Rapport WEB

MERRET Damien

Pour le projet je suis reparti du TP2, la mécanique et le fonctionnement des fichiers est donc très similaire.

Le projet doit tourner sur un serveur pour pouvoir récupérer le json. J’utilise XAMPP

Les phase de jeu :

Tous se passe dans le fichier main.js. Le jeu est divisé en phase de jeu allant de 0 à 3. La phase 0 permet de laisser du temps pour que le programme aille récupérer le JSON des niveaux et pour que les obstacles et cibles tombent. Dans cette phase, on ne peut pas infliger de dégâts aux cibles. Elles ne prennent donc pas de falldamage au début. Au bout de 1,2 secondes (temps choisi arbitrairement), on passe en phase 1. Lors de cette phase le joueur peut cliquer avec sa souris sur le canvas. Le projectile va alors bouger en fonction de la position de la souris tout en restant dans un cercle de rayon prédéfinie. Le système de tir est classique, plus on écarte le projectile de son point original et plus il aura de vitesse quand l’utilisateur lâchera le clic de la souris (Lors de cette phase les cibles et obstacles destructibles peuvent enfin être détruits. Lorsqu’on lâche le clic, on passe en phase 2 un court instant. Ce passage en phase 2 permet de laisser un moment avant que l’utilisateur puisse retirer. Au bout de 1 seconde, on passe en phase 3. Dans la phase 3, si l’utilisateur reclique sur la souris, le projectile revient à sa position initiale et on repasse en phase 1.

Organisation du JSON :

Il n’y a qu’un seul JSON. Celui-ci contient un parcours composé de plusieurs niveaux. Dans chaque niveau, on précise :

-le nom

-la largeur

-la hauteur

-l’origine des tirs [x, y]

(Note : pour chacun des objets des tableaux suivants, veuillez préciser l’origine « X » et « Y », « width » (largeur de l’objet) et « height » (sa hauteur))

-un tableau d’obstacles :

(Note : un obstacle est un objet qui peut être « destructible » (bool). Si oui on lui met des « pv » et « pvmax » différents de 0 (« pv » doit être égale à « pvmax »), sinon on les met à 0. On précise aussi la « masse »)

-un tableau de cibles

(Note : une cible est destructible par défaut, il faut cependant lui préciser ses « pv », « pvmax » (« pv » doit être égale à « pvmax ») et sa « masse »)

-un tableau de murs solides

(Note : un mur a une masse infinie, il faut lui préciser « élasticité » (de base mettre 0.5) et s’il est « movable » ([bool, bool]). movable[0] indique si l’objet peut bouger en x, movable[1] en y. Ensuite, il faut mettre des valeurs dans « mouvement » ([int x1, int y1, int x2, int y2]. S’il est non movable, on met des 0. Sinon, on remplit le tableau en sachant que les 2 premières valeurs sont les x et y pour le point extrême haut gauche du mouvement et les 2 dernières sont pour le point extrême bas droite. L’objet se déplacera alors en ligne droite d’un point à l’autre. Il faut également préciser les « directionsDroiteBas » ([bool, bool]). Exemple : si [true, true] l’objet commencera à aller en bas à droite jusqu’à ce qu’il atteigne le point (mouvement [2], mouvement [3]) qui le fera changer de sens.)

Veuillez vous reporter au JSON pour les exemples.

Le code en lui-même :

Les objets sont créés juste après avoir récupéré le json. Lors de leur création, beaucoup d’éléments sont déjà prérentrées de part la nature de l’objet. Par exemple, une cible sera de base destructible et aura automatiquement son attribut « cible » à true, un mur aura de base une masse infinie et le projectile aura son attribut « projectile » à true. Les spécifications des objets viennent des informations entrées dans le json.

Les murs de délimitation son aussi créé sans l’intervention de l’utilisateur. Ils ont l’air de faire simplement le contour du canvas mais en réalité, ils font chacun 100 px d’épaisseur au-delà des limites du canvas pour éviter aux projectiles de les traverser (bug qui était fréquent dans la version originale).

Lorsque « setUpInterval » est lancé, cela marque le début du programme. Celui-ci, couplé aux phases de jeu permettra de gérer les interactions avec le programme ainsi que l’actualisation des événements (rendered.update). La partie update est très similaire à celle du TP2. Pour chaque objet, on va regarder s’il n’est pas en collision avec un autre. Si oui, on les décale pour qu’il ne soit plus confondu et on leur redonne une nouvelle direction. On profite de la fonction update pour pouvoir bouger les murs movables. Ils se déplacent de 1 px à chaque appelle d’update. Ils vont en ligne droite de leur premier point jusqu’au second. Ils peuvent entrer en collision avec d’autre objet de masse infinie cela n’aura pas d’importance (ils se traverseront). En revanche, il ne faut pas que 2 objets soient généré en se chevauchant sous peine de la disparition de ceux-ci.

Lorsque 2 objets entrent en collision on va comparer les masses. Un objet de masse infinie ne bougera pas. Un objet de masse faible sera éjecté. L’élasticité étant réglable à la création des objets (murs dans le json), les propriétés physiques peuvent varier (Exemple : dans le niveau 2, il y a de la boue en marron qui réduit fortement les rebonds, au contraire dans le niveau 3, il y a de la glace/gel qui les augmentes). Dès qu’il y a collision, on réduit à la main la vélocité des 2 objets pour simuler un effet de frottement.

Les points de vie :

Les cibles sont destructibles et les obstacles peuvent l’être. Lorsqu’il y a collision, on va regarder si les objets en collision sont destructibles. Si oui, ils perdront des PV en fonction de la somme des vélocités des deux entités (une des vélocités doit être inversé). Le choc maximal est 2 le minimal est 0.6. Cela qui signifie que si la somme du choc est en dessous de 0.6, il n’y aura pas de dégàts, et s’il est au-dessus de 2, on ramène le choc sur 2 et on inflige des dégats.

Les dégats infligés sont le choc / maxshoc (donc 2) \* degatMax (mis à 3). Un objet ne peut pas prendre plus de 3 dégats. Cependant, Lors d’un choc, plusieurs autres chocs ont lieu à la suite. Les objets prennent donc plusieurs fois des dégâts (par exemple lorsqu’un objet est éjecté rapidement, s’il se retrouve à glisser sur le sol trop vite, il continuera de prendre des dégâts.).