Documentation :

1) Présentation du jeu

Notre jeu est un platformer 2D d'évitement de type die and retry. A travers notre jeu, nous voulions souligner l'aspect du tryhard et du speedrun dans les jeux vidéo. En effet, nous avons pensé le jeu afin qu'il s'adapte à toute catégorie de joueurs. Une fois le jeu lancé, le joueur entre dans le menu principal, il y retrouve 4 niveaux :

- Le tutoriel, ici le joueur découvrira le jeu et les possibilités de gameplay qui s'offrent à lui afin d'avoir une prise en main plus facile
- Les niveaux 1 à 3, qui sont tous différents, avec leur propre univers. Le niveau de difficulté augmente au fur et à mesure des niveaux mais il y a toujours un chemin moins compliqué que les autres

a) Comportement du joueur attendu et possibilités

Le comportement attendu du joueur est de lancer le tutoriel afin de découvrir les différentes possibilités que le jeu lui offre. Une fois le tutoriel complété, il est téléporté dans le niveau numéro 1 afin de mettre en application les différentes possibilités de gameplay vues dans le tutoriel. Une fois terminé, le joueur passe directement au niveau suivant et ce jusqu'à terminer le 3ème et dernier niveau. S'il arrive jusque là, il est renvoyé vers le menu principal et peut constater le nombre de cerises qu'il a ramassé au total à travers les différents niveaux. Le joueur peut alors choisir de recommencer n'importe quel niveau, son compteur de cerises sera réinitialisé à 0. Il peut ainsi recommencer les niveaux afin de récupérer toutes les cerises, de tester toutes les "ways" possibles et compléter à 100% le jeu. A noter que, à n'importe quel moment, le joueur peut presser "echap" et recommencer les niveaux, quitter le jeu ou revenir au menu principal.

b) Fonctionnalités

Les fonctionnalités peuvent être séparées en plusieurs catégories : les fonctionnalités relatives au joueur, celles relatives aux ennemis, celles relatives aux objets et celles relatives à l'environnement.

i) le joueur

A l'aide des flèches directionnelles du clavier, le joueur peut se déplacer de gauche à droite. Avec la flèche du haut il peut sauter. Il peut avoir plusieurs inputs en même temps et donc changer de direction dans les airs par exemple.

Le joueur dispose d'un dash utilisable en pressant la touche "c" du clavier.
Ce dash le propulse rapidement dans la direction dans laquelle il regarde et le rend invulnérable pendant la durée de son dash.

Une mécanique de wall-jump est également disponible. En effet, lorsque le joueur est collé à un mur et qu'il presse la touche directionnelle en direction de ce dernier alors il s'y accroche. Si le joueur relâche la touche directionnelle alors il n'est plus accroché au mur.

Mais si le joueur presse la touche de saut alors un saut est effectué dans la direction opposée du mur auquel il est accroché.

ii) les ennemis

Les ennemis sont présents dans les différents niveaux. Ils peuvent tous tuer le joueur mais le joueur ne peut leur rendre la pareille. Il existe 4 ennemis différents :

- Les rhinocéros : Ils vadrouillent sur des surfaces planes allant d'un point A à un point B à une vitesse linéaire. Le joueur meurt s' il rentre en collision avec.

- Les abeilles : Elles sont dans les airs et peuvent détecter le joueur si il s'approche d'elles et se mettent à le suivre mais seulement sur l'axe horizontal tout en gardant toujours la même distance sur l'axe vertical (si le joueur saute l'abeille se déplacera en conséquence sur l'axe vertical). Elles tirent des projectiles de manière verticale. Le joueur meurt s'il rentre en collision avec. Les abeilles vont moins vite que le joueur, le joueur peut alors sortir de la zone de détection des abeilles et donc ne plus être poursuivi.

- Les chauves-souris : Elles se posent à des plafonds et dorment avant que le joueur ne rentre dans la zone de détection de ces dernières. Si il rentre dans cette zone, les chauves-souris se réveillent et se mettent à suivre le joueur. Le joueur meurt s'il rentre en collision avec. Les chauves-souris vont moins vite que le joueur, il peut alors sortir de leur zone de détection et donc ne plus être poursuivi. Mais attention, les chauves-souris traversent les murs.

- Les troncs d'arbre : Ils vadrouillent sur de surfaces planes allant d'un point A à un point B à une vitesse linéaire. Si ils détectent le joueur, ils s'arrêtent et se mettent à tirer des projectiles de manière horizontale vers la direction dans laquelle le joueur se trouve. Le joueur meurt s' il rentre en collision avec les projectiles ou l'ennemi lui-même.

iii) les objets

Les objets sont présents dans les différents niveaux. Ils apportent des bonus différents au joueur. Il existe 5 objets différents :

- Les cerises : Elles sont présentes dans tous les niveaux et peuvent être ramassées par le joueur. Elles servent à la complétion à 100% du jeu. Si le joueur veut terminer le jeu, il doit toutes les ramasser.

- Les pommes : Elles sont présentes dans tous les niveaux et peuvent être ramassées par le joueur. Les pommes permettent au joueur de sauter plus haut pendant 5 secondes.

- Les pastèques : Elles sont présentes dans tous les niveaux et peuvent être ramassées par le joueur. Elles font grossir le joueur et lui donnent une vie

supplémentaire. Si il est touché par un ennemi ou des pics, il perd une vie et revient à sa forme initiale.

- Les kiwis : Ils sont présents dans tous les niveaux et peuvent être ramassés par le joueur. Ils permettent au joueur de se déplacer plus rapidement pendant 5 secondes.

- Les bananes : Elles sont présentes seulement dans le tuto et dans le niveau 2 et peuvent être ramassées par le joueur. Elles permettent au joueur de devenir invincible pendant 5 secondes.

iv) L'environnement

L'environnement change au fur et à mesure des niveaux mais certains éléments sont présents dans chaque niveau. Nous pouvons compter 4 éléments :

- Les plateformes "sticky" : Présentes dans tous les niveaux, ces plateformes ralentissent le joueur et diminuent la hauteur de son saut.

- les "pics" : Présents dans plusieurs niveaux, ces plateformes enlèvent une vie au joueur. Si le joueur n'a pas ramassé de pastèque, il meurt en touchant cette plateforme.

- Les plateformes "tombantes" : Présentes dans plusieurs niveaux, ces plateformes disparaissent 1 seconde après que le joueur ait marché dessus.

- Les trophées : Présents dans tous les niveaux, en les touchant ils permettent au joueur de finir le niveau et de passer au suivant. Dans le menu principal, le joueur peut sélectionner un niveau en sautant sur un trophée.

2) Documentation technique

Nos services sont divisés en fonction des objets auxquels ils sont appliqués. Ainsi, un répertoire “Views” comprend tous les “Model”, “Controller”, “Prefab”, etc de l’objet auquel ils sont rattachés. De ce fait, notre projet respecte le pattern MVC.

a) View de la Chauve-souris

En entrant dans le dossier “Views” on descend l’arborescence et on trouve le dossier “BatView”.

i) BatView

Le répertoire “BatView” comprend un script “BatView” dans lequel est défini un integer “bat_state” qui permet de savoir quelle animation doit être appliquée à la chauve-souris. Ce répertoire contient aussi un répertoire “Common” qui lui-même comprend un dossier “Prefabs” dans lequel les prefabs en lien avec la chauve-souris sont contenus, à savoir, le prefab “Ennemy Bat”, ainsi qu’un dossier “BatAnimation” qui contient toutes les animations liées à la chauve-souris. Le dossier “Common” contient aussi un répertoire “Scripts” qui contient 3 autres dossiers “Controllers”, “Enums” et “Models”. Ces répertoires contiennent, respectivement, les scripts “BatController”, “BatEnums” et “BatModels”.

ii) BatEnums

Ce fichier ne comprend qu’une énumération qui donne les différentes possibilités d’animation soit : “Idle”, “Aware”, “Flying” et “GoBack”.

iii) BatModel

Ce fichier définit les différentes variables dont le script “BatController” a besoin pour fonctionner. Il les définit sans les initialiser car l’initialisation sera réalisée par “BatController”.

iv) BatController

Ce script commence par importer le model et la view définis plus haut. Grâce à ces importations, le script récupère toutes les variables dont il a besoin pour fonctionner. Il initialise ensuite les variables du model et de la view. Cela permet de passer à la logique même du script qui définit les conditions où la chauve-souris repère le joueur, chasse le joueur et retourne à son point de départ après que le joueur se soit trop éloigné. En fonction de l’état de la chauve-souris (joueur repéré, chasse du joueur, retour au point de départ) l’animation sera modifiée pour être en cohérence avec les mouvements de la chauve-souris. A noter que la chauve-souris sera tournée en cohérence avec le sens dans lequel elle va.

b) View du Rhinocéros

En entrant dans le dossier “Views” on descend l’arborescence et on trouve le dossier “RinoView”.

i) RinoView

Le répertoire “RinoView” comprend un script “RinoView” dans lequel est défini un integer “isRunning” qui permet de définir l’animation de course du rhinocéros. Ce répertoire contient aussi un dossier “Common” qui lui-même comprend un dossier “Prefabs” dans lequel les prefabs en lien avec le rhinocéros sont contenus, à savoir, le prefab “Ennemy Rino”, ainsi qu’un dossier “RinoAnimation” qui contient toutes les animations liées au rhinocéros. Le dossier “Common” contient aussi un répertoire “Scripts” qui contient 2 autres dossiers “Controllers” et “Models”. Ces répertoires contiennent, respectivement, les scripts “RinoController” et “RinoModel”.

ii) RinoModel

Ce fichier définit les différentes variables dont le script “RinoController” a besoin pour fonctionner. Il les définit sans les initialiser car l’initialisation sera réalisée par “RinoController”.

iii) RinoController

Ce script commence par importer le model et la view définis plus haut. Grâce à ces importations, le script récupère toutes les variables dont il a besoin pour fonctionner. Il initialise ensuite les variables du model. Cela permet de passer à la logique même du script qui définit les conditions où le rhinocéros patrouille.Un flip (retournement du sprite) est effectué en fonction de la direction dans laquelle se déplace le rhinocéros.

c) View du tronc

En entrant dans le dossier “Views” on descend l’arborescence et on trouve le dossier “TrunkView”.

i) TrunkView

Le répertoire “TrunkView” comprend un script “TrunkView” dans lequel est défini un integer “trunk_state” qui permet de savoir quelle animation doit être appliquée au tronc. Ce répertoire contient aussi un répertoire “Common” qui lui-même comprend un dossier “Prefabs” dans lequel les prefabs en lien avec le tronc sont contenus, à savoir, le prefab “Ennemy Trunk”, ainsi qu’un dossier “TrunkAnimation” qui contient toutes les animations liées au tronc. Le dossier “Common” contient aussi un répertoire “Scripts” qui contient 3 autres dossiers “Controllers”, “Enums” et “Models”. Ces répertoires contiennent, respectivement, les scripts “TrunkController”, “TrunkEnums” et “TrunkModels”.

ii) TrunkEnums

Ce fichier ne comprend qu’une énumération qui donne les différentes possibilités d’animation soit : “Idle”, “Run” et “Attack”.

iii) TrunkModel

Ce fichier définit les différentes variables dont le script “TrunkController” a besoin pour fonctionner. Il les définit sans les initialiser car l’initialisation sera réalisée par “TrunkController”.

iv) TrunkController

Ce script commence par importer le model et la view définis plus haut. Grâce à ces importations, le script récupère toutes les variables dont il a besoin pour fonctionner. Il initialise ensuite les variables du model et de la view. Cela permet de passer à la logique même du script qui définit les conditions où le tronc patrouille, tire sur le joueur et retourne à sa patrouille après que le joueur se soit trop éloigné. En fonction de l’état du tronc (patrouille ou tire sur le joueur) l’animation sera modifiée pour être en cohérence avec les mouvements du tronc. A noter que le tronc sera tourné en cohérence avec le sens dans lequel il va ou tire. Pour le projectile tiré, le script fait appel à un autre (“Bullet_trunk”) qui permet de donner un comportement au projectile.

d) View de l'abeille

En entrant dans le dossier "Views" on descend l'arborescence et on trouve le dossier "BeeView".

i) BeeView

Le répertoire "BeeView" comprend un script "BeeView" dans lequel est défini un integer "bee_state" qui permet de savoir quelle animation doit être appliquée à l'abeille. Ce répertoire contient aussi un répertoire "Common" qui lui-même comprend un dossier "Prefabs" dans lequel les prefabs en lien avec l'abeille sont contenus, à savoir, le prefab "Ennemy Bee", ainsi qu'un dossier "BeeAnimation" qui contient toutes les animations liées à l'abeille. Le dossier "Common" contient aussi un répertoire "Scripts" qui contient 3 autres dossiers "Controllers", "Enums" et "Models". Ces répertoires contiennent, respectivement, les scripts "BeeController", "Bullet", "BeeEnums" et "BeeModels".

ii) BeeEnums

Ce fichier ne comprend qu'une énumération qui donne les différentes possibilités d'animation soit : "Idle" et "Attack".

iii) BeeModel

Ce fichier définit les différentes variables dont le script "BeeController" a besoin pour fonctionner. Il les définit sans les initialiser car l'initialisation sera réalisée par "BeeController".

iv) BeeController

Ce script commence par importer le model et la view définis plus haut. Grâce à ces importations, le script récupère toutes les variables dont il a besoin pour fonctionner. Il initialise ensuite les variables du model et de la view. Cela permet de passer à la logique même du script qui définit les conditions où l'abeille tire sur le joueur, suit le joueur en étant toujours à la même distance verticale et reste à sa position après que le joueur se soit trop éloigné. En fonction de l'état de l'abeille (reste en position ou tire sur le joueur) l'animation sera modifiée pour être en cohérence avec les mouvements de l'abeille. A noter que l'abeille sera tournée en cohérence avec le sens dans lequel elle va ou tire. Pour le projectile tiré, le script fait appel à un autre ("Bullet") qui permet de donner un comportement au projectile.

e) View du personnage

En entrant dans le dossier “Views” on descend l’arborescence et on trouve le dossier “PlayerView”.

i) PlayerView

Le répertoire “PlayerView” comprend un script “PlayerView” dans lequel est défini un integer “player_state” qui permet de savoir quelle animation doit être appliquée au player. Ce répertoire contient aussi un répertoire “Common” qui lui-même comprend un dossier “Prefabs” dans lequel les prefabs en lien avec le player sont contenus, à savoir, le prefab “Player”, ainsi qu’un dossier “PlayerAnimation” qui contient toutes les animations liées au player. Le dossier “Common” contient aussi un répertoire “Scripts” qui contient 3 autres dossiers “Controllers”, “Enums” et “Models”. Ces répertoires contiennent, respectivement, les scripts “PlayerMovement”, “PlayerLife”, “SceneManagement” et “ItemCollector” “PlayerEnums” et “PlayerModels”.

ii) PlayerEnums

Ce fichier ne comprend qu’une énumération qui donne les différentes possibilités d’animation soit : “Death”, “Falling”, “Hanging”, “Idle”, “Jumping” et “Running”.

iii) PlayerModel

Ce fichier définit les différentes variables dont les scripts présents dans le dossier “Controller” ont besoin pour fonctionner. Il les définit sans les initialiser car l’initialisation sera réalisée par les scripts du dossier “Controller”.

iv) Player Life

Ce script permet de gérer tout ce qui est relatif à la vie du joueur. C’est à dire s’il prend des dégâts, s’il est invincible ou s’il meurt. Par défaut, le joueur n’a qu'une seule vie, gérée par la variable lives, initialisée à 1. Cette variable va ensuite être impactée par une multitude d’objets en cas de collision. Ainsi, entrer en collision avec une pastèque donnera au joueur une vie supplémentaire (visible en jeu par un grossissement du personnage), alors qu’entrer en collision avec un ennemi ou un piège (les pics) lui fera perdre une vie. Deux cas sont alors possibles :

-Le joueur n’a qu’une vie et meurt (animation de mort et reprise du niveau à son début)

-Le joueur a deux vies, il en perd une et devient invincible pendant une seconde pour éviter de prendre des dégâts en continu

L’invincibilité est visible via un clignotement léger pour que le joueur comprenne qu’il a pris un dégât et qu’il possède un petit temps pour sortir d’une situation mortelle.
Enfin, le dernier objet géré par ce script en cas de collision est la banane qui rend le joueur invincible pendant 5 secondes toujours accompagné par le clignotement caractéristique de l’impossibilité de prendre des dégâts.

v) Player Movement

Script très complet qui gère tous les comportements du joueur liés aux mouvements. Par défaut le joueur possède une vitesse de déplacement initialisée à 4 et une hauteur de saut initialisée à 7. Le saut ne peut être activé que sur les zones considérées comme du sol définies dans ce script. Le joueur doit être au sol, ne pas être en chute, etc.
De plus, ce script gère aussi le dash et fait appel au script Player Life pour rendre le joueur invulnérable pendant la durée du dash. La mécanique de wall jump qui permet de déterminer quand un joueur est en position de faire un wall jump et la direction dans laquelle il sera propulsé est aussi gérée.
Chaque mouvement ou saut à sa propre animation gérée via ce script, l'immobilité est aussi animée.
Enfin, le comportement du personnage en fonction de la texture avec laquelle il est en texture est aussi géré via ce script. Ainsi, c'est ici que l'on détermine à quel point le joueur est ralenti sur un sol dit "sticky" et le fait qu'il ne puisse sauter que très peu haut. La notion de plateformes fragiles est aussi définie ce qui permet d'indiquer la durée avant que la plateforme ne disparaisse.

vi) Item Collector

Dans ce script sont définis les effets des objets lorsque le joueur entre en collision avec. C'est ici qu'est géré le compteur du nombre de cerises ramassées lors du niveau mais aussi le fait que prendre un kiwi permet d'augmenter la vitesse via un appel au script "PlayerMovement" et la modification des variables "MovementSpeed" et MovementSpeedInit. De la même manière, la prise d'une pomme modifie les variables "JumpForce" et "JumpForceInit". Ces améliorations étant provisoires, le script remet à leurs valeurs initiales les variables de saut et de vitesse au bout de 5 secondes

vii) Scene Management

Ce dernier script permet de gérer les changements de scène en fonction des objets avec lesquels le joueur rentre en collision. Ainsi, dans la scène du menu principal, le joueur peut toucher le trophée "Tuto", "Level 1", "Level 2" ou "Level 3" ou l'objet "Quit" et la scène sera changée en adéquation avec l'objet touché. A noter que pour l'objet "Quit" seul un message est affiché, il n'y a pas de changement de scène. Dans chaque niveau, toucher le trophée situé à la fin permet de passer au niveau suivant. Le trophée du 3ème et dernier niveau, permet, quant à lui, de revenir au menu principal. C'est aussi dans ce script qu'est géré le comportement du compteur de cerises entre chaque scène, pour que le compteur ne reparte pas de 0 à chaque niveau.

f) Autres scripts

Ces scripts ne sont pas liés à des fonctionnalités majeures ils n'ont donc pas de dossier "View" à leur nom. Ils sont simplement disponibles dans le dossier "Scripts" à la racine du projet.

i) CameraController

Script qui fait bouger la caméra en fonction de la position du joueur. La caméra est fixée sur le joueur mais il y a un offset qui permet de voir un peu plus bas qu'une caméra classique.

ii) Finish

Script qui permet de détecter si le joueur entre en collision avec le trophée. Si c'est le cas, alors la fonction "CompleteLevel()" est appelée et permet au joueur de quitter le niveau. Un autre script prend le relais pour lancer le niveau suivant si nécessaire.

iii) MainMenu

Script qui permet de charger la première scène et de charger les scènes suivantes mais aussi de faire quitter le jeu en cas de lancement via un build Microsoft ou Mac.

iv) MenuScript

Script qui gère le menu pause, il détermine la touche qui permet d'activer la pause, en l'occurence, "échap". Il gère aussi les actions en fonction du bouton sur lequel le joueur clique :

- Si le joueur clique sur "Reprendre" le jeu reprend là où le joueur avait mis en pause
- Si le joueur clique sur "Retour QG" le jeu retourne sur le menu principal
- Si le joueur clique sur "Quitter" le jeu se fermera s'il est lancé via un build Microsoft ou Mac

g) Autres prefabs

Ces prefabs ne sont pas liés à des fonctionnalités majeures, ils ne nécessitent pas de MVC, donc ils ne sont pas dans un dossier "View". Ils sont simplement disponibles dans le dossier "Prefabs" à la racine du projet. Ces prefabs voient leur comportement définis dans d'autres scripts comme les cerises ou les kiwis dont le comportement est défini dans "ItemCollector". Ces prefabs sont, néanmoins, utilisés dans presque tous les niveaux et donc essentiels au jeu et à son gameplay.