# Projet : Tower Defense

---

- Projet : Tower Defense
  - Lancement du jeu
    - Compilation
    - Nettoyage
    - Lancement du programme
    - Les paramètres
  - Modularisation du code
    - Bin
    - Doc
    - Include & Source
    - Resources
    - Autres fichiers
  - Déroulement du projet
    - Implémentation par rapport au sujet
      - Vagues
      - Projectile
      - Element
      - Mana / Gem
      - Makefile
    - Amélioration
      - Drag and Drop
  - Conclusion
    - Améliorations possibles
      - Trap
      - Vague amélioré
      - Element amélioré
    - Avis sur le projet

## Lancement du jeu

### Compilation

Pour lancer le projet, vous devez décompresser l'archive et compiler avec la commande `make`.

```
unzip prog-chapelain_laborde.zip
make
```

Si vous dézipez l'archive à la main, pensez à aller dans le dossier crée avant d'effectuer la commande `make`.

```
# dézipage à la main
cd prog-chapelain_laborde
make
```

### Nettoyage

Pour nettoyer le projet une fois l'utilisation terminée vous pouvez utiliser les commandes :

- `make distclean` : pour supprimer les fichiers objets.
- `make clean` : pour supprimer les fichiers objets et l'exécutable.

### Lancement du programme

Pour lancer le programme compilé vous devez utiliser la commande suivante :

`./gemcraft`

### Les paramètres

Prototype des paramètres de la commande.

./gemcraft [-w **WIDTH**x**HEIGHT**] [-f] [-d] [-h]

- `-w`, `--window` (facultatif) : Change la taille de la fenêtre. (doit être au minimum de taille 720x480)
- `-f`, `--full-screen` (facultatif) : Lance le jeu en plein écran.
- `-d`, `--difficult-mode` (facultatif) : Active le mode difficile. (passer une vague ne donne pas de mana)
- `-h`, `--help` (facultatif) : Affiche le menu d'aide.

## Modularisation du code

Nous avons séparé notre code en plusieurs dossiers et fichiers afin de rendre le code plus lisible et plus facile à maintenir. Voici l'arborescence abstraite du projet :

- bin
- doc
- include
  - events
  - graphic
  - tools
- logdev
- Makefile
- README.md
- resources
  - font
  - images
  - metadata.tex
- src
  - events
  - graphic
  - tools

### Bin

Contient les fichiers objets générés par la compilation du programme. Ils sont stockés dans ce dossier afin de ne pas polluer le dossier principal.
Vous trouverez aussi dans ce dossier les fichiers de dépendances générés par le flag `-MMD` du compilateur. Ils permettent de recompiler automatiquement les fichiers qui ont été modifiés. Ce flag permet de ne pas avoir à écrire à la main les dépendances de chaque fichier.
Vous verrez aussi que dans ce dossier, l'arborescence des dossiers `include` et `src` est reproduite. Cela permet de garder une arborescence propre et de ne pas avoir à modifier les chemins d'accès aux fichiers objets dans le Makefile.

### Doc

Dans ce dossier, vous trouverez la documentation technique du projet ainsi que ce rapport.
Vous pourrez aussi trouver dans le sous-dossier `html` la documentation générée par Doxygen.
Pour que tel soit le cas, je vous invite à utiliser la commande `make doxygen`. (Vous devez avoir Doxygen d'installé sur votre machine)

### Include & Source

Dans ces deux dossiers, vous trouverez respectivement les fichiers d'en-tête et les fichiers sources du programme.
Dans ces dossiers, vous trouverez aussi des sous-dossiers qui permettent de séparer les fichiers en fonction de leur utilité.
Voici une description de ces sous-dossiers :

- **events** : Contient toute la gestion des événements avec un module permettant de stocker les events de la libMLV de manière plus simple. Cela permet aussi d'éviter d'avoir des fonctions de récupération d'événement en vrac dans la boucle principale. D'autres modules permettant la gestion d'interactions entre les événement joueur et le code. Enfin, nous avons un module entier dédié à la partie `DragAndDrop` permettant de faire glisser visuellement les objets à l'aide de la souris.
- **graphic** : Possède tout ce qui est lié à l'affichage graphique. Séparé en différents modules qui gèrent une grande partie de l'affichage. L'affichage général se situe dans le module `graphic` et il y a aussi un module `window` permettant de séparer les différentes parties de l'interface graphique pour pouvoir différencier la carte de l'inventaire sur la droite.
- **tools** : Ce dernier est utilisé pour stocker des modules utilitaires comme le `DynamicArray` qui est un tableau de stockage multi-type utilisé pour le stockage des données de la carte. Ou encore, le module `Utils` contentant des fonctions pratiques tel qu'une structure de coordonnées ou bien de directions. Le module `TimeManager` permet de gérer les usages de la structure `timespec` de `<time.h>`. Cette fonction est utilisée, entre autres choses, pour gérer les timers des vagues et des projectiles.
- Enfin, il nous reste les documents "en vrac" le dossier src. Il représente le squelette du projet comme les vagues, la carte, l'inventaire, la gestion du mana par exemple. Tous ces modules sont utilisés par la module game stockant la boucle principale du jeu. Il y a aussi le module `main` qui est le point d'entrée du programme. Il permet de lancer le jeu et de gérer les paramètres de la ligne de commande.

### Resources

Contient les fichiers de ressources utilisés par le programme. Dans notre cas, il s'agit uniquement de la police d'écriture utilisée pour le menu. Elle est stockée dans le dossier `resources/fonts`.
Nous avons aussi adapté nos fonctions pour pouvoir utiliser des images à la place par exemple des dessins pour les tours ou les monstres. Nous avons cependant préféré ne pas les utiliser pour une question de facilité de développement.
Vous avez aussi peut-être remarqué le fichier `metadata.tex` et le dossier `images`. Ils sont tous deux utilisés dans le Makefile pour générer ce rapport au format PDF via pandoc.

### Autres fichiers

- **Doxyfile** : Fichier de configuration de Doxygen. Il permet de générer la documentation API du projet.
- **logdev** : Fichier dump des logs de développement. Il permet de garder une trace des modifications effectuées sur le projet. Vous pouvez aussi retrouver les logs de développement sur le dépôt Git du projet.
- **Makefile** : Fichier de compilation du projet. Il permet de compiler le projet, de nettoyer les fichiers objets et l'exécutable. Il permet aussi de générer la documentation Doxygen et de générer ce rapport au format PDF via pandoc.
  Vous avez un descriptif plus haut dans le rapport sur l'utilisation de ce fichier.

## Déroulement du projet

Ce projet a été réalisé en plusieurs étapes.
Nous avons commencé par faire une grande liste résumant les fonctionnalités à implémenter. Vous verrez donc dans les premiers commit un fichier `TODO.md` qui résumait une très grande partie de ce qu'il fallait faire.
Nous avons ensuite commencé par modulariser le projet, c'est à dire séparer le code en plusieurs fichiers et dossiers. Vous verrez ainsi, dans un commit, la création de l'arborescence du projet.
Tout cela a été réalisé ensemble en pair-programming. Cela nous a permis d'avoir une base solide pour commencer le projet et nous a permis de nous mettre d'accord sur la manière de faire.
Nous avons ensuite pu nous séparer pour travailler sur des parties différentes du projet.

Nous avons commencé par la base du projet, c'est à dire la carte et le mana qui sont les deux éléments essentiels du jeu.
Nous avons ensuite créé une première version de la génération du terrain.
Ensuite, nous avons entamé la partie graphique du projet afin de pouvoir afficher la carte et le chemin.
Puis, de fil en aiguille, nous avons implémenté les différentes fonctionnalités du jeu.
Nous avons fait la partie déplacement des monstres, la gestion des vagues, la gestion des tours, la gestion des projectiles, etc…
Nous avons aussi implémenté des fonctionnalités supplémentaires comme le drag and drop des gemmes ou encore l'inventaire graphique sur la droite de l'écran.

Vous pouvez retrouver l'historique du projet sur le dépôt Git du projet ou dans le fichier `logdev`.

### Implémentation par rapport au sujet

**Vagues** Les déplacements des vagues ont été implémenté dans leurs intégralités.
Les timers sont réglés comme souhaité dans l'énoncé.
La touche espace permet bien de déclencher la première vague et le passé à la suivante.
Cependant nous avons interprété le sujet de la manière suivante : il ne peut pas y avoir plus d'une vague en même temps. Cela est aussi pratique pour équilibrer le jeu car le fait de gagner du mana en sautant les vagues est vite trop fort avec la calibaration du sujet. Suite à ça, nous avons, comme vous l'avez peut être remarqué, implémenté un mode difficile.
Ce mode est activable via la ligne de commande avec le flag `-d` ou `--difficult-mode`, et permet de ne pas avoir de gain de mana quand nous coupons le temps pour faire apparaître la prochaine vague.

**Projectile** Vous verrez, en jouant au jeu, que les projectiles ont été implémentés comme demandé dans le sujet.
Ils suivent leurs cibles autant qu'ils le peuvent et disparaissent quand ils touchent leur cible.
Ils sont aussi détruits quand les monstres atteignent la fin du chemin et sont bannis de la carte.
Cependant dans le code le monstre avance en fonction du framerate, le projectile doit donc faire pareil donc la formule à été changé pour mettre un produit en croix afin de pouvoir calibrer la vitesse à partir d'une constante.

**Element** La partie sur les éléments quant à elle été implémenter totalement la gestion des timer est peut être différente de celle souhaité.
Les éléments appliquent les effets avec les constantes voulues de l'enoncé. Cependant la gestion des timer à été réalisé de la manière suivante : quand un élément hydro frappe le monstre on déclenche le timer de ralentissement et ensuite si un projectile pyro le touche le monstre le timer disparaît pour appliquer l'effet de vaporisation. Et pendant que le monstre est en état d'enracinement le monstre je peux pas subir d'autre effet ormi la propagation de la vaporisation de la part de montres voisins.

**Mana / Gem** Pour le mana et le gem l'unité de stockage étant l'int nous avons fait en sorte de bloquer les améliorations à un certain niveau histoire de ne pas faire de dépassement. Par exemple, le niveau des gemmes est limité à 24 mais nous pouvons toujours les fusionner pour faire des gemmes de plus haut niveau.

**Makefile** Comme vous l'avez peut être remarqué, nous avons utilisé un Makefile pour compiler le projet.
Un Makefile est un fichier très utilisé en C pour automatiser la compilation d'un projet et toute commande répétitive.
Vous pouvez retrouver une description de ce fichier plus haut dans le rapport.

Le Makefile a été réalisé très rapidement dans le projet.
Il nous a permis d'avoir une compilation plus rapide et plus simple.
Nous avons aussi comme demandé dans le sujet, créer des dépendances pour les fichiers objets.
Cela permet de recompiler correctement les fichiers qui ont été modifiés, même si ce n'est pas le fichier source qui a été modifié.
Cependant, nous avons préféré ne pas les écrire à la main et nous avons trouvé le flag `-MMD` du compilateur qui permet de générer ces dépendances automatiquement.
Il suffit aussi de dire à notre Makefile d'inclure ces fichiers de dépendances pour que tout fonctionne correctement.

Nous avons aussi rajouté une règle pour générer la documentation Doxygen et une autre pour générer ce rapport au format PDF via pandoc. (Vous devez avoir Doxygen et pandoc d'installé sur votre machine)
Vous pourrez donc utiliser les commandes `make doxygen` et `make rapport` pour générer ces documents.

### Amélioration

**Drag and Drop** Comme précédemment dit, nous avons implémenté le drag and drop des gemmes.
Celui-ci permet de faire glisser des gemmes dans tous les sens.
Cependant, un bug réside dans le fait que notre code utilise la fonction `MLV_delay_according_to_frame_rate`.
Celle-ci permet d'avoir un framerate constant mais elle a pour effet de ralentir la détection des événements.
Ainsi, si vous faites glisser une gemme trop vite, le clic de relâchement ne sera pas détecté et la gemme restera bloquée.
Pour faire fasse à ce problème, qui au final n'est pas très gênant, il vous suffit de cliquer à nouveau sur le bouton gauche de la souris pour débloquer la gemme.

## Conclusion

### Améliorations possibles

**Trap** Nous aurions voulu implémenter des pièges sur la carte comme l'amélioration nous le permettait.
Vous verrez d'ailleurs dans nos précédents commits que notre code est prévu pour pouvoir implémenter cette fonctionnalité.
Cependant, nous avons préféré nous concentrer sur d'autres fonctionnalités plus importantes pour le jeu.
Nous avons donc laissé cette fonctionnalité de côté mais elle pourrait être implémentée facilement.

**Vague amélioré** Si nous repensons la structure du module `Wave` nous pouvons faire un sorte d'avoir plusieurs structures de spawn et un seul tableau de monstre afin de pouvoir gérer plusieurs timer en simultané avec différent type de monstre en simultané, ce que notre structuration actuelle ne permet pas.

**Element amélioré** Pour les éléments améliorer nous pourrons faire en sorte de pouvoir cumlumer les timers des effets c'est-à-dire que le monstre pourrait avoir des résidus élémentaires.

**Avis sur le projet**

- Sujet complet et intéressant.
- Implémentation de l'algorithme de la génération de chemin.
- Tower défense basé sur un vrai jeu.
- Le fait d'appuyer sur espace nous fait gagner trop de mana rendant le jeu trop simple, car le calcul est fait par rapport à la barre de mana directement et gagner jusqu'à 20% "gratuitement".