Yoann Paquette  
Catherine Beaudoin-Rheault  
Jimmy Tremblay-Bernier

Développement de jeu

**Here’s Teddy**Spécifications techniques

Travail présenté à  
Vahik Toroussian et Dominic Girard

Collège de Maisonneuve  
Le 04 octobre 2018

**Description technique du projet**

*Here’s Teddy* est un jeu du type *Metroidvania* en 2.5D développé pour ordinateur avec l’aide du moteur de jeu *Unity 3D*. Le jeu sera entièrement une expérience à un seul joueur, aucun multijoueur.

Le personnage principal, les ennemis, l’environnement et les objets seront modélisés par les membres de l’équipe. Pour ce qui est des animations, elles seront importées de sur *Mixamo*, créée à la main et importées d’une banque personnelle d’animation *MoCap* (“*Motion capture*”) qui sont libres de droits.

Un système de progression intermédiaire sera intégrée dans le jeu pour permettre au joueur d’avancer dans le jeu. Il pourra monter de niveau pour augmenter ses statistiques. Une série d’objets, d’améliorations et d’équipements lui seront disponibles à débloquer à travers le jeu. Un système de combat sera aussi implémenté, main dans la main, avec le système de progression. Il sera possible que les habiletés de combat soient modifiées selon le type d’équipement porté. Des techniques de mouvements variées (ruée avant/arrière, accroupie, saut, etc.) seront ajoutées pour donner plus de mouvements sur les axes verticaux et horizontaux.

Avec la disposition de l’environnement en 2.5D, les contrôles du joueur ne seront limités qu’à seulement deux axes, horizontaux et verticaux. L’environnement sera divisé en plusieurs salles/sections de chargement où seulement certains ennemis et/ou objets seront instanciés à la fois pour optimisation. Des points de sauvegardes manuelles seront accessibles dans certaines salles pour permettre au joueur de sauvegarder sa progression. Aucun système de sauvegarde automatique n’est prévu pour le jeu.

Pour la conception sonore du jeu, nous allons utiliser une immense collection de musiques et sons diverses que nous avons achetée sur l’*Asset Store* de *Unity*. Nous prévoyons d’ajouter des sons que nous allons créer par nous-même si la collection manque de ce que nous avons besoin.

L’ambiance du jeu, sonore comme visuelle, sera en transition constante au fur et à mesure que le joueur avancera dans le jeu. Elle commencera joyeuse et claire au début du jeu pour devenir de plus en plus sombre, obscure, instable et glauque. L’apparence du joueur et des ennemis sera aussi affectée par ces changements.

Un mini tutoriel sera affiché à l’écran dans la première salle du jeu pour permettre au joueur de se familiariser avec les contrôles de bases. Il pourra en tout temps aller les vérifier dans les options du menu de pause.

**Description des différents modules algorithmiques**

L'entièreté du système interne de scripts du jeu utilise un système d’événements pour gérer de façon la plus modulaire et détaché possible toutes les interactions entre les scripts du programme. Cela permettra une séparation et édition plus simple et rapide de chacune des fonctionnalités du jeu.

1. **Module du système d'événements**

D’une certaine façon, ce module est le moteur central de gestion de toutes les interactions entre la majorité des scripts. Le système d’événements sert de station de terminus où toutes les actions/événements sont reçues, triés et distribués à tous les scripts qui les attendent.

Un script n’a qu’à s’abonner (*subscribe*) à la liste d’un événement spécifique et attendre que cet événement soit déclenché (*triggered*). Quand le déclenchement a lieu, via un autre script, des données concernant l’événement sont envoyées au système. Elles sont par la suite récupérées par le ou les scripts concernés.

Le système d’événements est une entité statique unique qui est accessible par n’importe quel script du jeu.

1. **Module de l’interface**
   1. **Points de vie**

Les points de vie seront affichés sous la forme d’une barre avec le nombre de points précis restent affichés dessus. Il n’y aura aucun système de régénération des points de vie. Pour les récupérer, le joueur devra utiliser un objet, monté de niveau ou bien sauvegarder la partie.

* 1. **Expérience**

Une barre de progression pour atteindre le prochain niveau sera présente sur l’interface.

* 1. **Points de “Rage”**

La barre de rage indique si le joueur possède assez de points de rage pour engager le mode Rage. Le joueur gagne des points de rage en éliminant des ennemis et en récupérant certains objets.

* 1. **Objets secondaires**

À travers le jeu le joueur va pouvoir récupérer des objets secondaires qu’il pourra utiliser pour regagner des points de vie, de rage, des armes de lancer à utilisation unique, etc. Avec les touches U et I (clavier) ou LB et RB (manette), le joueur pourra défiler à travers les objets qu’il possède.

* 1. **Carte**

En appuyant sur la touche Tab (clavier) ou Back (manette), le joueur peut afficher en “*overlay*” une carte du jeu qui lui indique sa position actuelle.

1. **Module du joueur**
   1. **Mouvements**

En utilisant les touches WASD (clavier) ou la croix (manette) on peut déplacer le personnage de gauche à droite (AD), s’accroupir (S) et viser en l’air (W). En utilisant Espace (clavier) ou A (manette), il sera possible de sauter.

* 1. **Habilités**

Un “Dash” sera récupérable au fur du jeu ainsi qu’un grappin permettant d’atteindre certaines zones en hauteur.

1. **Module de Caméra**
   1. **Cinemachine**

Les caméras seront gérées par cinemachine (un plugin développé par unity pour faciliter la mise en place des caméras). La caméra restera centrée sur le joueur dans la majorité des cas. Celle-ci bloquera sur les bords de l’écran de jeu quand le personnage approchera d’une nouvelle zone de chargement.

1. **Module de Menu**
   1. **Options**

Une section permettant de régler divers paramètres tels que le son et quelques options graphiques.

* 1. **Contrôles**

Une section permettant de voir les contrôles au clavier et à la manette. Les contrôles ne seront pas modifiables en jeu directement.

* 1. **Charger une sauvegarde**

La section permettant de continuer sa partie précédemment commencer.

* 1. **Quitter**

La section permettant de quitter le jeu directement depuis l’intérieur du jeu.

1. **Module de chargement de scènes**

Les niveaux seront organisés en une espèce de tableau, mais pas directement en code. Nous utiliserons plutôt une nomination des prefabs nous permettant ainsi d’instancier un morceau de tableau grâce à la position du joueur. Le jeu sera séparé en plusieurs morceaux contenant plusieurs niveaux (prefabs) qui seront instanciés en lot dans le but d’éviter un chargement trop long. Le seul chargement souhaité sera lors du changement de morceaux.

1. **Module de combat**

Le personnage aura la possibilité d’utiliser jusqu’à 2 armes à 1 main ou une arme à 2 mains. Quand l’arme est à une main le joueur pourra choisir quelle arme est utilisé avec la touche correspondant à la main voulue. (ex: J = main gauche et K = main droite). Quand l’arme est à 2 mains, les mains vont être changées pour un coup rapide ou un coup plus lent mais plus puissant.

1. **Module d’inventaire**
   1. **Équipements**
      1. **Armes principales**

Les armes principales seront des armes de corps à corps relativement similaires entre elle-même. Par contre elles auront quelques effets différents entre les armes. Ex: plus grande allonge, plus grand *hitbox*, vitesse d’attaque, etc.

* + 1. **Armures**

Les armures pourront être récupérées et équipées dans un menu dédié. Par contre seules les statistiques seront modifiées, aucun changement physique.

* 1. **Objets secondaires**

Quelques objets pourront être trouvés tout au long du jeu assez régulièrement et aux mêmes endroits. Ceux-ci n’auront pas besoin d’être équipé pour être utilisé, mais seul un objet pourra être utilisé à la fois. On pourra changer facilement d’item sans aller dans le menu grâce à aux touches U et I au clavier et LB et RB à la manette. Leur nombre sera limité dans l’inventaire et leur nombre sera affiché directement dans l’UI du jeu.

* + 1. **Offensifs**

Ce sont des objets qui pourront être lancer de différentes façons. Certains objets iront en ligne droite, d’autres tomberont de façon oblique, etc.

* + 1. **Défensifs**

Certains objets permettent de reprendre de la vie ou de gagner un bonus temporaire.

1. **Module de progression**
   1. **Points d’expérience**

De l’expérience pourra être récupérée sur chaque ennemi tué. L’expérience variera en fonction du niveau de l’ennemi comparé au niveau du joueur. Plus le joueur sera un niveau élevé comparé à l’ennemi et moins l’expérience récupérée sera importante.

* 1. **Niveau du personnage**

Le personnage aura un niveau qui augmentera à force de tuer des ennemis. Lors de la montée de niveau, de la vie sera récupérée, des points de compétences seront attribués et la quantité d’expérience nécessaire pour monter de nouveau de niveau sera augmentée.

* 1. **Statistiques du personnage**
     1. **Constitution**

La constitution va permettre au joueur d’augmenter sa quantité maximum de points de vie. Celle-ci augmente avec la montée en niveau et avec l’équipement.

* + 1. **Force**

La force affecte directement la quantité d’attaque du joueur. Celle-ci augmente avec la montée en niveau et avec l’équipement.

* + 1. **Attaque**

L’attaque définit la quantité de dégâts infligés aux ennemis. Celle-ci augmente avec la force et avec les armes.

* + 1. **Défense**

La défense permet de réduire la quantité de dégâts que les ennemis infligent. Celle-ci augmente avec la montée en niveau et avec l’équipement.

* + 1. **Chance**

La chance est convertie en bonus aux chances de butin que les ennemis peuvent laisser à leur mort. Elle donne en plus un bonus à la quantité de points d’expérience gagnés de toutes les sources. Celle-ci augmente seulement avec l’équipement.

1. **Module de l’IA**
   1. **Finite State Machine (FSM)**

Grâce à l’*animator* de *Unity*, nous allons pouvoir créer une state machine avec des scripts qui pourront être interchangés en fonction de certains paramètres. Notamment la distance comparée au joueur, etc.

* 1. **État de patrouille**

Dans cet état, l’IA va se déplacer dans un chemin prédéterminé jusqu’à ce que le joueur s’approche.

* 1. **État de chasse**

Dans cet état, l’IA va tenter de se rapprocher du joueur en vue de l’attaquer.

* 1. **État de combat**

Dans cet état, l’IA est suffisamment proche du joueur et va donc tenter de lui asséner des coups jusqu’à ce que le joueur s’éloigne (retour à l’état de chasse), meurt ou que l’IA meurt.

1. **Module de butin**

En explorant le jeu, le joueur pourra trouver différents équipements comme des armes, armures et objets à utilisation unique. Ceux-ci pourront être trouvés dans certains endroits et d’autres auront une certaine probabilité d’être lâchés à la mort de certains ennemis.

1. **Module de sauvegarde**

Dans certaines salles du jeu il sera possible de sauvegarder la partie. On aimerait potentiellement sauvegarder le tout dans un fichier externe à *Unity* comme un *JSON* ou en local avec le plugiciel *PlayerPrefsX*.

1. **Module de sons**
   1. **Musique et ambiance dynamique**

La musique et l’ambiance seront dynamiquement modifiées à travers le jeu. De quelque chose de plus joyeux a plus lugubre à fur et à mesure que le joueur avance. Avec l’aide du mixeur interne de *Unity*, nous pourrons faire des transitions parfaites entre chaque section sonore du jeu. En plus de conserver un contrôle précis de chaque source audio dans des groupes audio (volume, *pitch*, effets audio, etc.).

* 1. **Sons ponctuels**

Un contrôle du son pour la majorité des actions du joueur et des ennemis sera aussi présent dans le mixeur et avec l’aide d’un gestionnaire de son. (attaque, saut, ruée, mort, dégât, etc.)