.
- Une variable d'activation de timer. Indique si oui ou non il y a un timer dans le niveau.
- Une variable de timer. Indique le temps donné au joueur pour finir le niveau.

De par la manière dont cet "éditeur de niveau" fonctionne, si on permet la génération d'un fruit, et que l'on donne les mêmes coordonnées à 2 fruits sur les 3 présents, alors seulement 2 fruits apparaîtront dans le niveau. Cela permet d'avoir un contrôle sur le nombre de fruits d'un même type. De la même manière, donner 3 mêmes coordonnées à un fruit n'en fera apparaître qu'un seul.
Par mesure de précaution, on donne aux fruits qui ne doivent pas apparaître la même position y:x qu'un rocher [200].
Un exemple de template est disponible en annexe.

2.4 Éditeur de niveaux

2.4.2 Injection

Une fois le template généré, on le stocke dans une grande base de données.
Par exemple, base_de_donnee_grille est un triple pointeur, c'est-à- dire un tableau contenant 33 tableaux 2D de 20 par 20. Chaque tableau 2D de 20 par 20 correspond à un niveau. On fait la même chose pour base_de_donnee_texture, qui contient 33 grilles de textures de 20 par 20, une pour chaque niveaux.
Pour base_de_donnee_fraise et base_de_donnee_raisin, c'est des tableaux contenant 33 grilles de 3 par 2, correspondant aux tableaux de 3 par 2 contenant les positions initiales de fraises et raisins.
Pour les données d'activation de fraise, raisin, timer, et même juste les limite de temps des timers, on le stocke directement dans un tableau simple.
Ces "bases de données" utilisent des allocation statiques pour les tableaux, ce qui veut dire que :

- Il existe une limite théorique au nombre de niveaux stockables, déterminés par la taille de la pile.
- A chaque fois que l'on crée un tableau, il faut changer le nombre de pointeurs que l'on alloue aux tableaux

Il est important de noter que, a la différence des données d'activations qui sont directement rentrées dans la base de donnée, les grilles de plateau, de texture, et même les tableaux de coordonnées sont d'abord entrés dans des pointeurs (ex : grille_fraise_level_one jusqu'à grille_fraise_level_thirtythree), et qu'on utilise ensuite ces pointeurs pour les rentrer dans les bases de données respectives avec un memcpy.
Pour ce qui est du nom de chaque niveau, ils sont traités en interne de l'affichage de la boucle principale, dans un pointeur tableau_nom_de_niveaux.

Un aperçu de l'initialisation de la base de données se trouve en annexe.

3 - Problèmes Rencontrés

3.1 Demi-tour

Dû à la logique de jeu, il a fallu trouver un moyen d'empêcher au snake d'aller dans la direction opposée à sa direction actuelle, ou courir le risque qu'il se mange tout seul. Pour cela, on utilise deux variables : dernier_deplacement_effectif et direction.
dernier_deplacement_effectif n'est mis a jour qu'après un déplacement, et garde en mémoire le dernier déplacement effectué.
La direction est mise à jour après chaque pression de touche de déplacement, et est utilisée pour effectuer le déplacement. Cependant, il ne sera pas mis à jour si on presse une touche concernant une direction opposée a dernier_deplacement_effectif .

3.2 Dédoublement

Dans le 31ème niveau du jeu, on contrôle deux snake. Dans l'idée , cela aurait du être une fonctionnalité complète affectée à quelques niveaux car la logique permet déjà de prendre en compte 2 snake (puisqu'elle ne regarde pas la tête du serpent pour le faire se déplacer, mais toute les cases dont la valeur est égale à la tête du serpent), cependant quelques problèmes sont survenus :

- Quand un des serpents mange un fruit, la tête du deuxième disparaît. Cela est dû à la compensation de la décrémentation des parties du serpent qui n'est appliquée que pour le premier serpent. La tête du deuxième disparaît alors avec la décrémentation.
  Le problème a été réglé pour le raisin, et consiste à rechercher une deuxième tête pour y appliquer une compensation de décrémentation.
- Un fruit peut apparaître sur un case accessible par un des serpents, mais pas par l'autre.
  Pour cela, on utilise des caches invisibles [999] sur la grille de texture, ce qui crée alors des objets sans collisions ou effets, mais qui font passer les cases de "endroit viable pour un fruit" à "endroit occupé par un objet".

3.3 Performances

Avec htop, le jeu utilisait 100% d'un cœur lors de l'affichage des menus. Un mécanisme de limitation du nombre de frame affiché par seconde a été mis en place, utilisant le temps entre chaque affichage et une fonction SDL_Delay permettant l'attente du programme pour limiter l'affichage (et par conséquent la mise en place d'une nouvelle itération de la boucle d'affichage) à un nombre acceptable. Cependant, l'affichage était toujours trop faible. Il y a ainsi eu un ajout d'une condition dans la boucle qui checke les événements claviers pour que l'on doive en sortir obligatoirement lorsqu'une touche est pressée. L'addition de cette condition et du mécanisme de limitation de fps semble avoir réglé le problème.

3.4 Déplacement entre les menus

Appuyer sur une touche une fois, ou cliquer sur une des flèches une fois pouvait faire avancer le menu de plusieurs pages, comme si on avait cliqué plusieurs fois sur les flèches. La solution mise en place à l'époque a été de décaler les flèches d'un menu à un autre pour éviter de cliquer sur la même flèche. Mais au vu des dernières informations, on aurait aussi pu regarder non pas l'enfoncement du clic de la souris, mais le relâchement du clic de la souris, pour être sûr que le programme n'utilise pas un même clic sur plusieurs fenêtres différentes.

3.5 Visibilité

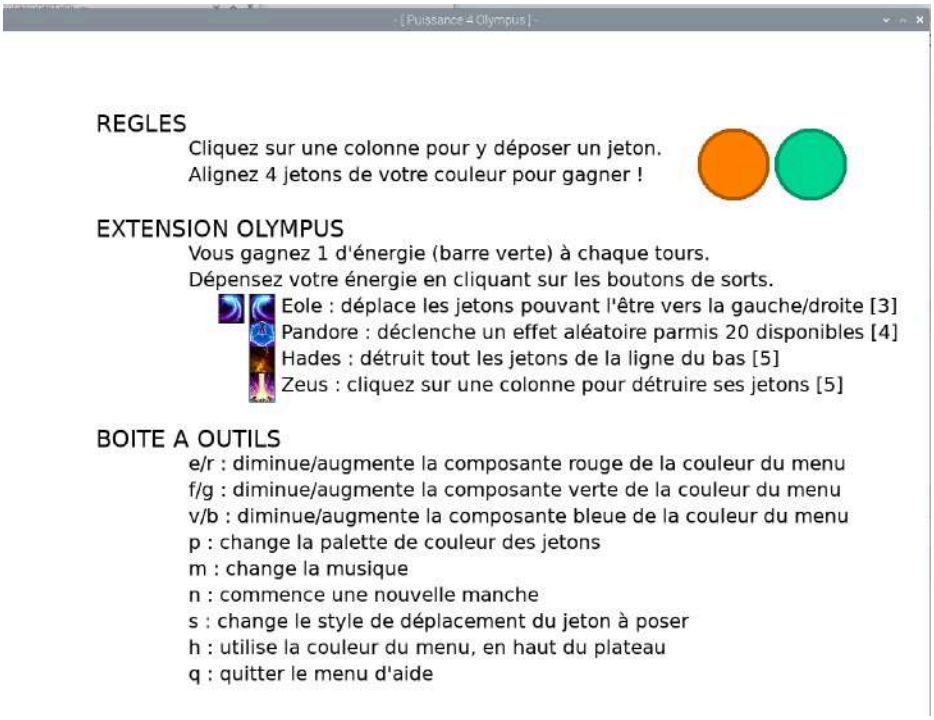
A cause des nombreuses images et des différentes couleurs , le plateau de jeu était difficile à comprendre. Il a donc fallu rajouter des modes d'affichage différents qui permettent de mieux distinguer les éléments du décor des objets de plateau. Selon le mode d'affichage, au niveau de la boucle principale, on active ou non les contours et les images, juste avant le renderer, au niveau de la double boucle for.

3.6 Messages de Game Over

Les messages de game over sont stockés dans un tableau alloué statiquement. L'idée principale était de générer un nombre aléatoire et de l'utiliser pour aller piocher un message dans le tableau, à afficher au joueur. Cependant, à chaque nouvelle frame, un nouveau nombre aléatoire était tiré, et un nouveau message s'affichait. Il a fallu mettre une condition selon laquelle le nombre aléatoire est tiré seulement s' il a une valeur de 99. Enfin, à chaque fois que l'on recommence le niveau, on remet cette variable aléatoire à 99.

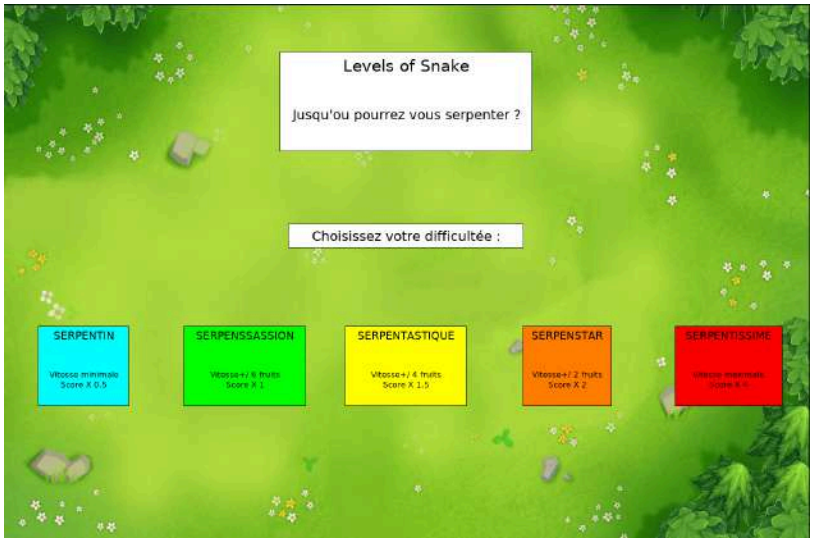
4 - Annexe

Pouvoirs de Puissance 4 Olympus



Les pouvoirs permettent de modeler le plateau de jeu a l'image de la stratégie de ses joueurs, et rendent chaque partie incroyablement complexes et longues. On peut se retrouver après une trentaine de coups avec un plateau complètement vidé par l'utilisation de pouvoirs destructeurs.

Menu de difficultée

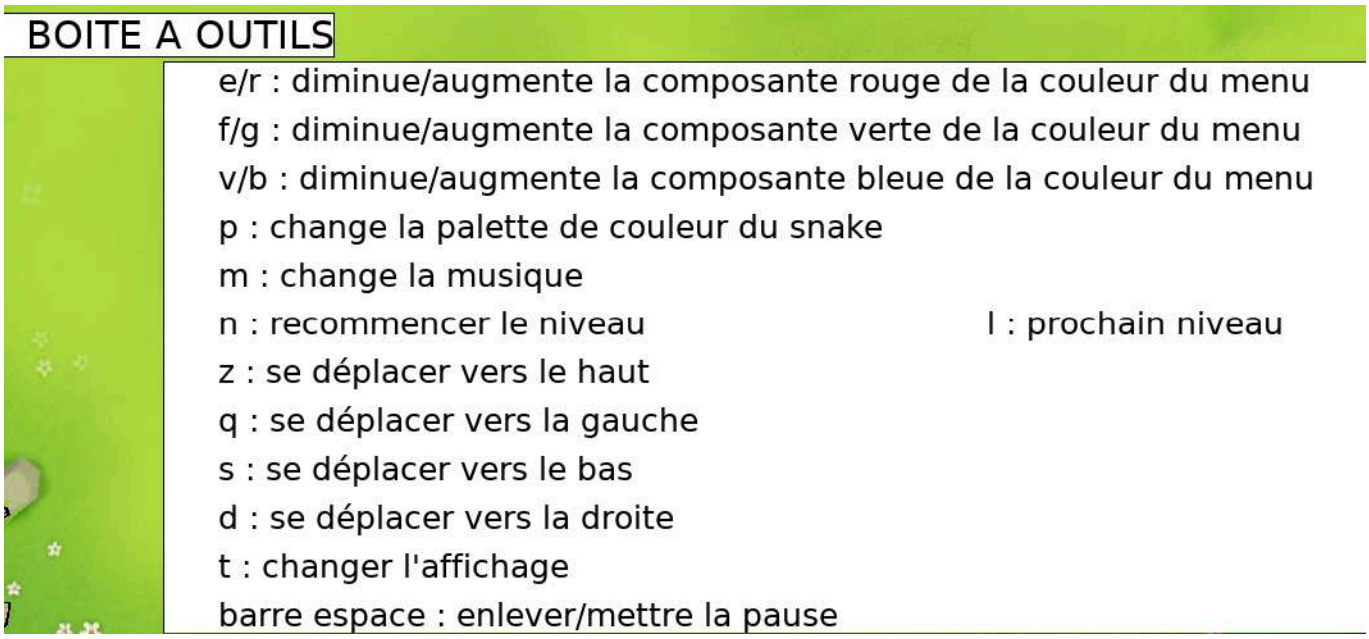


Chaque rectangle de couleur est un bouton qui met en place la difficultée, et incrémente la variable menu de 1

Menu tutoriel

Les flèches en bas sont des boutons qui incrémentent ou décrémentent la variable menu pour passer d'un menu à un autre

Boîte à outils



Les touches accessibles aux utilisateurs pour customiser leur expérience de jeu

Valeurs correspondantes de trou d'eau

```
// est ce que le serpent va sur un trou d'eau ?
// trou d'eau :
//       600 - 609 = rentre gauche sors gauche
//       610 - 619 = rentre gauche sors droite
//       620 - 629 = rentre gauche sors haut
//       630 - 639 = rentre gauche sors bas
//       640 - 649 = rentre droite sors gauche
//       650 - 659 = rentre droite sors droite
//       660 - 669 = rentre droite sors haut
//       670 - 679 = rentre droite sors bas
//       680 - 689 = rentre haut sors gauche
//       690 - 699 = rentre haut sors droite
//       700 - 709 = rentre haut sors haut
//       710 - 719 = rentre haut sors bas
//       720 - 729 = rentre bas sors gauche
//       730 - 739 = rentre bas sors droite
//       740 - 749 = rentre bas sors haut
//       750 - 759 = rentre bas sors bas
// et chaque trou d'eau emmene a un trou d'eau dont les unités sont similaires.
// ex : trou 652 emmene sur trou 662 ou 672 ou meme 712.
// en clair : seulement 10 portails autorisés
```

Les valeurs de trou d'eau sont aussi disponibles en commentaire, dans la fonction deplacement_snake.

Affectation du niveau 11

```
602  const int grille_level_eleven[20][20] = {
603      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
604      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
605      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
606      {0, 0, 0, 0, 200, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 200, 0, 0, 0, 0},
607      {0, 0, 0, 200, 200, 200, 0, 0, 0, 0, 0, 0, 0, 0, 200, 200, 200, 0, 0, 0},
608      {0, 0, 0, 0, 200, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 200, 0, 0, 0, 0},
609      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
610      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
611      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
612      {0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
613      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
614      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
615      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
616      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
617      {0, 0, 0, 0, 200, 0, 0, 0, 0, 0, 0, 0, 0, 0, 200, 0, 0, 0, 0},
618      {0, 0, 0, 200, 200, 200, 0, 0, 0, 0, 0, 0, 0, 0, 200, 200, 200, 0, 0, 0},
619      {0, 0, 0, 0, 200, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 200, 0, 0, 0, 0},
620      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
621      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
622      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
623
624  };
625
626  const int grille_texture_level_eleven[20][20] = {
627      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
628      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
629      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
630      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
631      {0, 0, 0, 0, 0, 0, 400, 400, 400, 400, 400, 400, 400, 400, 0, 0, 0, 0, 0, 0},
632      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
633      {0, 0, 0, 0, 0, 500, 0, 0, 0, 0, 0, 0, 0, 0, 0, 500, 0, 0, 0, 0},
634      {0, 0, 0, 0, 500, 0, 0, 0, 0, 0, 0, 0, 0, 0, 500, 0, 0, 0, 0},
635      {0, 0, 0, 0, 500, 0, 0, 0, 0, 0, 0, 0, 0, 0, 500, 0, 0, 0, 0},
636      {0, 0, 0, 0, 500, 0, 0, 0, 0, 0, 0, 0, 0, 0, 500, 0, 0, 0, 0},
637      {0, 0, 0, 0, 500, 0, 0, 0, 0, 0, 0, 0, 0, 0, 500, 0, 0, 0, 0},
638      {0, 0, 0, 0, 500, 0, 0, 0, 0, 0, 0, 0, 0, 0, 500, 0, 0, 0, 0},
639      {0, 0, 0, 0, 500, 0, 0, 0, 0, 0, 0, 0, 0, 0, 500, 0, 0, 0, 0},
640      {0, 0, 0, 0, 500, 0, 0, 0, 0, 0, 0, 0, 0, 0, 500, 0, 0, 0, 0},
641      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
642      {0, 0, 0, 0, 0, 0, 400, 400, 400, 400, 400, 400, 400, 400, 0, 0, 0, 0, 0, 0},
643      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
644      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
645      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
646      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
647
648  };
649
650  const int grille_fraise_level_eleven[3][2] = {{9, 0},
651                                                {10, 0},
652                                                {11, 0}};
653
654  const int grille_raisins_level_eleven[3][2] = {{3, 4},
655                                                 {3, 4},
656                                                 {3, 4}};
```

On fait spawner les raisins au même endroit, à la position d'un rocher, pour être sûr qu'il n'apparaissent jamais et ne soient jamais mangés. Ce n'est qu'une précaution cependant, car le tableau de données d'activation s'assure qu'un fruit ne puisse pas apparaître si l'on ne le permet pas.

Injection des grilles de coordonnées et données d'activation dans "db"

```
3558    int base_de_donnee_raisins[33][3][2];
3559    memcpy(base_de_donnee_raisins[0], grille_raisins_level_one, sizeof(grille_raisins_level_one));
3560    memcpy(base_de_donnee_raisins[1], grille_raisins_level_two, sizeof(grille_raisins_level_one));
3561    memcpy(base_de_donnee_raisins[2], grille_raisins_level_three, sizeof(grille_raisins_level_one));
3562    memcpy(base_de_donnee_raisins[3], grille_raisins_level_four, sizeof(grille_raisins_level_one));
3563    memcpy(base_de_donnee_raisins[4], grille_raisins_level_five, sizeof(grille_raisins_level_one));
3564    memcpy(base_de_donnee_raisins[5], grille_raisins_level_six, sizeof(grille_raisins_level_one));
3565    memcpy(base_de_donnee_raisins[6], grille_raisins_level_seven, sizeof(grille_raisins_level_one));
3566    memcpy(base_de_donnee_raisins[7], grille_raisins_level_eight, sizeof(grille_raisins_level_one));
3567    memcpy(base_de_donnee_raisins[8], grille_raisins_level_nine, sizeof(grille_raisins_level_one));
3568    memcpy(base_de_donnee_raisins[9], grille_raisins_level_ten, sizeof(grille_raisins_level_one));
3569    memcpy(base_de_donnee_raisins[10], grille_raisins_level_eleven, sizeof(grille_raisins_level_one));
3570    memcpy(base_de_donnee_raisins[11], grille_raisins_level_twelve, sizeof(grille_raisins_level_one));
3571    memcpy(base_de_donnee_raisins[12], grille_raisins_level_thirteen, sizeof(grille_raisins_level_one));
3572    memcpy(base_de_donnee_raisins[13], grille_raisins_level_fourteen, sizeof(grille_raisins_level_one));
3573    memcpy(base_de_donnee_raisins[14], grille_raisins_level_fifteen, sizeof(grille_raisins_level_one));
3574    memcpy(base_de_donnee_raisins[15], grille_raisins_level_sixteen, sizeof(grille_raisins_level_one));
3575    memcpy(base_de_donnee_raisins[16], grille_raisins_level_seventeen, sizeof(grille_raisins_level_one));
3576    memcpy(base_de_donnee_raisins[17], grille_raisins_level_eighteen, sizeof(grille_raisins_level_one));
3577    memcpy(base_de_donnee_raisins[18], grille_raisins_level_nineteen, sizeof(grille_raisins_level_one));
3578    memcpy(base_de_donnee_raisins[19], grille_raisins_level_twenty, sizeof(grille_raisins_level_one));
3579    memcpy(base_de_donnee_raisins[20], grille_raisins_level_twentyone, sizeof(grille_raisins_level_one));
3580    memcpy(base_de_donnee_raisins[21], grille_raisins_level_twentytwo, sizeof(grille_raisins_level_one));
3581    memcpy(base_de_donnee_raisins[22], grille_raisins_level_twentythree, sizeof(grille_raisins_level_one));
3582    memcpy(base_de_donnee_raisins[23], grille_raisins_level_twentyfour, sizeof(grille_raisins_level_one));
3583    memcpy(base_de_donnee_raisins[24], grille_raisins_level_twentyfive, sizeof(grille_raisins_level_one));
3584    memcpy(base_de_donnee_raisins[25], grille_raisins_level_twentysix, sizeof(grille_raisins_level_one));
3585    memcpy(base_de_donnee_raisins[26], grille_raisins_level_twentyseven, sizeof(grille_raisins_level_one));
3586    memcpy(base_de_donnee_raisins[27], grille_raisins_level_twentyeight, sizeof(grille_raisins_level_one));
3587    memcpy(base_de_donnee_raisins[28], grille_raisins_level_twentynine, sizeof(grille_raisins_level_one));
3588    memcpy(base_de_donnee_raisins[29], grille_raisins_level_thirty, sizeof(grille_raisins_level_one));
3589    memcpy(base_de_donnee_raisins[30], grille_raisins_level_thirtyone, sizeof(grille_raisins_level_one));
3590    memcpy(base_de_donnee_raisins[31], grille_raisins_level_thirtytwo, sizeof(grille_raisins_level_one));
3591    memcpy(base_de_donnee_raisins[32], grille_raisins_level_thirtythree, sizeof(grille_raisins_level_one));
3592
3593    int base_de_donnee_activation_fraise[33] = {1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1};
3594    int base_de_donnee_activation_raisin[33] = {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1};
3595    int base_de_donnee_activation_timer[33] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0};
3596    int base_de_donnee_timer[33] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 100, 100, 100, 100, 100, 0, 0, 100, 0, 100, 0, 0, 0};
```

L'utilisation de structures et d'allocation dynamique aurait grandement réduit la complexité de la "base de donnée"

Ces jeux n'ont pas pour objectif la perfection d'un projet de développement, mais l'application de techniques et de connaissances nécessaire à l'avancée sur la route de l'apprentissage. Je n'y jouerais pas forcément, mais leurs imperfections me serviront à faire de bien meilleurs projets dans le futur !

« Le succès est le résultat de la préparation, de l'effort et de l'apprentissage de l'échec. »

– Colin Powell