

Hitty Knife - Redux

Projet de programmation

ALVES VIEIRA Lionel | COULOMBEL Corentin | FACCINI BANDEIRA Guilherme | LES BIENS Cédric | LOTHSAVAN Hugo

# Sommaire (liens cliquables) :

## [Introduction](#_Introduction_:)

## [description du jeu](#_Description_du_jeu)

## [mÉthodologie de dÉvelopement](#_Méthodologie_de_développement)

## [Évolution du jeu et valeur ajoutÉe](#_Évolution_du_jeu)

## [conception et architecture du jeu](#_Conception_et_Architecture)

## [points algorithmiques et techniques](#_Points_algorithmiques_et)

## [Liste des fonctionnalitÉs](#_Liste_de_fonctionnalités_1)

## [conclusion](#_Conclusion_:)

# Introduction :

*"Hitty Knife Redux"* est une réinterprétation du jeu *"Hitty Knife"*, dans lequel les joueurs doivent détruire des cibles mobiles en maîtrisant l’art du lancer de couteau.  
Initié par M. Yan Jurski, ce projet offre une expérience de jeu immersive et divertissante, tout en offrant des possibilités infinies de variations et d'améliorations.

Notre objectif initial est séparé en deux parties distinctes : une première partie où l’on devait créer des bases solides, proches de la description initiale du jeu et une seconde pour apporter une plus-value au jeu, des ajouts intéressants et pertinents, tout en faisant certaines abstractions.

Le jeu repose sur des mécanismes simples et addictifs où les joueurs doivent ajuster leur lancer pour atteindre des cibles, tout en affrontant des Boss redoutables.

Cette version repensée propose une expérience de jeu enrichie, avec des vagues de défis, comprenant des Boss uniques dans leur conception et des environnements variés.

De plus, les joueurs peuvent choisir parmi une sélection de skins de couteaux pour personnaliser leur expérience de jeu, et s'affronter dans un tout nouveau mode versus !

Dans ce rapport, nous explorerons en détail les différentes phases du développement du projet *"Hitty Knife Redux"*,en commençant pas une description approfondie du jeu initial. Nous discuterons ensuite de la méthodologie de développement utilisée. En outre, nous présenterons les différentes variations et améliorations implémentées pour enrichir l'expérience de jeu, ainsi que les choix de conception et d’architecture qui ont guidé le développement.

Enfin, nous dresserons un bilan du projet et proposerons des pistes d’améliorations pour l’avenir. Bonne lecture !

# Description du jeu :

Le jeu *"Hitty Knife Redux"* propose une expérience amusante où les joueurs incarnent des lanceurs de couteaux talentueux, chargés de détruire des cibles mobiles avec précision. Inspiré du jeu original *"Hitty Knife"*, cette version revisitée propose de nouvelles fonctionnalités et améliorations pour enrichir l'expérience de jeu.

Dans ce jeu, les joueurs sont confrontés à des cibles flottantes en mouvement, représentées sous forme de cibles circulaires de différentes tailles. Ils doivent ajuster leur lancer de couteau pour atteindre ces cibles tout en évitant les obstacles et en tenant compte de la trajectoire balistique de leurs lancers. Chaque cible atteinte rapporte des points, tandis que les échecs réduisent le nombre de vies du joueur. Arrivé à trois lancers loupés, le joueur doit recommencer le niveau.

Les niveaux sont quant à eux divisés en 4 vagues : 3 vagues avec des cibles « normales », et « bonus », tandis que la dernière vague est un boss. Les boss, contrairement aux cibles usuelles, sont dotés de trajectoires complexes mais régulières, différentes pour chacun des boss et rendant la visée plus difficile. Certains d’entre eux ont des fonctionnalités spécifiques que nous aborderons plus tard dans ce rapport. Ces confrontations ajoutent un défi supplémentaire au jeu, testant les compétences de précision et de timing du joueur.

Un des atouts majeurs de *"Hitty Knife Redux"* est l'introduction d’un tout nouvel univers -plus d’une quinzaine de backgrounds ! La direction artistique a été entièrement repensée et les joueurs peuvent désormais jouer dans un Japon féodal.

En outre, les joueurs peuvent personnaliser leur expérience de jeu en sélectionnant des skins de couteaux, musiques et backgrounds uniques, leur permettant d'exprimer leur style et leur personnalité. Tous ces ajouts cosmétiques sont accessibles par un « shop » implémenté dans le jeu, permettant au joueur de dépenser de la monnaie virtuelle pour acheter ces différentes cosmétiques ! L'argent est accessible simplement en jouant, chaque niveau remporté rapportant au joueur un certain montant d'argent.

Cependant, certaines ce ces récompenses ne seront disponibles au joueur qu’en avançant dans le tout nouveau BattlePass. Un tout nouvel ajout, offrant encore plus de satisfaction et de récompenses, pour chaque palier atteint à force de jouer et de s’améliorer.

Enfin, un tout nouveau mode versus permet aux joueurs de s'affronter dans des duels palpitants, testant ainsi leurs compétences contre celles de leurs adversaires. Les joueurs doivent vaincre un certain nombre de cibles et de boss avant leur adversaire.

# Méthodologie de développement :

Le développement du jeu *"Hitty Knife Redux"* s'est déroulé selon une méthodologie AGILE, avec une forte collaboration et flexibilité pour s'adapter aux évolutions du projet. Nous avons utilisé GitLab comme système de contrôle de versions, ce qui nous a permis de suivre efficacement l'avancement des tâches, de gérer les problèmes et de collaborer sur le code source.

Notre approche AGILE s'est articulée autour de sprints hebdomadaires. Au début de chaque sprint, une réunion était organisée où chaque membre de l'équipe présentait ce qu'il avait accompli lors du sprint précédent et discutait des tâches à réaliser pour le sprint en cours. Cette réunion était également l'occasion de planifier les objectifs et les priorités pour le sprint à venir, ainsi que d’échanger avec le Chargé de Projet, M. Jurski, dont l’écoute, l’aide et les conseils ont été d’une importance cruciale.

L'utilisation d'issues sur GitLab a permis à chaque membre de l'équipe de proposer des tâches et de se les assigner ou les laisser à d’autres membres en fonction de leur disponibilité et de leurs compétences. Cette approche a favorisé une répartition équilibrée du travail et une collaboration fluide au sein de l'équipe. De plus, la flexibilité offerte par cette méthode a permis d'ajuster les priorités et les objectifs en fonction des besoins du projet.

Les 6 premiers sprints ont été consacrés à l'établissement d'une base solide pour le jeu et la direction artistique, en mettant en place les fonctionnalités de base et en stabilisant le code. Les 6 sprints suivants ont été consacrés à l'ajout de fonctionnalités avancées et à la perfection du jeu. Cela inclut la conception et l'implémentation de nouveaux boss uniques, une refonte complète du GUI –dont on parlera plus en détails dans la partie « Évolution du jeu et valeur ajoutée » -, une perfection du code source (refactorisation, JavaDoc…) et l’ajout du mode Versus.

En résumé, notre méthodologie de développement agile et notre utilisation de GitLab nous ont permis de travailler de manière collaborative, flexible et efficace pour donner vie au jeu.

# Évolution du jeu et valeur ajoutée :

L'évolution du jeu et sa valeur ajoutée ont été des aspects cruciaux de notre projet. Nous avons commencé par recréer le jeu de base, en mettant en place un menu intuitif permettant aux joueurs d'accéder facilement au jeu en appuyant simplement sur un bouton. Au sprint 6, les ajouts et refontes ont commencé :

* **Reconception complète du GUI en CardLayout.** Cette étape a été plutôt ardue et complexe, mais a été très utile dans le futur du projet, permettant l’ajout de Panel beaucoup plus facilement, une ergonomie bien meilleure et une meilleure structure du code, permettant aussi de faire beaucoup de factorisation et d’encapsulation.
* **Introduction du mode Versus**, où deux joueurs s’affrontent et le premier arrivé à 50 points gagne. Ces points se gagnent en touchant les différentes cibles. Des bosses peuvent aussi apparaître. On peut aussi changer le background de ce mode à l’aide des différents backgrounds gagnés ou achetés.

Une image contenant texte, nuage, Jeu PC, lune

Description générée automatiquement

* Une image contenant capture d’écran, texte

  Description générée automatiquement**Ajout d’une boutique** pour l’achat de skins de couteau et d’arrière-plans exclusifs en mode versus, avec un panier a validé grâce à l’argent gagné en jeu.
* Une image contenant pixel, Graphique, Caractère coloré

  Description générée automatiquementUne image contenant Graphique, capture d’écran, cercle, pixel

  Description générée automatiquementUne image contenant Graphique, graphisme, clipart, symbole

  Description générée automatiquementUne image contenant pixel

  Description générée automatiquement**Inclusion de quatre types de boss différents**, chacun avec sa propre logique de déplacement.
* Une image contenant lune, arbre, plein air, nuit

  Description générée automatiquement**Ajout d’une toute nouvelle Direction Artistique**. Le jeu se déroule dans un univers médiéval japonais, avec des boss inspirés des Yokais, élargissant ainsi l'univers du jeu.
* Une image contenant texte, capture d’écran, graphisme, Compositing numérique

  Description générée automatiquement**Mise en place d'un inventaire** permettant aux joueurs de personnaliser leur expérience de jeu en mode versus : musique, skin d'arme et background.
* Une image contenant Jeu PC, capture d’écran, Logiciel de jeu vidéo, Jeu vidéo de stratégie

  Description générée automatiquement**Intégration d'un BattlePass** offrant des récompenses à long terme telles que de l'argent, des skins d'armes et différentes musiques. Les paliers sont atteints au fur et à mesure que le joueur atteint un certain montant d’expérience. Cette expérience est gagnée en terrassant des cibles dans le mode Solo.
* **Introduction de cibles bonus** tels que l'Excalibur, offrant des avantages spécifiques aux joueurs comme traverser les cibles (permettant d’en toucher plusieurs à la fois), d’être invincible pendant un court moment et de terrasser les boss en un coup. Il y a aussi les cibles XP et Money permettant respectivement de gagner un certain montant d’expérience et d’argent et enfin, un bonus freeze, permettant de stopper pendant quelques instants les cibles mouvantes.

Une image contenant pixel, capture d’écran, art

Description générée automatiquement Une image contenant pixel, jaune, Caractère coloré, capture d’écran

Description générée automatiquement Une image contenant Graphique, pixel, conception

Description générée automatiquementUne image contenant pixel, Caractère coloré, Graphique, motif

Description générée automatiquement

# Conception et Architecture du jeu (double clic):



1. **MainFrame** : Une image contenant texte, diagramme, Parallèle, capture d’écran

   Description générée automatiquement
   * **Description** : **MainFrame** est la classe principale de la fenêtre du jeu. Elle utilise un **CardLayout** pour gérer différents panneaux (panels) tels que **GameView**, **ShopPanel**, **HomeMenu**, et **BattlePassPanel**.
   * **Rôle** : Elle centralise l'affichage et la navigation entre les différentes sections de l'application. Cela permet de passer facilement d'un écran à un autre tout en gardant une structure cohérente.
2. **Entités du jeu** :
   * **Couteaux et Cibles** :
     + Les entités du jeu sont constituées de couteaux et de cibles.
     + **Cible** : Une classe parente **Cible** est créée, de laquelle héritent les classes **de bonus** et **Boss**.
     + **Boss** : La classe **Boss** est abstraite, et chaque type de boss spécifique hérite de cette classe. Cela permet de définir des comportements communs pour tous les boss tout en permettant des spécialisations pour chaque type de boss.

Une image contenant texte, diagramme, capture d’écran, ligne

Description générée automatiquement

**Choix d'abstraction** : En utilisant des classes abstraites et des héritages, nous pouvons réutiliser du code commun et faciliter l'extension avec de nouveaux types de cibles et de boss.

1. **Package Geometry** :

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

* + **Description** : Le package **geometry** gère toutes les opérations liées aux coordonnées, aux équations de mouvement et à la logique de déplacement.
  + **Rôle** : En encapsulant la logique géométrique dans un package distinct, nous isolons les calculs complexes des autres parties du code, ce qui simplifie le débogage et la mise à jour de ces fonctionnalités.

1. **Package User** : Une image contenant texte, capture d’écran, ligne, diagramme

   Description générée automatiquement
   * **Description** : Le package **user** contient toutes les informations nécessaires pour les sauvegardes des utilisateurs ainsi que la base de données de tous les joueurs.
   * **Rôle** : En centralisant la gestion des utilisateurs et des données persistantes, nous facilitons la sauvegarde et le chargement des profils de joueurs, ainsi que l'intégration de fonctionnalités liées aux utilisateurs (comme les scores et les paramètres).
2. **Game** :

Une image contenant texte, diagramme, capture d’écran, Parallèle

Description générée automatiquement

* + **Description** : La classe **Game** contient la majorité de la boucle principale du jeu, avec une fonction **loop** qui gère les ticks.
  + **Rôle** : Cette structure permet de séquencer les événements du jeu de manière régulière et contrôlée. La boucle principale est le cœur de la logique du jeu, assurant la mise à jour continue de l'état du jeu.

1. **Managers de Bonus et de Rounds** :
   * **Description** : Les bonus et les rounds sont factorisés dans des managers dédiés.
2. **Debug** :
   * **Description** : Permet de debugger le jeu
   * **Rôle** : A chaque période de développement et ajout dans le jeu, cette classe a été essentielle dans le débogage et dans le bon fonctionnement du jeu.

# Points algorithmiques et techniques :

## Sauvegardes :

La gestion des sauvegardes dans *"Hitty Knife Redux"* est un élément crucial pour offrir aux joueurs une expérience fluide et personnalisée. Nous avons mis en place un système de sauvegarde efficace, basé sur une structure de données sérialisée, permettant de conserver l'état du jeu pour chaque utilisateur.

### Implémentation :

L'implémentation des sauvegardes repose sur l'utilisation de fichiers JSON pour stocker les informations des utilisateurs et leur progression dans le jeu. Nous avons développé une classe User qui représente les utilisateurs, avec des attributs tels que le nom d'utilisateur, le mot de passe, l’inventaire, le niveau actuel, le chemin de sauvegarde et le montant d'argent. Cette classe est sérialisée dans un fichier JSON pour chaque utilisateur.

La classe *UserManager,* quant à elle, joue un rôle central dans la gestion des utilisateurs et de leurs données dans *"Hitty Knife Redux"*. Elle simule une base de données en chargeant toujours la même instance de *UserManager* au lancement du jeu.

#### Simulation de base de données

Le *UserManager* est implémenté selon le modèle Singleton, assurant qu'une seule instance de la classe est utilisée dans toute l'application. Cela permet de centraliser la gestion des utilisateurs et de maintenir la cohérence des données.

La simulation de base de données est réalisée en chargeant les informations des utilisateurs à partir d'un fichier JSON lors du démarrage de l'application. Les utilisateurs sont stockés dans une structure de données HashMap, où chaque utilisateur est associé à son nom d'utilisateur. Cela permet un accès rapide et efficace aux données utilisateur tout au long de l'exécution du jeu.

La classe UserManager gère la création, la connexion et la sauvegarde des utilisateurs. Elle utilise un algorithme de hachage SHA-256 pour sécuriser les mots de passe stockés.

Lorsqu'un utilisateur se connecte dans le ConnectionMenu, le système charge son état de jeu à partir du fichier JSON correspondant à son nom d'utilisateur.

#### Gestion des sauvegardes

Lorsque l'utilisateur effectue des actions importantes dans le jeu, telles que la progression de niveau ou l'achat d'éléments cosmétiques, l'état du jeu est sauvegardé automatiquement.

La classe Game gère quant à elle la sauvegarde de l'état du jeu.   
Cette approche garantit que les données des utilisateurs sont persistantes et accessibles à chaque exécution du jeu.

## CARDLAYOUT :

Lors du sprint 6, nous avons décidé d'adopter un **CardLayout** pour la gestion des différents écrans de notre jeu. Cette décision a permis de simplifier la navigation entre les différents panneaux (panels) tout en maintenant une seule fenêtre (frame) principale. Voici les points clés de cette implémentation :

* **Navigation Simplifiée** : Le **CardLayout** permet de basculer facilement entre les différents panels en utilisant des boutons dédiés. Chaque bouton, lorsqu'il est pressé, appelle une méthode **switchToPanel(String name)** pour afficher le panel souhaité.
* **Gestion de l'État des Panels** : Lorsque l'on change de panel, l'ancien panel est mis en pause grâce à la méthode **pausePanel()**, ce qui permet de garantir que le jeu se déroule sans interruption. Par exemple, si un joueur décide de changer de skin pendant un niveau, le panel du jeu est mis en pause pour assurer que le gameplay reste intact.
* **Vérification du Mode de Jeu** : Lors du passage d'un panel à un autre, particulièrement pour le jeu (**GameView**), une vérification est effectuée pour déterminer si le mode de jeu est en solo ou en versus. Cette vérification se fait à travers la variable **isSolo** de **GameView**. Si **isSolo** est vrai, un **GameView** solo est ouvert ; sinon, un **GameView** versus est ouvert.
* **Création Dynamique des Panels** : Des méthodes spécifiques comme **CreateGamePanel()**, **CreateShopPanel()**, etc., sont implémentées pour créer et initialiser chaque panel du jeu. Ces méthodes sont appelées lors de la construction de **MainFrame**, et les panels sont ajoutés aux écouteurs d'actions (**ActionListeners**) des boutons correspondants. C'est également à ce moment que les observateurs, comme **battlePassPanel()**, sont passés pour surveiller les événements du jeu.

## BATTLEPASS :

* **Structure du BattlePass** :
  + Une **Map<Integer, Boolean>** nommée **rewardsReached** est utilisée pour suivre les paliers atteints par le joueur. Chaque entrée dans la carte représente un palier spécifique et si la récompense associée a été obtenue.
  + Une **private Map<Integer, Image>** nommée **rewardIcons** permet de personnaliser les icônes des récompenses pour chaque type de palier, offrant une interface visuelle attrayante et claire pour le joueur.
* **Logique de Progression** :
  + La progression du joueur est gérée dans la classe **Game**. Chaque fois qu'une cible est touchée, le joueur gagne de l'expérience (XP) via la méthode **addXP()**.
  + Lorsque l'XP accumulé atteint un certain seuil (défini par **thresholdXP**), la méthode **checkLevelUp()** est appelée pour vérifier si le joueur passe au niveau suivant. Si le niveau est atteint, **notifyLevelObservers()** est appelée pour informer les observateurs du changement de niveau.
* **Distribution des Récompenses** :
  + Les récompenses sont distribuées via la méthode **giveRewards()**, qui utilise un **switch** pour déterminer quelle récompense donner en fonction du niveau actuel du joueur.
  + L'interaction entre **Game** et **BattlePassPanel** est assurée en temps réel grâce à un observateur (**GameObserver.java**). Cet observateur écoute les événements de niveau et met à jour le **BattlePassPanel** en conséquence. Chaque fois qu'un niveau est atteint, la méthode **notifyLevelObservers()** dans **checkLevelUp()** notifie les observateurs, déclenchant ainsi l'avancement dans le Battle Pass.

## Gestion des mouvements / géométrie

Tous les mouvements dans le jeu sont gérés à partir de la classe Geometry. Cette dernière comporte plusieurs fonctions décrivant différents types de mouvements (horizontaux, verticaux, etc.), également des mouvements soumis à la gravité, comme le saut du couteau.

La classe Geometry utilise deux autres classes, la classe Coordinate qui décrit un tuple de double x, y correspondant aux coordonnées théoriques d’un objet, et la classe Loop qui est la boucle principale du jeu et qui actualise le jeu.

Ce qui est important pour Geometry dans la classe Loop, c’est la valeur delta qui permet de calculer avec précision les nouveaux emplacements des objets en mouvement, en tenant compte du temps écoulé entre les mises à jour. Cela permet de créer des simulations plus réalistes et fluides dans des environnements de jeu ou des applications impliquant des objets géométriques en mouvement.

#### Affichage théorique et réel

Deux types de coordonnée sont distincts dans le jeu, les coordonnée théoriques et réelles. En effet, les coordonnées théoriques sont utilisées pour faire interagir les différents objets géométriques entre eux, tandis que les coordonnées réelles prennent en compte la taille de l’écran et gardent le même ratio entre les différents éléments du jeu.

Cela a pour avantage de pouvoir choisir quelle zone du jeu « théorique » on veut afficher, et donc pouvoir faire sortir certains éléments de l’écran, une mécanique qui est d’ailleurs utilisé dans les combats de boss.

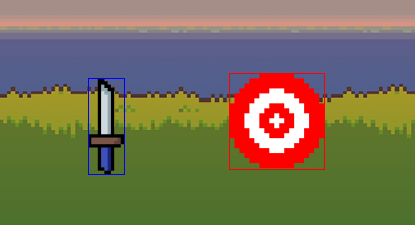
Cette façon de gérer les coordonnées est très pratique puisqu’elle permet d’afficher le jeu uniquement dans le terminal en se passant de l’affichage graphique. C’était très pratique lors du début du développement du jeu quand on n’avait pas encore d’interface graphique.

#### Gestion des collisions

La gestion des collisions se fait principalement en trois étapes :

La première étape consiste à créer un CollisionMask à partir de l’image pour générer un objet type Shape.

Pendant longtemps, on a gardé une Shape de type Rectangle :



Pour ensuite partir sur des formes plus complexes en utilisant un Area. Pour cela, on parcourt l’image et dès qu’on tombe sur un pixel qui n’est pas transparent, on l’ajoute à l’Area. Ce qui donne une hitbox plus détaillé :

Enfin, on utilise la méthode intersects() de java.awt.Shape qui retourne True si les deux Shape passées en paramètre sont en collision.

* Mise en avant des aspects algorithmiques importants du jeu.
* Discussion sur les techniques de programmation utilisées pour implémenter les fonctionnalités.

## BOUTIQUE DU JEU :

Afin d’enrichir l’expérience du joueur, une boutique du jeu a été mise en place. Elle constitue une composante essentielle offrant du contenu supplémentaire en termes de personnalisation.

Cette boutique a été implémentée dans le CardLayout via une interface utilisateur permettant une navigation fluide entre différentes catégories d’articles. Ainsi, elle est accessible depuis n’importe quelle composante du jeu. La boutique permet notamment de dépenser de la monnaie virtuelle, gagné en combattant les cibles du jeu, pour acheter des articles propres au jeu tel que des skins – couteaux, des images de fonds.

#### Fonctionnement de la boutique

Les joueurs peuvent acheter et s’équiper de skins de couteaux sans affecter leurs performances en jeu. Les couteaux proposés sont principalement à thème médiéval. Les fonds de jeu proposés dans la boutique ajoutent une expérience visuelle unique au joueur. Parmi ces fonds, on en trouve des paysages médiévaux et naturels, ajoutant ainsi une couche de personnalisation. Des musiques du jeu ont été ajoutées à la boutique du jeu, mais nous avons décidé de les intégrer comme récompenses du passe de combat.

#### Panier de la boutique

Les articles achetés sont directement ajoutés au panier de la boutique. Le joueur a accès aux informations des articles de son panier, tels que le nom, prix, et l’aperçu de celui-ci. Il peut également consulter le montant total de son panier. Le joueur a également la possibilité de supprimer cet article de son panier. De plus, après avoir acheté son panier – et s’il a assez de ressources disponibles pour cet achat, les nouveaux articles sont ajoutés à l’inventaire du jeu – composante du jeu détaillée plus tard.

Ainsi, par le biais de la boutique, le joueur est encouragé à s’investir dans le jeu afin de créer une ambiance qui leur est propre, et de personnaliser le plus possible son expérience dans celle-ci.

## INVENTAIRE DU JEU :

L’inventaire était en premier temps implémenté comme une composante du CardLayout, mais afin de fournir une expérience visuelle plus riche nous avons pris la décision de l’intégrer à la page d’accueil.

Il est codé dans la classe Game comme étant une liste d’objets de la boutique – List<ShopItem>. La classe *Game* permet de charger cette liste à partir de l’inventaire sérialisé en JSON qui se trouve dans la sauvegarde locale. Dans le cas où l’utilisateur ne possèderait pas d’inventaire, un cas où son inventaire est vide, alors la liste est initialisée avec deux articles par défaut : le couteau avec lequel il joue, et la musique du jeu.

#### Fonctionnement de l’inventaire

Comme évoqué précédemment, après achat d’un objet dans la boutique, celui-ci s’ajoute à la liste des *ShopItem,* c’est-à-dire l’inventaire.

Après chaque ajout, les observateurs sont notifiés, le but étant d’informer la page d’accueil qui implémente l’interface *LibraryObserver*, de cette mise à jour. Cela permet d’assurer l’affichage en temps réel des articles de l’inventaire sur la page d’accueil du jeu.

De plus, l’état actuel de l’inventaire est sauvegardé dans le fichier sérialisé, garantissant le fait que le joueur a toujours accès à son inventaire.

Gestion des niveaux :

Dans Hitty Knife - Redux, la progression du joueur est conçue autour d'une série de niveaux, chacun étant divisé en quatre rounds distincts. Cette structure est essentielle pour offrir une progression claire et mesurable au joueur, tout en augmentant le challenge et l'intérêt à chaque étape. Chaque niveau se compose de trois rounds normaux suivis d'un round de boss, orchestrés par la classe RoundManagement et détaillés par la classe Round.

*Composition des Rounds*

* **Rounds Normaux :** Les trois premiers rounds de chaque niveau sont conçus pour introduire progressivement les mécaniques du jeu et les types de cibles au joueur. Ces rounds utilisent une variété de cibles normales, mobiles, et bonus pour diversifier le gameplay et tester différentes compétences du joueur.
* **Round de Boss :** Le quatrième round de chaque niveau est un round de boss. Ce round est crucial, car il représente un défi majeur qui nécessite que le joueur applique les compétences et les stratégies apprises dans les rounds précédents. Les boss sont conçus pour être nettement plus difficiles et offrent une conclusion intense et gratifiante pour les niveaux.

*Fonctionnement technique des Rounds*

La classe Round gère la liste des cibles pour chaque round. La flexibilité de cette classe permet d'ajuster facilement le contenu de chaque round pour correspondre aux exigences du niveau en cours, qu'il s'agisse de rounds normaux ou de rounds de boss.

*Gestion des Cibles*

Chaque Round contient une liste de cibles List<Cible>, où chaque cible est spécifiée selon son type et son comportement. Cette liste est dynamiquement peuplée et gérée par RoundManagement pour refléter les défis spécifiques de chaque round.

*Rôle de RoundManagement*

La classe RoundManagement joue un rôle central dans la gestion des rounds et la structure des niveaux. Cette classe ne se contente pas seulement d'organiser la séquence des rounds, mais elle intègre également une logique complexe qui adapte le gameplay aux différents stades du jeu, assurant ainsi une expérience cohérente et stimulante pour le joueur.

*Gestion Dynamique des Rounds*

* Au début de chaque niveau, RoundManagement initialise une série de rounds à l’aide d’une liste de Round -List<Round>, en préparant chacun avec des cibles spécifiques adaptées aux défis du niveau. Cette préparation est faite en accord avec les spécifications du gameplay, assurant que chaque round contribue à la progression globale du joueur.
* La classe gère ensuite l'ordre dans lequel les rounds sont joués, avec un mécanisme pour passer au round suivant une fois le round actuel complété avec succès. Elle contrôle également les transitions entre les rounds normaux et le round de boss, augmentant la difficulté de manière appropriée pour culminer dans un défi significatif.

*Synchronisation avec la Classe Game*

La classe Game s'interconnecte profondément avec RoundManagement pour assurer que la progression à travers les rounds et les niveaux est non seulement suivie, mais aussi ajustée en fonction des actions du joueur.

Lorsque le jeu démarre ou lorsqu'un nouveau niveau commence, Game appelle RoundManagement pour initialiser et configurer les rounds selon le niveau actuel. Cette initialisation inclut la sélection des types de cibles et de leur configuration spatiale afin créer un ensemble cohérent de défis.

Ensuite Game reçoit et traite les entrées du joueur, telles que les succès ou échecs à atteindre les cibles. En fonction de ces résultats, Game peut demander à RoundManagement d'ajuster les rounds en cours, par exemple en réinitialisant le round actuel si le joueur perd toutes ses vies ou en passant au round suivant après avoir réussi à atteindre toutes les cibles des différends rounds du niveau.

## BOSS :

Afin d’améliorer l’expérience du joueur dans notre jeu, des boss sont générés et si on arrive à les tuer, nous pourrons passer de niveau. L’idée est d’ajouter une fonctionnalité que fait que le joueur a une difficulté en plus que les cibles normales.

Ces Boss ont été créés dans le package entity/bosses, chaque boss possède une logique de mouvement différente, l’implémentation fait donc que nous avons besoin d’une classe abstract qu’hérite Cible, que nous permettra de créer les méthodes que nous avons besoin dans chaque BossType.

#### Fonctionnement des boss

Nous avons défini que chaque boss possédera 3 vies, ce qui signifie que chaque boss devra être touché 3 fois par le couteau pour qu’il puisse être détruit et pour qu’on puisse passer de niveau.

L’idée est d’avoir des boss différents, qu’ont des mouvements différents et ajoutent une difficulté au joueur dans les deux modes de jeu.

Nous possédons 4 boss différents dans le jeu pour le moment, et ils apparaissent de manière aléatoire dans le dernier round de chaque niveau, ayant 1 seul par boss round.

#### La classe Boss

La classe boss, étant une classe abstract qu’étend la classe Cible, crée les fonctions que doivent être utilisées par chaque BossType, la classe principale pour chaque boss est la classe updateMovement(), c’est là qu’on trouve l’implémentation des mouvements pour chaque boss, leurs vies, et une méthode pour savoir si le boss est mort.

#### Les mouvements de chaque boss

BossType 1 est le boss que fait le mouvement le plus basique, dans sa classe, la fonction updateMovement (double delta) fait appel à la fonction verticalMovementWithHorizontalAdjustment (…) de la classe Geometry (où on place tous les mouvements du jeu), les mouvements qu’il fait sont d’aller du côté bas droit de l’écran vers le côté haut droit et dès qu’il sort de l’écran, il passera du côté haut gauche et descendra vers le côté bas gauche jusqu’à sortir de l’écran à nouveau et refait le même mouvement.

Une image contenant arbre, ciel, peinture, plein air

Description générée automatiquement

BossType 2 est le boss que fait le mouvement circulaire, la fonction updateMovement (…) fait appel à la fonction moveInCircle (…) de la classe Geometry, les mouvements qu’il fait sont tourner en rond, allant du bas de l’écran jusqu’à la partie du haut de l’écran, en répétant cela constamment.

Une image contenant capture d’écran, art

Description générée automatiquement

BossType 3 est le boss que fait le mouvement de gauche droite, la fonction updateMovement (…) fait appel à la fonction horizontalMovementWithVerticalAdjustment (…) de la classe Geometry, les mouvements qu’il fait sont en descendant de peu à peu jusqu’à arriver à une position Y spécifique, ce qui le réinitialise au début de son mouvement, il répète ces mouvements constamment.

Une image contenant capture d’écran, bougie, Jeu PC

Description générée automatiquement

BossType 4 est le boss que fait le mouvement le plus différent, la fonction updateMovement (…) fait appel à la fonction moveInPattern (…) de la classe Geometry, il commence en haut à droite, arrive à un certain X, ensuite descend jusqu’à un certain Y, repart à droite jusqu’à sortir de l’écran et pouvoir réapparaître de l’autre côté allant jusqu’à un certain X, monte jusqu’à un certain Y et repart à gauche jusqu’à sortir de l’écran et pouvoir réinitialiser son mouvement.

Une image contenant ciel, capture d’écran, Jeu PC, Compositing numérique

Description générée automatiquement

# Liste de fonctionnalités (double clic) :

Une image contenant texte, capture d’écran, nombre, Parallèle

Description générée automatiquement

Une image contenant texte, capture d’écran, document, Parallèle

Description générée automatiquement

# 

# Conclusion :

Le développement de « *Hitty Knife Redux* » a été une aventure riche en défis et en apprentissages, tant dans le plan humain que dans le plan académique. Ce projet a non seulement permis de créer un jeu en partant de rien - ce qui nous a amené à vite tenir à cœur à ce projet - mais il a également été l'occasion de développer et de concrétiser les compétences acquises depuis le début de notre cursus.

Les difficultés ont quant à elle su nous rassurer dans nos compétences, tant en Java que dans le travail d’équipe, car aucun problème n’a pas pu être résolu.

Durant ce projet, toutes les fonctionnalités clé ont été implémentées, reflétant le respect du cahier des charges, tout en intégrant d’autres éléments. Jamais « *Hitty Knife Redux* » n’a cessé de se réinventer. Les défis techniques et de conception rencontrés ont été, bien que parfois éprouvants, de précieuses opportunités pour apprendre et s'adapter, contribuant ainsi à un constant affinage de nos capacités.

L'exploration des pistes d'extensions futures pour « Heroes and Defenders » ouvre la voie à de nouvelles améliorations (malus dans le mode versus, typeCible devenant une sous-classe de cible) et à une envie partagée de continuer ce projet tout au long de notre licence.

Enfin, toute l’équipe souhaite remercier M. Yan Jurski pour son accompagnement et ses conseils tout au long du développement de ce projet, mais aussi vous Mesdames et Messieurs les Jurys pour l’attention portée à notre projet et notre rapport.