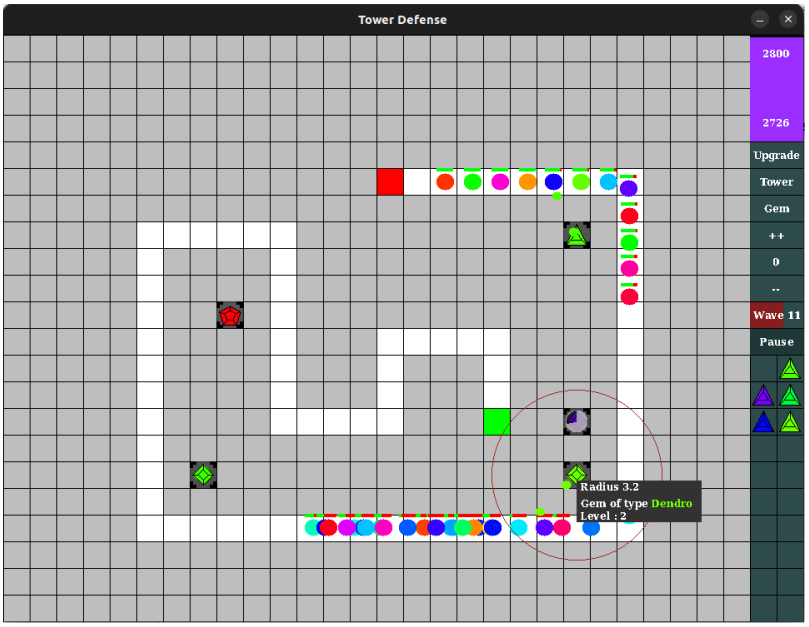
Rapport Projet Programmation C

Tower-defense - KIES Rémy, COSERARIU Alain

L'objectif final du projet est de produire un jeu de type Tower-Defense en langage C en suivant le cahier des charges fourni. Comme cela constitue notre dernier projet de programmation C, l'un des enjeux principaux est de respecter au mieux le paradigme du langage : structuration du projet en module et sous module, et de faire attention aux imports sous-jacents, efficacité du code, robustesse du programme.
Ce rapport détaille les procédés de construction du programme dans une documentation technique, les différents choix effectués dans le projet et la répartition du travail au sein du groupe.



Sommaire

Documentation technique

Cette section du rapport détaille les procédés mis en œuvre pour résoudre les problèmes d'implémentation des éléments de jeu dans le programme. Nous allons surtout nous pencher sur le moteur du jeu plutôt que sur les différents modules d'interface utilisateur. Ceux-ci relèvent d'un plus grand intérêt d'un point de vue algorithmique. Une liste exhaustive est disponible dans une documentation doxygène.

Structures de données

Comme le produit final est un jeu, nous avons décidé de réfléchir en terme d'élément de jeu pour structurer nos données. Par exemple, les monstres ont leur propre module, qui possède aussi la gestion des tableaux de monstres.
Ensuite, si des éléments plus spécifiques se greffent à un élément de jeu déjà existant, nous construisons de nouveaux modules. Reprenons l'exemple des monstres, il est possible qu'il possèdent un effet de status. Donc nous créons un nouveau module contenant les effets et nous ajoutons les interactions dans le module gérant les monstres.

Pour les interactions entre les différents modules du moteur du jeu, la plupart s'effectuent dans le module field. C'est l'endroit le plus cohérent puisqu'il possède toutes les informations sur l'état des éléments de jeu.

Gestion de la mémoire

Avec pour objectif de respecter au mieux les paradigmes de la programmation C, nous avons cherché à avoir un impact sur la mémoire le plus faible possible. Nous avons fait en sorte que la plupart des données soient connues à la compilation du projet, et donc de limiter au maximum les allocations mémoire. Ainsi, l'extrême majorité des différents tableaux sont statiques avec une taille fixe.

Le meilleur exemple pour l'illustrer est le tableau des tours: la taille de la carte étant connue, il y a donc un nombre fini d'emplacement possible pour poser une tour sur le jeu.
Pour le tableau des monstres, nous avons fixé un maximum théoriquement impossible à atteindre. Et s'il s'avère qu'existe une circonstance qui vient à le remplir nous avons décidé de ne pas produire de nouveaux monstres et juste attendre qu'une place se libère dans le tableau.

Pour les projectiles nous avons décidé de procéder à un tableau dynamique qui double de taille si la limite est atteinte. Cependant dans l'optique de limiter le plus possible les manipulations de mémoires nous avons fixé une grande première limite. De même, il est très dur d'atteindre cette limite, mais c'est une précaution qui permet au tours de toujours tirer des projectiles.

Position des éléments

Conformément au sujet, le jeu prend part sur une grille. Le terrain est représenté par un tableau à deux dimensions qui indique la nature de chaque case (camp du joueur, nid du

monstre, emplacement de tour). Ensuite chaque élément déplaçable entre les cases possèdent une position x et y décrite en float. Cela permet par exemple à un projectile qui part d'une tour positionné en (3, 3) d'apparaître au centre de la case (3.5, 3.5) et de se déplacer d'une distance libre.
Cette approche nous permet de garder un niveau d'abstraction entre l'affichage du jeu et son déroulement réel.

Génération du terrain et déplacement des monstres

Les tuiles du chemin des monstres sur le terrain sont placées dans l'ordre du départ du nid jusqu'au camp du joueur. Ceux-ci font office de rails sur lesquels les monstres seront guidés.

Un monstre se déplace de case en case, plus précisément une fois sa destination atteinte il cherchera sa prochaine destination. Pour procéder de la sorte, un monstre garde en mémoire quel est l'indice de la dernière case atteinte (initialisé au nid). Puis cherche la prochaine case sur le chemin des monstres. Pour mettre à jour ses coordonnées, on calcule l'angle entre sa position et sa destination. Puis on calcule la distance parcourue sur la frame, en appliquant la fluctuation et les possibles ralentissement due aux effets.

Sa coordonnée en x vaut alors : cos(angle) * step
et en y : sin(angle) * step

Il ne reste plus qu'à vérifier si sa nouvelle position correspond à sa destination. Comme nous manipulons des flottant, nous avons une marge de comparaison et on regarde si le monstre se situe entre sa destination ± marge.
Cette manière de procéder nous permet surtout d'être plus souple sur le comportement des monstres en cas d'ajout de fonctionnalités. Par exemple, cela permet d'ajouter plutôt simplement les monstres qui se dirigent du nid vers le camp à vol d'oiseau. Il suffit d'initialiser la destination du monstre au camp à la place de la première tuile du chemin.

Gestion du temps

Plusieurs éléments dans le jeu font appel à des mécaniques de temps. Par exemple, le statut parasitaire sur un monstre dure 10 secondes avec un dégât infligé toutes les 0.5 secondes. Un autre exemple est la génération d'un monstre au nid, toutes les secondes (ou 0.5 secondes) un monstre sort du nid.

Pour mesurer le temps dans notre programme, on utilise le concept de frame à la place de mesurer le temps de la machine. On utilise une structure clock qui agit comme un métronome et comme un chronomètre. Comme notre jeu tourne en 60 images par seconde, dans le chronomètre 1 secondes est représenté par un entier, 60 * 1. (note : 60 est définie dans une macro que l'on peut modifier). Ensuite à chaque tour de boucle on décrémente l'entier de 1. Si l'horloge est définie comme métronome, une fois arrivé à -1 on la réinitialise à sa valeur d'origine.

Cela nous permet de maîtriser les événements dans le programme liée au temps de manière plus simple que simplement utiliser le temps système mesuré par l'ordinateur. Un

autre avantage est qu'en cas de ralentissement de l'ordinateur ou de pause du programme, les horloges des éléments ne sont pas affectées puisqu'ils se mettent aussi en pause, et le joueur n'est donc pas pénalisé.

La seule exception à ce mode de fonctionnement est évidemment la boucle principale de jeu. Pour avoir une impression de fluidité et libérer du temps processeur pour les autres programmes, le jeu attend 1/60eme de secondes entre chaque image. Pour se rapprocher le plus possible de ce temps de pause, on mesure une première fois le temps système au début de la frame, puis à la fin on demande une pause de 1/60 - (temps_actuelle - temps_mesuré).

Tableau de monstre

Comme sur le terrain plusieurs monstres bougent en même temps, nous utilisons naturellement une structure de données qui permet de manipuler toutes ces entités. Il n'y a pas de notion d'ordre, donc un simple tableau fera l'affaire. Pour ajouter un monstre on parcourt le tableau en regardant si un monstre est déjà mort, si c'est le cas on écrit notre nouveau monstre sur celui-ci, sinon on l'écrit en fin de liste.
Cette approche n'est pas la plus optimale. Ce que nous aurions dû faire c'est une mise à jour du tableau en fin de frame pour supprimer les monstres décédés et réécrire par dessus les dernier monstre de la liste, comme ça le gain de mana et le gain de score aurait été centralisé.

Inventaire

L'inventaire est affiché comme un tableau à deux dimensions, mais c'est en réalité encore une fois un tableau statique. Pour savoir quelle gemme le joueur peut déplacer, il suffit de prendre les coordonnées de la souris et d'appliquer le calcul suivant pour trouver quel indice du tableau contient la case qui intéresse le joueur: y*2 + x.
Le tableau n'est pas un tableau de gemme mais un tableau d'une structure, qui elle contient la gemme. Cette différence nous permet de faire la distinction entre une case vide et une case qui contient une gemme.

Intéractions

Les intéractions sont toutes gérées par la même structure. Les actions sont déterminées en fonction de l'action courante et des évènements utilisateurs comme le clique ou une touche qui est pressée. On distingue deux types d'actions: les actions prioritaires et non prioritaires. Les actions non prioritaires sont celles que l'on peut écraser et remplacer à tout moment par une autre. Par exemple, si l'on affiche actuellement le prix d'une tour, on veut pouvoir le remplacer par l'action de créer une tour. A l'inverse, les actions non prioritaires ne peuvent pas se remplacer. C'est pour cela que l'on ne peut pas déplacer une gemme et placer une tour en même temps.
Ainsi, comme on ne peut déplacer qu'un seul objet à la fois, la structure gérant les intéractions possède une union sur les types d'objets qui nous intéressent, à savoir les tours, les gemmes, et les petits panneaux d'information. Cette union contient une copie des objets que l'on est en train de manipuler. On connaît donc à tout moment le niveau et le type

de gemme que l'on déplace. C'est d'ailleurs essentiel pour l'afficher correctement lors de son déplacement.

Erreurs

Sur le projet, la gestion des erreurs est complètement décentralisée. Chaque fonction modifiant les données du jeu retournent un code d'erreur indiquant si la modification du jeu a pu (ou non) se faire, et les raisons de l'échec le cas échéant. Cette gestion décentralisée nous permet de savoir exactement ce qu'il se passe dans le jeu. Ces erreurs sont retransmises jusqu'à la partie graphique, qui décide s'il est pertinent ou non de les afficher.

Gestion générale des structures

L'ensemble des structures de données sont réunies au même endroit: la structure *"Game"*. Elle permet de faire la liaison entre les données du jeu et les intéractions de l'utilisateur sur la fenêtre graphique. Ainsi, pour les intéractions, on connaît à n'importe quel moment l'état du jeu pour trouver l'intention du joueur (par exemple, pour combiner deux gemmes, il faut s'assurer que l'action courante est bien un déplacement de gemme, et que l'endroit où le joueur lâche la gemme dans l'inventaire en contient bien une autre
Cependant, ce n'est pas parce que graphique et donnée sont réunis dans une même structure que l'actualisation du jeu est dépendant l'une de l'autre. En effet, la mise à jour des données du jeu (concernant les monstres, les tours, les gemmes et les projectiles) est complètement indépendante de la partie graphique. On peut très bien mettre à jour les données du jeu sans rafraîchir l'affichage (ce qui donnerait un résultat un peu étrange au prochain rafraîchissement, mais qui est pour autant tout à fait possible de mettre en œuvre.

Répartition du travail

Pour mener à bien le projet nous avons séparé les différentes tâches à réaliser de manière à avoir une charge de travail équitable entre les membres du groupe. Cette section détaille comment nous avons procédé pour nous répartir le travail.

Pour avancer dans le projet, nous avons décidé de segmenter notre progression par palier. Le tableau ci-dessous à été construit durant la semaine de vacance du 30/10. Il nous permet de visualiser quelles étapes du projet doivent être terminées pour quand et par conséquent nous donner un ordre d'idée de l'efficacité de notre progression.

Tâche à réaliser	Membre responsable de la tâche	Date fixée
Modularisation du projet, création des structures et mise en place du squelette du projet	-Tout le groupe	30/10/2023
Prototype très simplifié d'un module graphique pour visualiser les premiers éléments de jeu	-Kies Rémy	30/10/2023
Génération d'un chemin reliant nid et camp	-Coserariu Alain	06/10/2023
Déplacement de monstre du nid vers le camp	-Coserariu Alain	06/10/2023
Gestion de structure pour les gemmes, le mana, le joueur, et les interactions entre structure	-Kies Rémy	05/11/2023
Interaction entre les projectiles et les monstres, déplacement des projectiles et application des effets	-Coserariu Alain	12/11/2023
Gestion des évènements, ajout de l'inventaire (déplacement de gemmes, boutons, …)	-Kies Rémy	25/12/2023
Tirs des tours	-Coserariu Alain	06/01/2024
Finalisation de l'interface	-Kies Rémy	08/01/2024
Petites améliorations de l'interfaces (visuel du chargement des tours…)	-Coserariu Alain	14/01/2024
Ajout de "l'arbre de compétence"	-Kies Rémy	14/01/2024
Rendu final (relecture du code + rédaction du rapport)	-Tout le groupe	20/01/2024

Produit final

Cette section détaille tout ce qui à été traité dans le sujet, entre autres les améliorations ainsi que les choix effectués pour améliorer l'expérience de jeu, comme l'équilibrage des constantes de vie des monstres et de dégâts infligés par un projectile.

Cahier des charges

Équilibrage

Pour ce qui est de l'équilibrage des constantes du jeu, nous sommes d'abord parti de la relation entre gain de mana et vie d'un monstre. Comme la formule de perte de mana lors du bannissement d'un monstre est fixe (15% de la vie max du monstre * $1.3^{\text{niveau de la réserve}}$) nous avons fixé en premier la constante de la vie des monstres. On voulait que les premières vagues soient abordées d'un aspect tactique donc le gain de mana est relativement faible au début du jeu. Avec une constante à 100 on se retrouve à pouvoir générer lentement de nouvelles gemmes forçant la mécanique de fusion.
De cette observation nous avons cherché à calibrer les dégâts des projectiles. Pour se faire nous avons procédé par dichotomie jusqu'à avoir un résultat satisfaisant. Notre objectif est de pouvoir tuer les premières vagues avec une seule gemme en la déplaçant sur des endroits stratégiques.

Malheureusement seulement les premières vagues sont possibles à calibrer correctement, à partir d'un certain stade, à cause des relations exponentielles du dégât des projectiles en fonction du niveau, les gemmes deviennent trop fortes. Cela est en réalité un problème principalement à cause de la mécanique de gain de mana en forçant l'apparition d'une vague. Or on ne peut pas limiter le gain de mana en baissant la vie des monstres puisque le début du jeu deviendrait trop frustrant pendant trop longtemps.
Finalement la décision qui à été retenue pour essayer de respecter le cahier des charges au maximum on a uniquement limité le nombre de vagues maximal qui apparaissent à 3.

Nous avons limité le niveau maximal de la mana pool, cela nous évite de dépasser la limite des nombre entiers. Comme on est face à des relations exponentielles, même passer en unsigned long long n'aurait pas été une solution viable. Puis cela permet de limiter la création d'une gemme au niveau 23, forçant le joueur à faire face aux nouvelles vagues uniquement avec des fusions. À ce stade du jeu, les nombres deviennent si grands qu'on change presque de style de jeu en passant du tower defense au clicker. La seule solution viable à partir de la et de créer une nouvelle structure pour gérer des nombre aussi grand que nous le souhaitons. Mais cela dépasse le cadre du projet donc nous nous sommes arrêtés à cette limite de niveau (comptez au moins 15 minutes de jeu pour atteindre cette limite).

Améliorations

Tous les éléments de jeu on été traité et certaines améliorations ont été traité, elles sont listée ici :

- Rayon de détection et vitesse de tir d'une gemme qui augmente avec le niveau de la gemme.

Pour ce faire, nous avons pris comme base les gemmes de niveau 10. Au niveau 10 une gemme tirera exactement une case plus loin qu'une gemme de niveau 1. Et à chaque niveau elle gagne 1/100eme du niveau de la gemme. Comme ça au niveau 10 elle tire 1 projectile toutes les 0.4 secondes à la place de toutes les 0.5 secondes. Ceci représente un bonus qui est visible par le joueur, ce qui crée un sentiment de satisfaction sans être trop puissant.

- Ajout d'un arbre de compétence

Pour réaliser cet ajout, toutes les 6 vagues une interface permet à l'utilisateur de choisir une compétence entre les trois choix possibles affichés. Les compétences sont des éléments qui impactent directement l'état du jeu. Ainsi, il existe 5 types de compétences:

- Donne du mana
- Donne une tour gratuite
- Tue des monstres
- Offre une amélioration gratuite de la jauge de mana
- Donne une gemme pure d'un certain niveau

Les valeurs de chaque compétence sont calibrées en fonction du nombre de vagues et de la jauge de mana. Les compétences non choisies sont proposées de nouveau au prochain choix, avec les mêmes valeurs, et la nouvelle compétence ne peut pas être l'un des deux restantes. Ainsi, le joueur est encouragé à changer de compétence pour avoir des compétences de plus en plus puissantes.

Le reste de nos améliorations tient surtout de la partie graphique pour rendre le programme le plus agréable que possible. Nous avons ajouté à la fois des entrées clavier et des boutons pour interagir avec le programme. Chaque élément de jeu à des indications graphiques, par exemple le clique droit de la souris permet d'obtenir des informations sur les tours ou les gemmes. Augmenter le niveau de la mana pool, ajouter une tour sur le terrain ou générer une gemme est liée à un coût qui est indiqué sur la jauge de mana. Si le coût est trop grand l'indicateur vire au rouge.
Un autre exemple est le temps de chargement des gemmes, lorsqu'une gemme est placée dans une tour un cercle s'affiche pour indiquer combien de temps il reste avant le déploiement effectif de la gemme. Il n'est pas possible de dessiner des parties de cercles avec la librairie MLV, on dessine simplement un polygone régulier avec 100 points pour simuler une partie de cercle. Un dernier exemple est l'indicateur de temps sur le bouton de déclenchement d'une nouvelle vague, celui-ci décroît pour indiquer combien de temps reste-t-il avant la nouvelle vague
Donc plutôt qu'ajouter des nouvelles fonctionnalitées nous avons préféré rendre le jeu le plus agréable possible

Manuel utilisateur

Compilation

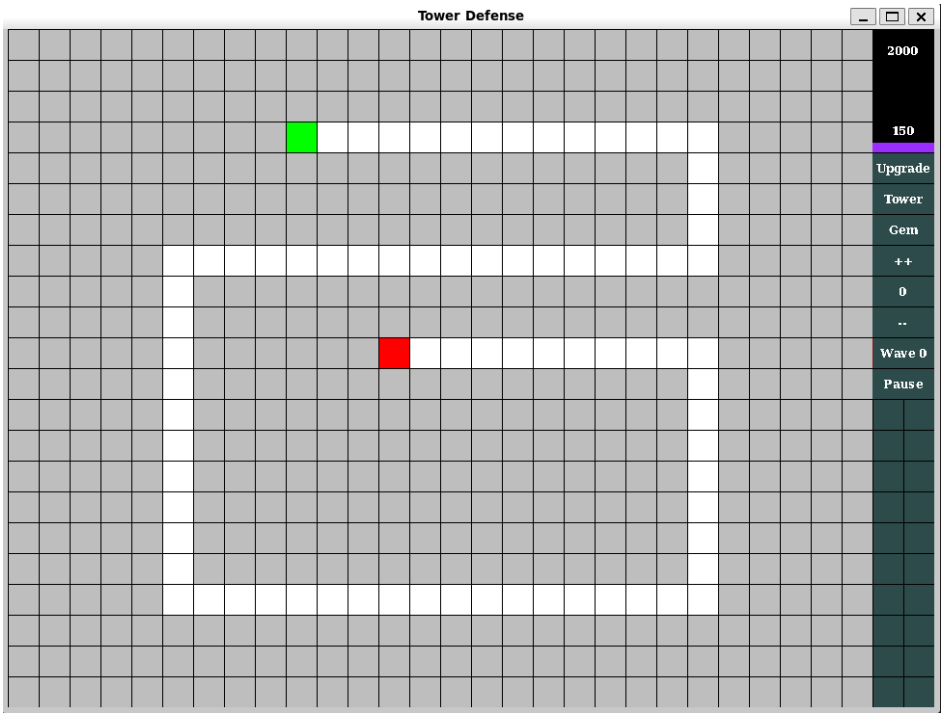
Le projet se compile avec la commande Unix “*make*”. A l’issue de cette commande, l’utilisateur possède dans le dossier courant au makefile l’exécutable “*GemCraft”* qui est le point d’entrée du programme, ainsi qu’un dossier “*bin*” contenant l’ensemble des fichiers binaires nécessaires à la création de l’exécutable “*GemCraft*”.
Le programme à été compilé et testé sous Ubuntu 22.04 et 20.04, ainsi que sur les distributions Debian disponible sur les machines de l’université.

La documentation doxygen peut être créée avec la commande *“make doc”*. Une fois terminée, elle se trouve dans le dossier “doc/doxygen”. Le fichier d’entrée de la documentation est le fichier “doc/doxygen/hml/index.html”.

Exécution

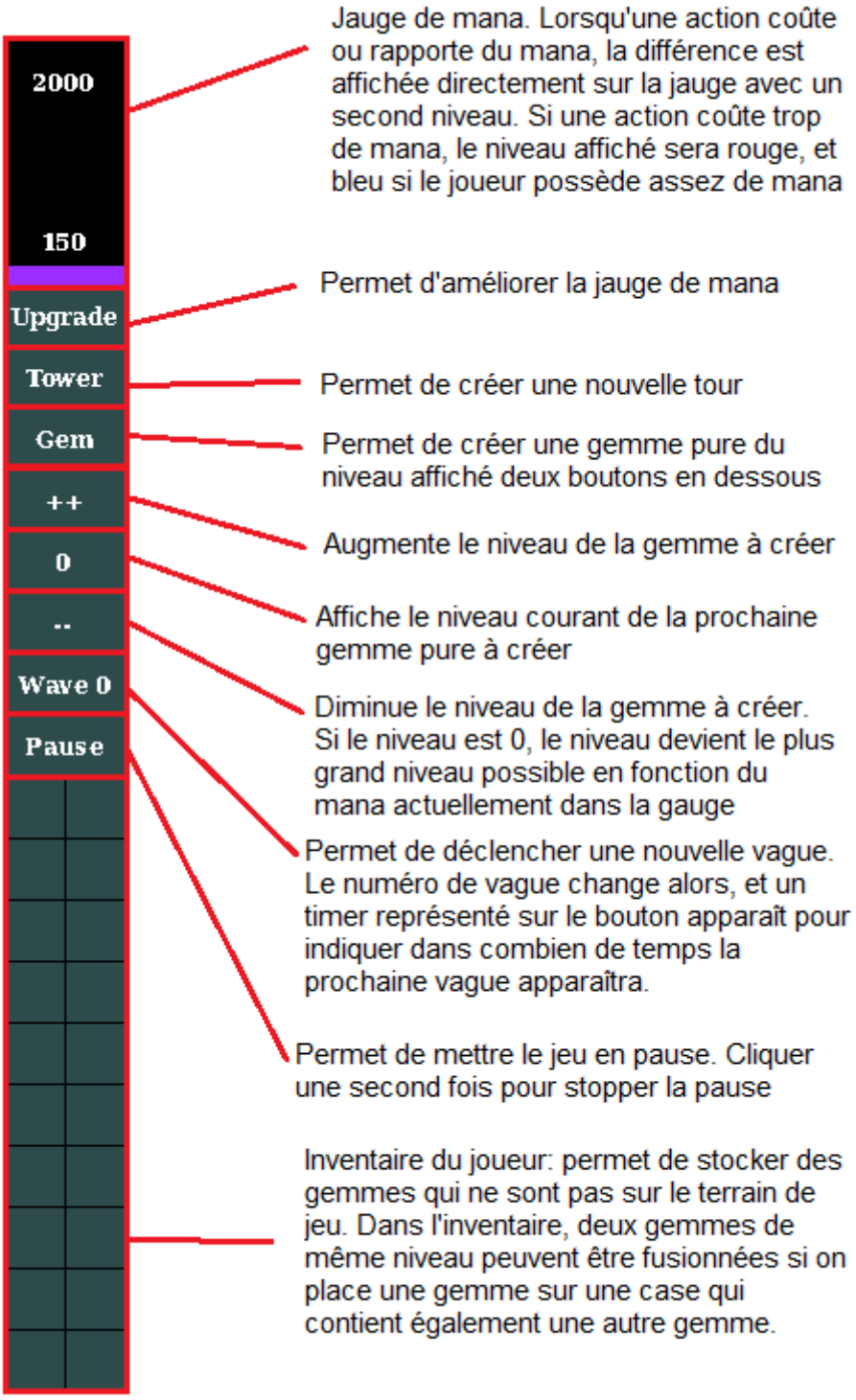
Pour lancer l’exécutable “*GemCraft*”, aucune option en ligne de commande n’est requise. Le programme s’exécute avec la commande “./*GemCraft*”. Une fenêtre graphique apparaît. Plus rien n’est à saisir dans le terminal, toutes les interactions avec le programme passent maintenant au via l’interface en utilisant la souris. Des raccourcis claviers existent et sont détaillés dans la section suivante.

Exemple d'affichage de l'interface après le lancement du programme



Contrôles clavier/souris

Sur la droite du jeu se trouve le panel utilisateur du jeu. Les boutons ou interfaces présents ici sont accessibles via le clique gauche.

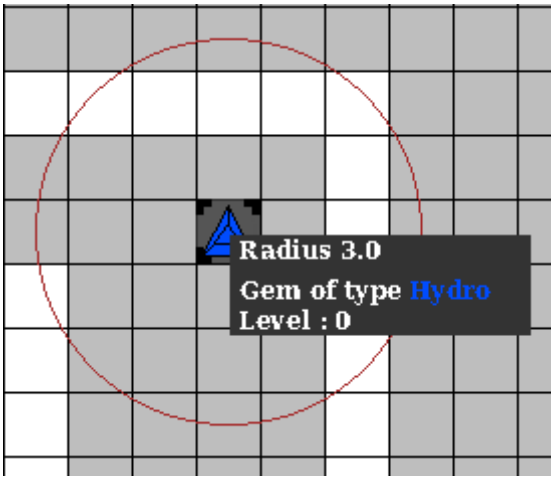


Le clique droit permet quant à lui d'annuler le déplacement d'un élément comme une gemme ou une tour, mais aussi d'avoir des informations supplémentaires sur ces derniers.

La plupart des actions sont également accessible via des raccourcis claviers :

- “U”: améliore la jauge de mana
- “T”: crée une nouvelle tour
- “G”: crée une nouvelle gemme du niveau indiqué
- “+”: augmente le niveau de la prochaine gemme à générer
- “-”: diminue le niveau de la prochaine gemme à générer
- “W”: déclenche une nouvelle vague
- “P”: met le jeu en pause

Une petite précision est nécessaire concernant le “+” et “-” : les touches du pavé numérique ne sont pas les mêmes du clavier de base, dans le sens ou le code ASCII renvoyé par MLV est différent des touches de saisie. Au pavé numérique, nous avons fait du mieux possible pour les gérer, mais nous ne savons pas si selon les claviers, le code renvoyé peut changer. Par conséquent, il est possible que selon les machines, les touches “+” et “-” du pavé numérique ne fonctionnent pas.



Exemple d’affichage d’informations supplémentaires avec le clique droit sur une tour

La forme des gemmes évolue au fur et à mesure que leur niveau augmente. Ainsi, les gemmes passent successivement d’un triangle à un carré, puis un pentagone et enfin un hexagone. Les couleurs quant à elles évoluent lors de combinaisons de gemmes.

Conclusion

Ce projet est le premier jeu que nous devons concevoir dans le langage C. Comme ce langage n’est pas adapté pour ce genre de projet, le plus important fût l’organisation au sein du groupe pour avancer sur plusieurs axes afin d’être le plus efficace possible. La tâche la plus délicate a été la structuration, mais après concertation nous somme parti sur une base solide qui nous à permis d’économiser suffisamment de temps pour mener à bien le projet. Comme le sujet ne détaille pas les structures de données à utiliser, nous avons pris les structures les plus adaptées à notre besoin. Avec une structure solide, l’ajout de fonctionnalité s'incorpore très bien au code déjà existant. Nous n’avons pas utilisé de concept particulièrement avancé spécifique au langage, comme les liste chaînée, pour essayer de respecter au plus l’idée de d’efficacité inclus dans le paradigme du langage.

Malgré l’équilibrage limité par le cahier des charges, nous pensons avoir réussi à réaliser un jeu agréable à jouer.