Le Mans Université

Institut d'Informatique Claude Chappe
Le Mans Université

Le Mans Université

Licence Informatique 2e année

Module 174UP02 Rapport de Projet

TapVenture

Noam MANFALOTI, Ibrahim SAPIEV, Lucas DUPONT et Logan Evenisse

9 avril 2025

Lien du projet

Table des matières

1 Introduction **3**

2 Analyse et conception **3**
2.1 Description du projet 3
2.2 Fonctionnalités 3
2.3 Direction artistique 4

3 Gestion du projet **5**
3.1 Planification 5
3.2 Organisation du travail 6
3.3 Outils de travail 6

4 Realisation **7**
4.1 Gestion des textes et de la traduction 7
4.2 Gestion de l'interface utilisateur 7
4.2.1 Structure de données 7
4.2.2 Fonctions de gestion de l'interface 8
4.3 Gestion de l'inventaire et du scroll 9
4.3.1 Implémentation du système de scroll 9
4.3.2 Gestion de l'inventaire 10
4.3.3 Intégration et synchronisation des modules 11
4.4 Gestion des sauvegardes 11
4.4.1 Type de fichier 11
4.4.2 Repartition des données 12
4.4.3 Structure des fichiers 12
4.4.4 Fonctions de sauvegarde 12
4.5 Système de combat et de challenge 13
4.5.1 Structure de données 13
4.5.2 Progression des monstres 13
4.5.3 Animation de dégâts 14
4.5.4 Fonctions du système de combat 14
4.5.5 Système de challenge 14
4.6 Les héros et jeux hors ligne 15
4.6.1 Structure des héros 15
4.6.2 Fonctions des héros 16
4.6.3 Gestion du jeu hors ligne 16
4.7 Système de prestige 17
4.7.1 Structure de données 17
4.7.2 Fonction du système de prestige 18

5 Conclusion **18**

6 Annexe **20**

1 Introduction

L'objet de ce document est de présenter le projet de développement du jeu TapVenture réalisé dans le cadre du module Conduite de Projet L2 Informatique de Le Mans Université sur une période de janvier à avril 2025. Le jeu TapVenture a été réalisé en langage C et avec la bibliothèque graphique SDL.
Le jeu TapVenture se base sur un concept de clic (catégorie de jeu de type Clicker). Le joueur doit cliquer sur des monstres pour gagner de l'or et améliorer les dégâts afin de gagner plus d'or. Ensuite, le joueur peut acheter des héros qui vont combattre les monstres automatiquement. Et lorsqu'il atteint le dernier niveau, il «prestige», c'est-à-dire qu'il recommence le jeu mais avec des points à dépenser dans un arbre de compétences.
Nous verrons dans une première partie la présentation du jeu et de ses principales fonctionnalités. Les éléments de gestion de projet feront l'objet d'une deuxième partie. Dans une troisième partie, nous décrirons les étapes de conception et de réalisation avant de conclure sur les aspects positifs et les limites de notre projet.

2 Analyse et conception

2.1 Description du projet

Le joueur commence l'aventure en cliquant sur les monstres pour les vaincre et récupérer leur or. Cet or sert ensuite à recruter de nouveaux héros qui attaquent à la place du joueur et à augmenter les dégâts que le joueur et ses héros infligent, permettant ainsi une progression exponentielle de la puissance de l'utilisateur. Le gameplay repose sur une boucle addictive d'accomplissement de tâches, de gain d'or, et d'amélioration de ses moyens d'attaque. La progression se déroule à travers plusieurs niveaux, chacun situé dans un univers aux défis et aux monstres croissants. Pour passer au niveau suivant, le joueur doit vaincre 10 fois les monstres de l'étape. Tous les 5 niveaux, un boss apparaît et doit être vaincu dans un temps imparti. À l'issue du dernier monde, le joueur «prestige». Enfin, TapVenture intègre des fonctionnalités complémentaires telles qu'un système multilingue, des challenge, ainsi qu'un système de récompense hors ligne. La gestion de la sauvegarde permet de reprendre le jeu quand on le souhaite, pour une expérience utilisateur optimale.

2.2 Fonctionnalités

Notre projet, TapVenture, intègre plusieurs fonctionnalités clés. Voici les fonctionnalités développées :

— **Système de combat** : Un mécanisme interactif permettant aux joueurs d'affronter des ennemis et de progresser dans le jeu.
— **Système de progression** : Les joueurs peuvent améliorer leurs dégâts et leurs héros au fil du temps, offrant une expérience de jeu évolutive, Lorsqu'ils atteignent le dernier niveau, ils doivent vaincre le boss final pour effectuer un prestige.
— **Système d' or** : Une monnaie virtuelle qui est utilisée pour acheter des améliorations et des héros dans le jeu.
— **Système de magasin** : Une boutique qui permet aux joueurs d'acquérir de nouveaux héros et d'augmenter ses dégâts.
— **Système de défis** : Des défis disponibles toutes les 30 minutes pour gagner des récompenses supplémentaires.
— **Système de langue** : Le jeu est multilingue, permettant aux utilisateurs de choisir leur langue préférée.
— **Arbre de compétences (prestige)** : Un système d'arbre de compétences permet aux joueurs de débloquer des améliorations passives en échange de points de prestige. Il est composé de 3 branches, chacune ayant un thème spécifique. La branche d'or augmente les gains d'or du joueur, la branche de dégâts augmente les dégâts infligés, et la branche de prestige augmente les points de prestige gagnés par le joueur, ainsi que les héros gardés lors du prestige.
— **Système de sauvegarde** : Les progrès des joueurs sont sauvegardés automatiquement, garantissant une continuité de l'expérience de jeu.
— **Système de récompenses hors ligne** : Les joueurs reçoivent des récompenses même lorsqu'ils ne sont pas connectés, favorisant une progression continue.
— **Système d'inventaire et de scroll** : Un système d'inventaire qui permet aux joueurs de gérer leurs objets et de faire défiler les éléments de manière fluide.
— **Système de notification** : Un système de notification qui informe les joueurs des événements importants, tels que la fin d'un défi ou l'arrivée d'un nouveau monstre.

2.3 Direction artistique

La direction artistique de TapVenture s'inspire d'un style fantastique moderne, combinant des éléments visuels immersifs et dynamiques. Afin de pouvoir offrir aux utilisateurs une expérience de jeu engageante et attrayante, nous avons opté pour des graphismes colorés, avec des animations pour rendre le jeu plus vivant et interactif.

3 Gestion du projet

Cette partie présente la planification du projet, l'organisation du travail au début et tout au long des séances, ainsi que les outils de travail utilisés pour assurer une collaboration efficace.

3.1 Planification

Au début du projet, nous avons établi un diagramme de Gantt prévisionnel [1] pour planifier les différentes étapes du développement. Ce diagramme nous a permis de :

- Définir et répartir les tâches sur la période de janvier à avril.
- Définir des jalons importants, tels que la fin de la conception, le début des tests et la livraison finale.
- Identifier les dépendances entre les tâches pour éviter les blocages.
- Prioriser les étapes critiques afin de garantir une progression fluide.

Le diagramme de Gantt prévisionnel a été conçu pour refléter les grandes étapes suivantes :

1. **Initialisation** :
   - Formation à SDL.
   - Mise en place de l'environnement technique.
   - Réflexion sur l'optimisation du jeu.
   - Analyse de la mécanique de jeu.
   - Création du planning global du projet.
   - Préparation de l'environnement de développement.
2. **Conception** :
   - Conception de l'UI.
   - Choix du style graphique.
   - Définition de la progression du jeu.
   - Définition des mécaniques de jeu.
3. **Codage & Test** :
   - Développement des mécaniques de base.
   - Création des premiers assets graphiques.
   - Programmation du gameplay.
   - Tests internes pour identifier les bugs et vérifier la stabilité.
   - Ajustements après tests et optimisation des performances.
4. **Finalisation** :
   - Corrections finales des bugs.
   - Optimisation du code et des assets graphiques.
   - Préparation de la version finale pour livraison.
5. **Livraison** :

1. Voir le Gantt prévisionnel : Gantt prévisionnel

— Rédaction du rapport.
— Préparation de la soutenance.

3.2 Organisation du travail

Pour organiser le travail, nous avons adopté une approche collaborative et structurée :

— **Répartition des tâches** : Des tâches précise ont été attribuées en fonction des compétences et des préférences de chaque membre de l'équipe. Par exemple, certains se sont concentrés sur le développement des fonctionnalités principales, tandis que d'autres ont travaillé sur l'interface utilisateur ou les tests.
— **Réunions régulières** : Chaque séance de travail débutait par une réunion rapide pour faire le point sur l'avancement, discuter des éventuels problèmes rencontrés et ajuster les priorités si nécessaire.
— **Mise à jour du planning** : Le diagramme de Gantt et le tableau Kanban ont été mis à jour régulièrement pour refléter l'état réel du projet. Cela nous a permis de :
  — Comparer le planning initial avec l'avancement réel.
  — Identifier les écarts et ajuster les priorités en conséquence.
  — Maintenir une vision claire des étapes restantes.

3.3 Outils de travail

Pour faciliter la collaboration et assurer une bonne coordination, nous avons utilisé plusieurs outils adaptés à nos besoins :

— **GitHub** : Utilisé pour le versionnement du code, le suivi des modifications. Cela a permis à chaque membre de travailler sur des fonctionnalités spécifiques et de voir les changement et mise a jour effectuer.
— **Discord** : Utilisé pour la communication en temps réel, les réunions à distance et le partage rapide d'informations. Cet outil a été particulièrement utile pour maintenir une bonne communication, même en dehors des séances de travail.
— **Outil Kanban** : Un tableau Kanban dans le git a été utilisé pour suivre l'avancement des tâches. Les colonnes représentaient les différentes étapes : todo (ce qui est a faire), in progress (le travail en cours), Done séance X (ce qui a été fait a la X^e séance), et les cartes correspondaient aux tâches spécifiques. Cela nous a permis de visualiser facilement l'état d'avancement du projet.

Ces outils ont joué un rôle clé dans la réussite du projet en nous permettant de travailler efficacement, même à distance, et de maintenir une bonne coordination au sein de l'équipe.

4 Realisation

Cette partie présente les différentes étapes de la réalisation du projet TapVenture, ainsi que les choix techniques effectués.

4.1 Gestion des textes et de la traduction

Le fichier `chaine.c` contient plusieurs fonctions dédiées à la manipulation dynamique des chaînes de caractères, permettant de construire, formater et traduire des textes de manière flexible. La fonction `formatChaine` reprend le principe de la fonction `printf`. Son objectif n'est pas de produire un affichage mais de créer une chaîne de caractère et d'allouer le bon espace mémoire. Les paramètres de la fonction sont les même que `printf`, a l'exception que les seules types acceptés sont ceux que nous utilisons `int`, `float`, `string` et nos types personnalisée :

- `%w` : pour afficher un nombre de type unsigned long long int avec un format compact.
- `%t` : pour insérer une traduction via la fonction Traduction

Dans TapVenture, les nombres peuvent devenir extrêmement grands en fin de partie, c'est pourquoi nous utilisons des unsigned long long int. Mais leur affichage à l'écran est difficilement lisible pour le joueur au-delà de 4 chiffres. Pour résoudre ce problème, le format personnalisé `%w` est utilisé, géré par la fonction `formatULLI`. Cette fonction prend en paramètre un unsigned long long int et renvoie une chaîne de caractères allouée dynamiquement, représentant le nombre dans un format plus compact. Par exemple, le nombre 1234 est transformé en 123A, a chaque dizaine supplémentaire la lettre de l'alphabet est incrémentée. Ce format d'affichage améliore considérablement la lisibilité des nombres pour le joueur.
La fonction `Traduction` présente dans le fichier `lang.c` prend en paramètre un enum `cleMsg` qui permet de représenter l'indice du texte dans un tableau de chaîne de caractère. Chaque langue a son propre tableau contenant toutes ses traductions. La fonction renvoie la chaîne dans le tableau de la langue actuelle.

4.2 Gestion de l'interface utilisateur

Cette partie détaille la gestion de l'interface utilisateur, qui comporte les boutons, les textes, les images et les notifications.

4.2.1 Structure de données

L'interface utilisateur est organisée autour de plusieurs structures clés :

— **uiPage** : Représente une page complète de l'interface, contenant un conteneur pour les éléments d'interface, une liste de boutons et une liste de buttonImg.
— **pageHolder** : Structure globale qui stocke toutes les pages du jeu, permettant de naviguer entre elles.
— **uiContainer** : Contient les éléments d'interface d'une page (textes, images).
— **uiTxt** : Élément textuel de l'interface, caractérisé par son contenu, sa police, sa couleur et sa position.
— **uiImg** : Élément graphique de l'interface, défini par son chemin d'image, sa position et son identificateur.
— **Button** : Élément interactif permettant aux joueurs d'effectuer des actions, caractérisé par son texte, sa position, son apparence et la fonction qu'il déclenche lors d'un clic.
— **ButtonImg** : Un élément avec les mêmes caractéristiques que Button, mais qui utilise une image comme icône à la place du texte.
— **Notif** : Une notification qui s'affiche sur l'écran pendant un temps défini, elle est caractérisée par son titre, sa description, sa position, sa taille et son image de fond.
— **NotifList** : Une liste de notifications, qui contient un tableau de Notif et un compteur pour suivre le nombre de notifications affichées.

Dans chaque élément d'interface lié à SDL, nous avons sauvegardé les textures des éléments afin de ne pas les recréer a chaque fois. cette texture est donc détruite et recréer lors d'un changement d'apparence sur un bouton (sauf s'il s'agit d'un changement de texte).

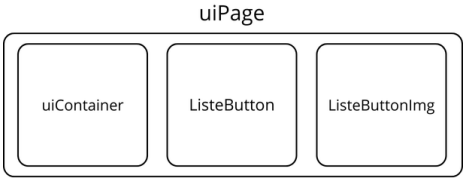

FIGURE 1 – Structure d'uiPage

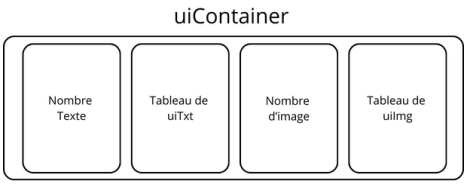

FIGURE 2 – Structure d'uiContainer

4.2.2 Fonctions de gestion de l'interface

— `createPage` : Initialise une nouvelle page en allouant la mémoire nécessaire pour son conteneur et en initialisant ses propriétés. La fonction prend en paramètre un pointeur vers la page à créer et ne renvoie rien.
— `destroyPage` : Libère la mémoire utilisée par une page, y compris tous ses éléments d'interface. Elle prend en paramètre un pointeur vers la page à détruire et ne renvoie rien.

— `createUIText` : Crée un élément textuel et l'ajoute à une page spécifique. Elle prend en paramètre la page cible, la police à utiliser, le texte à afficher, sa position, sa couleur et un identifiant.
— `createUIImg` : Ajoute une image à l'interface utilisateur. Elle prend en paramètre la page cible, le chemin de l'image, sa position et un identifiant.
— `createButton` : Crée un bouton interactif avec une apparence et une fonction spécifiques, ainsi qu'un coefficient de grossissement lors du hover de l'élément. Elle prend en paramètre la page cible, la position du bouton, son apparence, le coefficient de grossissement, sa fonction de rappel et d'autres attributs qui seront envoyés comme argument de la fonction de rappel.
— `createImgButton` : Variante de `createButton` qui utilise une image comme apparence du bouton. Elle prend des paramètres similaires mais inclut le chemin de l'image à utiliser.
— `checkButton` : Vérifie si un point (généralement la position de la souris) se trouve à l'intérieur d'un bouton. Elle prend en paramètre la position du bouton, les coordonnées x et y du point à vérifier, et renvoie 1 si le point est dans le bouton, 0 sinon.
— `renderUI` : Affiche tous les éléments d'interface de la page courante. Elle n'a pas de paramètre et ne renvoie rien.
— `createNotif` : Crée une notification et l'ajoute à la liste des notifications. Elle prend en paramètre le titre, la description, la position, la taille et l'image de fond de la notification.
— `uiNotifHandle` : Gère l'affichage des notifications en les rendant visibles à l'écran. Lorsqu'une notification expire, elle n'est pas supprimée immédiatement ; elle sera supprimée après l'affichage de la totalité des notifications. La fonction prend en paramètre la liste des notifications et ne renvoie rien.

4.3 Gestion de l'inventaire et du scroll

Pour assurer une gestion efficace de l'inventaire et une navigation aisée dans de larges listes d'items, TapVenture s'appuie sur deux modules complémentaires : le système d'inventaire (défini dans `inv.h` et `inv.c`) et le mécanisme de scroll (implémenté dans `scroll.h` et `scroll.c`). Ces deux modules sont étroitement liés, garantissant une interface utilisateur fluide, réactive et visuellement cohérente.

4.3.1 Implémentation du système de scroll

Le module de scroll gère l'affichage des contenus dont la taille dépasse la zone visible. Les fonctions principales et leur rôle sont détaillées ci-dessous :
— `creation_scroll` : Alloue et initialise un objet de défilement. Elle

configure la zone d'interaction et la zone de la scrollbar, calcule le coefficient de défilement via `cof_scrollbar_window` et définit la hauteur de celle-ci en fonction de la taille du contenu total. Ce mécanisme permet d'adapter dynamiquement la barre de défilement selon la quantité d'éléments affichés.

— `update_scroll` : Met à jour en temps réel la position et la taille de la barre de défilement. Elle ajuste des paramètres importants tels que `coef`, `min_pos` et `scrollbar_max_position` pour garantir que la barre ne dépasse pas la zone allouée, même lorsque le contenu évolue ou que l'utilisateur interagit avec la souris.

— `handle_scroll_event` : Gère les différents événements utilisateur (clic, mouvement de souris, molette) pour modifier la position de la scrollbar. Par exemple, lors d'un clic sur la barre, la fonction calcule un décalage correspondant à la position de la souris et ajuste `scroll_pos` en conséquence. Ces interactions sont encadrées par des vérifications via la fonction `est_dans_scroll` pour s'assurer que la mise à jour reste dans les limites définies.

— `aff_scrollbar_simple` : Se charge de l'affichage graphique de la barre. Elle dessine, en deux temps, le fond de la zone de défilement ainsi que la barre active en appliquant les couleurs configurables, garantissant ainsi un retour visuel constant lors de l'interaction.

4.3.2 Gestion de l'inventaire

L'inventaire regroupe l'ensemble des items disponibles pour le joueur ainsi que ceux de référence utilisés notamment pour les fusions. Son implémentation repose sur plusieurs fonctions travaillant de concert :

— `init_inv` et `gestion_inv` : Ces fonctions préparent l'inventaire en allouant de l'espace pour les items et en définissant l'aspect graphique de la grille (nombre de colonnes, nombre de lignes, dimensions des cases, décalages latéraux et verticaux). Elles intègrent également l'identifiant de scroll (`id_scroll`) pour lier le mécanisme de défilement à l'affichage de l'inventaire.

— `prem_vide` et `trier_inventaire` : Permettent de localiser la première case libre pour l'insertion d'un nouvel item et d'organiser les items existants selon un ordre précis. Cette gestion dynamique facilite des opérations telles que l'ajout, la fusion et la suppression d'items, notamment lors des événements d'interaction avec l'utilisateur.

— `refresh_inv` et `handle_inv_event` : Gèrent la mise à jour de l'affichage en réponse aux modifications de contenu et aux interactions utilisateur (clic, drag-and-drop, fusion d'items). Par exemple, la fonction `calcule_pos_inv` détermine l'item ciblé en se basant sur la position de la souris et la position actuelle de la scrollbar.

Les items eux-mêmes, représentés par la structure `item_t`, comportent des

attributs précisant leur identité (nom, label, fichier image) ainsi que des statistiques (statistiques de boost, rareté) essentielles pour la logique de fusion et de progression.

4.3.3 Intégration et synchronisation des modules

La coordination entre le module de scroll et l'inventaire est cruciale pour garantir un affichage synchronisé et une interaction intuitive :

- Chaque inventaire est associé à un identifiant de scroll qui permet d'aligner la zone de défilement avec les items affichés. Dès qu'une modification de la scrollbar est détectée (par exemple via `handle_scroll_event`), l'affichage de l'inventaire est recalculé pour refléter le nouvel état de la vue.
- L'actualisation de l'affichage se fait grâce à `refresh_inv`, qui repositionne les éléments de l'inventaire en prenant en compte la valeur de `scroll_pos`. Ainsi, lorsque l'utilisateur déplace la barre, la position des items dans la grille est décalée de manière cohérente.
- La synchronisation garantit également que les interactions sur les items (sélection, drag-and-drop, fusion) prennent en compte la position courante de la scrollbar, via des fonctions telles que `calcule_pos_inv` qui utilisent les coordonnées ajustées par la zone de défilement.
- Enfin, la fusion d'items, gérée par la fonction `deb_fusion`, illustre parfaitement cette intégration : après modification d'un item (par fusion ou mise à jour de statistiques), l'interface est rafraîchie pour reposer la grille d'inventaire avec le scrollbar positionné correctement.

Cette intégration étroite entre l'inventaire et le système de scroll permet à TapVenture de gérer efficacement de grandes quantités d'items tout en conservant une interface utilisateur réactive et intuitive (voir annexe 5).

4.4 Gestion des sauvegardes

Cette partie portera sur le systeme de sauvegardes de TapVenture basé sur des fichiers JSON.

4.4.1 Type de fichier

Toutes les données du jeux sont sauvegardées dans des fichiers JSON.
Nous avons fait ce choix pour plusieurs raisons :

- La simplicité de manipulation des fichiers
- Format de fichier standardisé
- La lisibilité des fichiers

C'est pourquoi nous avons opté pour un format de fichier courant tel que le JSON.
Le type de fichier JSON est un format de fichier texte standard qui est

utilisé pour représenter des données structurées basées sur la syntaxe des objets JavaScript.

4.4.2 Repartition des données

Les données sont reparties en plusieurs fichiers JSON.
— **player.json** : Contient les données liées au joueur.
— **heros.json** : Contient les niveaux des héros.
— **prestige.json** : Contient les données liées aux avancements de prestige.
— **item_inv.json** : Contient les données des items dans l'inventaire du joueur et de ses héros.
— **item_ref.json** : Contient les données de référence des items, leurs noms et un lien vers leurs images.

4.4.3 Structure des fichiers

Le type de fichier JSON est standardisé avec une structure de données en dictionnaire.
Chaque fichier JSON contient un dictionnaire avec des clés et des valeurs, comme on peut le constater sur l'image ci dessous.
Les clés et les valeurs sont des chaines de caractères mais les valeurs sont réinterpréter lors du chagement de la sauvegarde celons leurs types : entier, flottant, chaine de caractères.

FIGURE 3 – Extrait du fichier player.json

4.4.4 Fonctions de sauvegarde

— `createValueForKey` : Nous permet de sauvegarder les données, elle prend en paramètre des chaines de caratère : une clé, une valeur et le

nom du fichier liée a la sauvegarde. Le retour est un booléen qui nous permet de connaitre le bon deroulement de la fonction.
— `getValueForKey` : Nous permet de charger les données, elle prend en paramètre des chaines de caratère : une clé et le nom du fichier liée a la sauvegarde. Le retour est une chaine de caractères qui est la valeur associée a la clé.

4.5 Système de combat et de challenge

Cette partie détaille le système de combat, qui est au cœur du gameplay de TapVenture.

4.5.1 Structure de données

Le système de combat est organisé autour de plusieurs structures clés :
— **monstreInfo** : Contient les informations relatives à un monstre, notamment sa santé actuelle (mobHealth), sa santé initiale (iniHealth), ainsi que les valeurs minimales (coinMin) et maximales (coinMax) de pièces attribuées lors de sa défaite.
— **levelInfo** : Structure principale qui gère les niveaux de combat, contenant un tableau de monstres, le temps pour vaincre le monstre courant, le timer de début du niveau, le nombre de monstres tués, le nombre de monstres à tuer pour terminer le niveau, le niveau courant, le niveau maximal atteint, et des tableaux d'étiquettes et de chemins d'images pour les monstres.

4.5.2 Progression des monstres

La progression de la difficulté des monstres suit une formule exponentielle :
— **Santé des monstres normaux** : $\text{Santé}_i = \text{Santé}_{i-1} \times 1,5$
— **Récompense en or (min/max)** :
$\text{Or\_Min/Max}_i = \text{Or\_Min/Max}_{i-1} \times 1,25$
Pour les boss (apparaissant tous les 5 niveaux), les formules sont amplifiées :
— **Santé des boss** : $\text{Santé\_Boss}_i = \text{Santé}_{i-1} \times 5$
— **Récompense en or (min/max)** :
$\text{Or\_Min/Max\_Boss}_i = \text{Or\_Min/Max}_{i-1} \times 3,5$
Cette progression exponentielle garantit que le jeu devient progressivement plus difficile, incitant les joueurs à améliorer continuellement leurs héros, leur dégât et à utiliser le système de prestige.

4.5.3 Animation de dégâts

Le système implémente une animation visuelle lorsqu'un monstre subit des dégâts :

- Lors d'une attaque, la fonction `playDamageAnimation()` est appelée
- Cette fonction change temporairement la texture du monstre pour afficher une version «endommagée»
- La fonction `checkDamageAnimation()` vérifie si la durée de l'animation est écoulée et restaure l'apparence normale du monstre

4.5.4 Fonctions du système de combat

- `initLevel` : Initialise le niveau de combat en configurant les paramètres relatifs aux monstres, notamment leur santé et les récompenses en pièces. La fonction prend en paramètre un tableau de structures `monstreInfo` et renvoie 0 en cas de succès.
- `attack` : Gère la logique d'attaque sur un monstre. Elle réduit la santé du monstre, vérifie s'il est vaincu, attribue des récompenses et gère la progression de niveau. La fonction prend en paramètre un tableau d'arguments contenant les dégâts infligés et un booléen indiquant si l'attaque provient du joueur. Elle retourne 1 en cas de succès.
- `initBoss` : Initialise un combat de boss avec un temps limité pour le vaincre. Elle prend en paramètre le temps en secondes et renvoie 1 en cas de succès.
- `mobHandler` : Gère le comportement des monstres, notamment la réinitialisation de la santé des boss et la configuration des timers. Elle ne prend pas de paramètre et ne renvoie rien.
- `displayTimers` : Affiche les timers liées au combat, notamment le timer du défi en cours et celui du boss si applicable. Elle met à jour ou crée les éléments d'interface utilisateur correspondants.
- `changeLevel` : Change le niveau de combat en fonction du paramètre passé. Si le paramètre est 1, le niveau augmente (si possible) ; s'il est 2, le niveau diminue (si possible). Elle prend en paramètre un tableau contenant le sens du changement et renvoie 1 en cas de succès.
- `mobAnimationHandler` : Gère l'animation des monstres pendant le combat, notamment après avoir subi des dégâts. Elle ne prend pas de paramètre et ne renvoie rien.

4.5.5 Système de challenge

Le système de challenge permet au joueur de participer à un défi limité dans le temps pour gagner une récompense en or. La mécanique est basée sur une structure globale (`Challenge_t`) définie avec plusieurs paramètres :

- **active** : Indique si le challenge est en cours.
- **startTime** : Temps de démarrage du challenge (basé sur `SDL_GetTicks()`).

— **duration** : Durée maximale du challenge en secondes.
— **target** : Nombre de monstres à tuer pour réussir le challenge.
— **reward** : Récompense de base, qui sera multipliée par le niveau actuel.
— **cooldown** : glsTemps de recharge entre deux challenges.
— **lastTime** : Horodatage du dernier lancement du challenge.

La fonction `launchChallenge` initialise un nouveau challenge si plusieurs conditions sont remplies :

— Le challenge n'est pas déjà actif.
— Le niveau courant n'est pas celui d'un combat de boss.
— Le temps écoulé depuis le dernier challenge dépasse le temps de recharge. Dans le cas contraire, une notification informe le joueur du temps restant.

Une fois lancé, «`launchChallenge`» réinitialise le compteur de monstres tués (`level.mobKilled`) et définit le nombre de monstres à tuer (`level.mobToKill`) en fonction de la cible du challenge.

La fonction `updateChallenge` est appelée de manière régulière pour :

— Vérifier si la durée maximale du challenge est dépassée. Si oui, une notification d'échec est affichée via `createNotif` et le challenge est réinitialisé par `resetChallenge`.
— Vérifier si le joueur a atteint le nombre de monstres requis. En cas de succès, le joueur reçoit une récompense en or (la valeur étant multipliée par son niveau actuel) suivie d'une notification de réussite, puis le challenge est réinitialisé.

Enfin, la fonction `resetChallenge` remet le système de challenge à son état initial en désactivant le défi et en réinitialisant les compteurs de combats.

4.6 Les héros et jeux hors ligne

Les héros sont des personnages qui infligent des dégâts aux monstres tout les x temps pour aider le joueur dans sa progression.

4.6.1 Structure des héros

— `degat` : Nombre qui représente les dégâts qui seront appliquer par ce héro lors de son attaque.
— `prix` : Nombre qui représente le prix du héro pour l'améliorer au niveau suivant.
— `level` : Nombre qui représente le niveau actuel du héro.
— `cooldown` : Nombre qui représente le temps d'attente entre chaque attaque du héro.
— `lastAttack` : Nombre qui représente le moment de la dernière attaque du hero.

Les heros sont stockés dans une liste de type `hero`, pour les manipuler plus facilement. Une structure enum `herosIndex` est utilisée pour touver facilement l'indice de chaque heros dans la liste avec comme dennier element du `herosIndex` un `HEROS_COUNT` qui représente le nombre total de héros.

4.6.2 Fonctions des héros

- `initHeros` : Nous permet d'initialiser tous les héros avec des valeurs prédéfinies, chaque héros est une évolution du précédent. L'évolution entre les héros est gérée par des constantes telles que `DEGAT_UP`, `PRIX_UP`, `COOLDOWN_UP` qui multiplie les variables correspondantes (`dégâts`, `prix`, `cooldown`). La fonction ne prend aucun paramètre, son retour est un booléen qui nous permet de connaître le bon déroulement de la fonction.
- `upgradeHeros` : Nous permet d'améliorer un héro spécifique, ses attribut tel que les `dégâts` et le `prix` sont multipliés par les constantes `DEGAT_UPGRADE` et `PRIX_UPGRADE`. De plus, le niveau (`level`) du héro est incrémenté de 1. Elle prend en paramètre un int qui correspond au `herosIndex` pour représenter l'indice du héro et un booléen `pay` qui permet de choisir si l'amélioration du héro va décrémenter l'or du joueur. Ce booléen nous permet lors du chargement de la sauvegarde d'effectuer les amélioration sur les héros pour les mettre a leurs niveau précédent sans faire payer le joueur. Le retour est un booléen qui nous permet de connaître le bon déroulement de la fonction.
- `attackHeros` : Nous permet de gérer les attaques de tous les héros, la fonction est appelée à chaque tour de la boucle principale de jeu. Lors d'un appel, elle vérifie pour chaque héros si celui-ci peut attaquer. Elle compare le moment actuel récupéré grâce à la fonction `SDL_GetTicks` et le moment de la dernière attaque du héro (`lastAttack`) plus le temps d'attente du héro (`cooldown`). Si cette condition est rempli, alors on fait appel à la fonction `attack` qui se charge d'appliquer les `dégâts` au monstres, on lui passe en paramètre un tableau de pointeur sur void avec les dégât, puis un booléen qui indique si c'est le joueur qui attaques ou pas.

4.6.3 Gestion du jeu hors ligne

La gestion du jeu hors ligne permet de simuler la progression du joueur, du temps passer et des héros pendant son absence. Cette gestion est assurée par la fonction `goldGainOffline` dans le fichier `player.c`. Elle effectue les opérations suivantes :

- **Calcul du temps écoulé** : La fonction détermine le temps écoulé depuis la dernière sauvegarde en comparant le moment actuel (récupéré via `SDL_GetTicks`) avec le moment de la dernière sauvegarde.

Ce temps est utilisé pour simuler les actions des héros et les gains du joueur pendant son absence.

— **Gains d'or hors ligne** : Le temps écoulé est multiplié par la génération d'or par seconde des héros (donné par la fonction `herosGoldGenBySec`). Le résultat est ajouté au total d'or du joueur, permettant de récompenser sa progression même lorsqu'il n'est pas actif.

— **Mise à jour des notifications** : Lors de la reconnexion, des notifications sont affichées pour informer le joueur des gains obtenus et des événements survenus pendant son absence.

Cette logique repose sur la capacité des héros à infliger des dégâts automatiquement et à générer des récompenses, tout en prenant en compte les mécaniques de recharge des défis pour offrir une expérience de jeu fluide et cohérente, même hors ligne.

La gestion du jeu hors ligne permet également de recommencer un challenge si le temps de recharge est écoulé :

— **Calcul du temps avant un nouveau challenge** : Si le joueur participe au challenge nécessitant un temps de recharge, la fonction `launchChallenge` vérifie si le temps écoulé est suffisant pour réinitialiser ces défis. Le temps restant avant de pouvoir participer à un nouveau challenge est calculé en soustrayant le temps écoulé depuis le dernier challenge au temps actuelle. Si le temps écoulé dépasse le temps de recharge, le joueur peut immédiatement participer à un nouveau challenge.

4.7 Système de prestige

Cette partie portera sur le système de prestige, qui est un élément clé de TapVenture.

4.7.1 Structure de données

L'arbre de prestige est une structure prestigeTree_t (Voir annexe 4), composée de 3 structures prestigeList : Gold, Damage, Prestige. Cette structure contient une liste de structure prestigeItem et leur nombre. La structure prestigeItem est décrite par :

— **un nom** : le nom de l'amélioration.

— **une description** : une brève explication de l'effet de l'amélioration.

— **un coût** : le coût en points de prestige pour débloquer l'amélioration.

— **une valeur** : la valeur de l'effet de l'amélioration.

— **une fonction** : une fonction associée à l'amélioration, qui applique l'effet lorsqu'elle est débloquée.

— **un index** : sa position dans l'arbre de prestige.

— **des coordonnées** : sa position dans la fenêtre de jeu.

Les améliorations sont stockées comme variables flottantes globales agissant comme des multiplicateurs : prestigeModifier, damageModifier, goldModifier. Mais aussi comme des entiers : heroKeepLevel, heroKeepUpgrade.

4.7.2 Fonction du système de prestige

— `addPrestigeItem` : Nous permet d'ajouter un nouveau bonus de prestige achetable dans l'arbre passé en paramètre en précisant ses informations telles que le nom, la description, le coût, la fonction d'amélioration, et sa valeur d'amélioration.

— `prestigeTree_init` : Initialise l'arbre de prestige en créant les différentes branches (Gold, Damage, Prestige) et en y ajoutant les améliorations correspondantes, calcule l'écart nécessaire entre chaque amélioration pour afficher un nombre d'amélioration fixe défini par `NB_PRESTIGE_ITEMS_PER_PAGE`. Et enfin elle crée l'affichage de l'arbre avec des boutons d'améliorations. La fonction ne prend aucun paramètre et ne renvoie rien.

— `buyPrestigeItem` : Permet d'acheter une amélioration de prestige. Elle vérifie si le joueur a suffisamment de points de prestige pour acheter l'amélioration et qu'il possède les améliorations précédentes (si il ne s'agit pas de la 1re amélioration), puis applique l'effet de l'amélioration et met à jour les points de prestige restants. Elle prend en paramètre l'arbre de prestige qui est une chaine de caractere correspondant a «Gold», «Damage» ou «Prestige», l'index de l'amélioration à acheter, et le nombre de points de prestige du joueur. Elle ne renvoie rien.

— `doPrestige` : Permet de réinitialiser la progression du joueur en échange de points de prestige. Elle met à jour les multiplicateurs globaux (goldModifier, damageModifier, prestigeModifier) et le nombre de héros conservés après le prestige. Elle prend rien en paramètre et ne renvoie rien.

— `getMaxPrestigeItems` : Renvoie le nombre maximum d'améliorations de prestige disponibles dans l'arbre de prestige. Elle prend en paramètre l'arbre de prestige qui est une chaîne de caractères correspondant a «Gold», «Damage» ou «Prestige» et renvoie un entier.

— `checkDisplayPrestigeItemText` : Vérifie si un texte d'amélioration de prestige doit être affiché ou non, et met à jour l'affichage en conséquence. Elle n'a pas de paramètre et ne renvoie rien.

5 Conclusion

Le projet TapVenture a été un succès marquant, tant sur le plan technique que sur le plan de l'expérience utilisateur. Nous avons réussi à créer un jeu

de type "clicker" fonctionnel et addictif, avec une interface intuitive et des mécaniques de progression bien équilibrées.

La gestion des héros, du système de combat et de l'arbre de prestige a été particulièrement bien intégrée, offrant une expérience fluide et engageante. Le système de jeu hors ligne permet aux joueurs de progresser même en leur absence, maintenant ainsi leur intérêt sur le long terme.

La répartition des tâches a été efficace, chaque membre de l'équipe ayant pu contribuer selon ses compétences et ses centres d'intérêt. La communication constante entre les membres, facilitée par l'utilisation de Discord, a permis d'identifier rapidement les problèmes et de trouver des solutions adaptées.

Nous avons également perfectionné notre maîtrise d'outils de développement tels que GitHub pour la gestion de version et le tableau Kanban pour le suivi des tâches. L'utilisation de bibliothèques comme SDL pour le développement graphique et de formats standards comme JSON pour les sauvegardes nous a permis de nous concentrer sur les aspects créatifs du jeu plutôt que sur les détails techniques.

Si nous avions disposé de plus de temps, nous aurions aimé ajouter davantage de contenus (niveaux, héros, défis) et améliorer certains aspects visuels. Néanmoins, le résultat final répond pleinement aux objectifs que nous nous étions fixés au début du projet.

Une comparaison détaillée entre le planning initial (Gantt prévisionnel) et le planning final [2] a révélé les ajustements réalisés en cours de projet. Ce bilan a mis en lumière notre capacité d'adaptation face aux imprévus et aux défis rencontrés, renforçant la coordination et l'agilité de l'équipe.

Cette expérience nous a non seulement permis de développer nos compétences techniques en C et en conception de jeux, mais aussi d'apprendre à gérer efficacement un projet d'équipe sur plusieurs mois, une compétence précieuse pour notre future carrière professionnelle.

2. Voir le Gantt final : Gantt final

6 Annexe

Glossaire

challenge Un défi que le joueur doit relever pour gagner des récompenses. 14

dégâts Quantité de points de vie retirés au monstre. 3

hors ligne Le joueur n'a pas le jeu ouvert mais le jeu continue de tourner. 13

or Monnaie du jeu, obligatoire pour les améliorations. 3

prestige recommencé le jeu mais avec des points à dépenser dans un arbre de compétences.. 3

temps de recharge Le joueur doit attendre un certain temps avant de pouvoir recommencer un challenge. 14

Images du jeu :

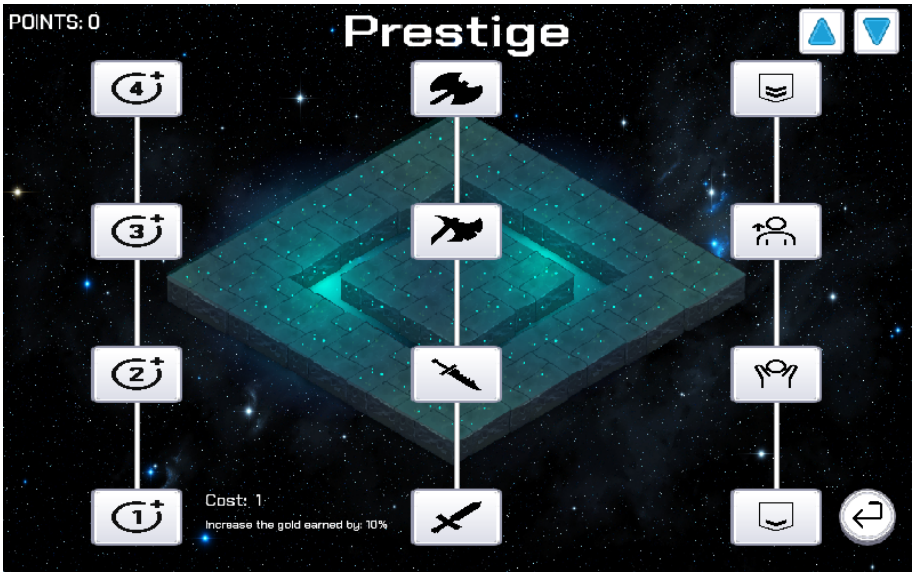

FIGURE 4 – Arbre de compétence de prestige

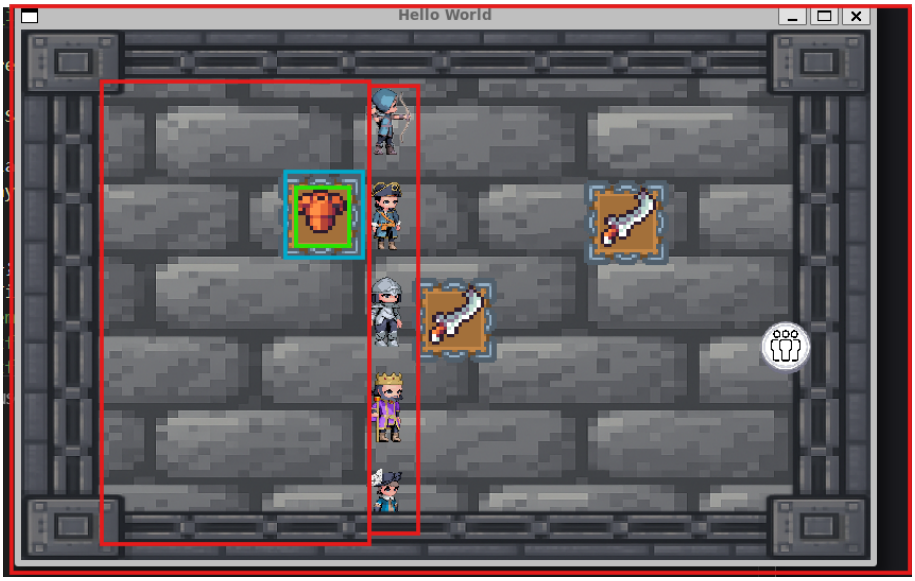

FIGURE 5 – Interface de l'inventaire avec les différents éléments visuels. Le carré rouge indique la zone de gestion des items et des interface scroll et les carrer vers les items .

FIGURE 6 – Menu principale

FIGURE 7 – Menu des héros

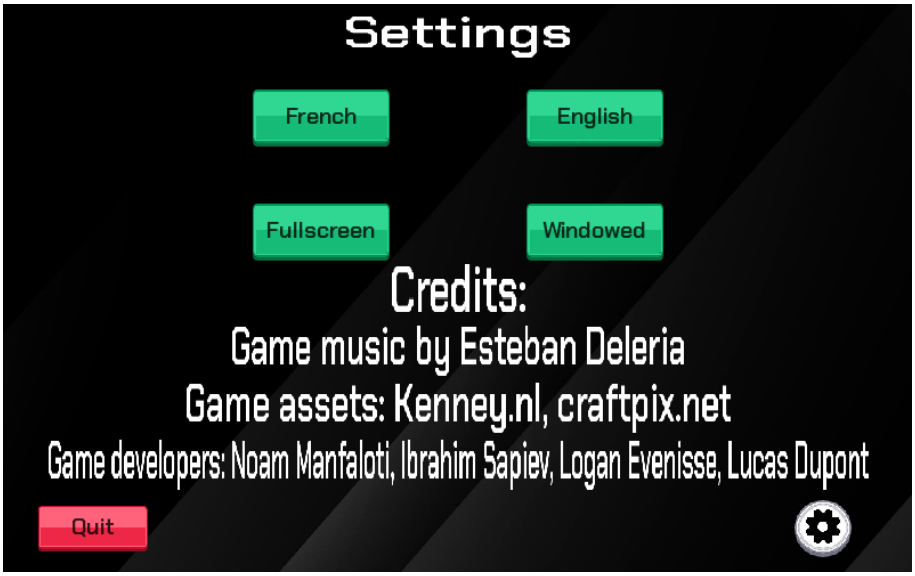

FIGURE 8 – Menu des paramètres