



CentraleSupélec

Projet jeu MarioCS

PROJET DES CODING WEEKS
1 ANNÉE CYCLE INGÉNIEUR

Du 13/11/2023 au 24/11/2023



CentraleSupélec

Etudiants :

YOUNES-JIHAD BOUMOUSOU
ZAKARIA BHEDDAR
IMAD AIT BEN SALEH
YASSINE MESSIOUI
MOHAMMED YASSINE BIHI
ADAM EL HACHIMI

Résumé

Ce projet a été le résultat d'un travail d'équipe de six étudiants de 1 Année en Cycle Ingénieur à CentraleSupélec au cours des deux semaines de Coding Weeks.

Au croisement de la tradition et de l'innovation, notre équipe est fière de présenter ce rapport détaillé sur le développement de notre jeu captivant et original inspiré du légendaire Mario, mais imprégné de notre culture marocaine. Notre projet émerge comme une fusion passionnante entre l'univers emblématique des jeux de plateforme et la vibrante palette d'influences marocaines, capturant l'esprit dynamique et diversifié de ce pays.

À travers ce rapport, nous partagerons les différentes étapes de conception, de développement et d'implémentation qui ont donné vie à notre création ludique. De la genèse de l'idée à la réalisation concrète du jeu, nous explorerons les défis rencontrés, les choix artistiques et techniques, ainsi que les moments de collaboration qui ont nourri notre progression en tant qu'équipe soudée.

Nous invitons les lecteurs à plonger dans ce périple créatif, où chaque détail, chaque personnage et chaque niveau ont été méticuleusement pensés. À travers cette expérience interactive, nous espérons offrir non seulement un divertissement, mais aussi une immersion dans les joyaux artistiques et traditionnels du Maroc.

Nous vous convions à explorer les coulisses de notre aventure, où l'ingéniosité rencontre la tradition. Bienvenue dans l'univers unique de notre jeu, où le patrimoine rencontre la technologie pour créer une expérience inoubliable, pour nous en tout cas.

Table des mati res

1	Fonctionnalit� : Cr�ation de l'univers de jeu	4
1.1	Etape : Cr�ation d'une interface de jeu	4
1.2	Etape : Cr�ation d'un personnage principal	4
1.3	Etape : Cr�ation d'une Carte de jeu simple sans animation (Map) . .	4
2	Fonctionnalit� : Affichage du jeu	4
2.1	Etape : Affichage de l'univers du jeu	4
3	Fonctionnalit� : Mouvements dans l'univers de jeu	5
3.1	Etape : Mouvements du joueur	5
3.2	Etape : Contr�le des mouvements du joueur	5
4	Fonctionnalit� : Cr�ation d'un niveau de jeu	5
4.1	Etape : Impl�mentation du niveau de jeu et de la cam�ra	5
4.2	Etape : D�finition des collisions entre entit�s	6
4.3	Etape : Connaissance de l'�tat du joueur	6
4.4	Etape : Compl�tion des propri�t�s de mise � jour et de lancement . .	6
5	Fonctionnalit� : Impl�mentation des ennemies	6
5.1	Etape : Impl�mentation des ennemies	6
5.2	Etape : Mouvement des ennemies	7
6	Fonctionnalit� : Interface de jeu et graphismes	7
6.1	Etape : Dessin, graphismes et leur accessibilit� lors du lancement . .	7
6.2	Etape : Affichage des dessins sur l'aire de jeu	7
6.3	Etape : Inversion des images	7
7	Fonctionnalit� : Quelques d�tails graphiques	8
7.1	Etape : Impl�mentation de la pouss�re d�e au mouvement	8
7.2	Etape : Animation des particules de la pouss�re	8
8	Fonctionnalit� : L'audio du jeu	8
8.1	Etape : Choix des sons	8
8.2	Etape : Int�gration du son au jeu	9
9	Fonctionnalit� : Configuration du jeu	9
9.1	Etape : Organisation des composantes du jeu	9
9.2	Etape : Victoire, E�ec et Game-Over	9
10	Fonctionnalit� : Boutons de l'interface <i>Menu</i>	10
10.1	Etape : D�finition des boutons	10
10.2	Etape : Affichage des boutons	10

11	Fonctionnalité : Attaques par projectiles	10
11.1	Etape : Définition d'un projectile	10
11.2	Etape : Lancée des tajines	10
12	Fonctionnalité : Quelques dernières fonctionnalités et leur affichage	11
12.1	Etape : Niveaux de vie	11
12.2	Etape : Interaction avec l'ennemi	11
12.3	Etape : Collection de pièces	11
12.4	Etape : Score du jeu	11

1 Fonctionnalit  : Cr ation de l'univers de jeu

1.1 Etape : Cr ation d'une interface de jeu

- Mise en place d'une interface utilisateur de jeu intuitive et fonctionnelle.
- Cr ation d'une boucle de jeu dans un fichier *main.py* qui m ne   un menu de jeu.
- Cr ation d'un fichier *settings.py* qui va contenir au cours du d veloppement du jeu : les param tres de boucle, d'interface graphique, du joueur et les variables de jeu.

1.2 Etape : Cr ation d'un personnage principal

- Cr ation du fichier *player.py* pour impl menter le personnage principal.
- Concevoir un personnage basique sans mouvements anim s ni d'apparence graphique.
- Cr ation de la classe *Player* ayant les principales attributs *x* et *y* indiquant la position.

1.3 Etape : Cr ation d'une Carte de jeu simple sans animation (Map)

- Cr ation d'une premi re structure de la carte *Tile* ayant les attributs *size* et *x, y* indiquant la position.
- Cr ation de sous-classes de la classe *Tile* nomm es *StaticTile*, *AnimatedTile* et enfin une sous-classe de *StaticTile* nomm e *Crate*. Ce sont des structures soit statiques ou anim es qui constituent l'environnement de jeu.

Commentaires et R sultats :

Nous avons jusqu'ici r ussi   d velopper un univers de jeu basique avec une boucle de jeu mais sans interface graphique, ni de mouvements, ni d'option d'affichage.

2 Fonctionnalit  : Affichage du jeu

2.1 Etape : Affichage de l'univers du jeu

- L'utilisation de la librairie *pygame* par l'interm diaire de plusieurs fonctions afin de donner un rendu graphique du code impl ment  jusqu'  pr sent.

Commentaires et R sultats :

Nous poss dons   pr sent une interface graphique qui permet de visualiser le jeu et ses diff rentes structures et entit s.

3 Fonctionnalit  : Mouvements dans l'univers de jeu

3.1 Etape : Mouvements du joueur

- Cr ation de l'horloge du jeu gr ce   *pygame* pour introduire la notion du temps au jeu.
- D finition de la direction du joueur gr ce   la commande *pygame.math.Vector2(0, 0)* par laquelle nous pouvons faire mouvoir le joueur au sein de l'environnement.
- Application de la gravit    tout instant par la fonction *apply_gravity* en agissant sur la direction du joueur vers le bas.
- Impl mentation de la fonction *jump* pour effectuer des sauts dans l'environnement de jeu.
- Impl mentation des mouvements de base vers la gauche et la droite et concevoir la possibilit  de superposer ces mouvements avec les sauts.

3.2 Etape : Contr le des mouvements du joueur

- Impl mentation de la fonction *entree_joueur* qui permet de recevoir les commandes de l'utilisateur sur l'interface homme-machine pour contr ler les mouvements du joueur.

Commentaires et R sultats :

Nous poss dons actuellement un univers de jeu o  le joueur peut se mouvoir gr ce aux commandes de l'utilisateur dans tout l'air du jeu.

4 Fonctionnalit  : Cr ation d'un niveau de jeu

4.1 Etape : Impl mentation du niveau de jeu et de la cam ra

- Cr ation d'une classe *Level* ayant comme attributs *screen*, c'est   dire l' cran d'affichage et *level_data* qui sont les variables et propri t s de jeu propres au niveau.
- Configuration du joueur par la m thode *player_setup*.
- Cr ation du niveau de jeu par un ensemble de *Tile* par la m thode *create_tile_group*.

- Impl mentation de la cam ra par la fonction *bordure_x* qui d termine les bordures selon xx' o  le joueur est visible pour que la cam ra le suive.

4.2 Etape : D finition des collisions entre entit s

- D finition des collisions entre les entit s du jeu (principalement entre *Player* et *Tile*) par les m thodes *collision_horiz* et *collision_ver* pour les collisions verticales et horizontales.
- D finition de l' tat standard du joueur comme  tant debout sur terre par la m thode *get_player_on_ground*.

4.3 Etape : Connaissance de l' tat du joueur

- D finition de la m thode *etat_joueur* qui permet de mettre   jour son  tat   chaque instant (*jumping*, *falling*, *idle*, *running*) selon les commandes entr es.
- Impl mentation de la m thode *import_character_assets* pour pouvoir   chaque instant acc der   l' tat du joueur.

4.4 Etape : Compl tion des propri t s de mise   jour et de lancement

- D finition des m thodes *update* dans les fichiers *player.py*, *tiles.py* pour mettre   jour leur  tat de mani re permanente et *run* dans *level.py* pour lancer le niveau de jeu.

Commentaires et R sultats :

Jusqu'ici, nous avons r ussi   cr er un univers de jeu plus r aliste physiquement gr ce aux collisions, nous avons aussi la possibilit  de visualiser notre joueur ind finiment gr ce   la cam ra. De plus, en connaissant l' tat du joueur nous pourr  corriger plusieurs fonctions et m thodes d finies pr c demment en ajoutant des conditions sur l' tat du joueur (*Exemple* : Ne pas pouvoir effectuer de sauts lorsqu'on est d j  dans une phase de saut ou de tomb e).

5 Fonctionnalit  : Impl mentation des ennemies

5.1 Etape : Impl mentation des ennemies

- Au cours de cette  tape, nous cr ons un nouveau fichier nomm  *enemy.py* ou l'on impl mente l'objet *Enemy* ayant les attributs *size*, *x* et *y* de position.

- Impl mentation de la m thode *collision_enemy* qui permet de d finir les collisions entre le joueur principal et les ennemies dans le fichier *level.py*.

5.2 Etape : Mouvement des ennemies

- D finition des m thodes *move* pour mouvoir les ennemies.
- Impl mentation des m thodes *reverse* qui permet d'inverser le vecteur vitesse de l'ennemi lorsqu'il fait des collisions avec des *Tiles* invisibles, ceci cr e un effet de va-et-vient permanent.
- Impl mentation de la m thode *update* essentielle pour mettre   jour l'ennemi en permanence.

Commentaires et R sultats :

Ainsi, nous avons d finis les ennemies sur l'espace de jeu, ce qui le rend plus interactif et plus captivant.

6 Fonctionnalit  : Interface de jeu et graphismes

6.1 Etape : Dessin, graphismes et leur accessibilit  lors du lancement

- Au cours de cette fonctionnalit , on fournit d'abord des dessins des diff rents composantes graphiques du jeu et on les anime en utilisant le logiciel *libresprite*.
- Importation de ces images dans le r pertoire du projet sous forme de fichiers csv, puis on cr e un nouveau fichier *path.py* qui contient les diff rents chemins d'acc s de toutes ces fichiers et la fonction *get_path* qui re oit le chemin et retourne les fichiers.

6.2 Etape : Affichage des dessins sur l'aire de jeu

- Cr ation d'un fichier *traitement_csv.py*, o  l'on impl mente les fonctions suivantes *import_csv_layout* et *import_cut_graphics* qui permettent de mettre les images et dessins sur l'aire de jeu gr ce   la librairie *pygame*.
- Regroupement des donn es sur les niveaux de jeu, les fichier csv, et les fichiers de l'interface graphique dans un nouveau fichier *game_data.py*.

6.3 Etape : Inversion des images

- Enfin, une derni re  tape consiste   renverser les images pour les d placements en direction inverse. En effet, sachant que les images sont d finies

pour les déplacements vers la droite, nous utilisons la fonction prédéfinie `pygame.transform.flip` pour inverser l'image vers la gauche.

Commentaires et Résultats :

Nous pouvons maintenant visualiser un univers de jeu graphiquement développé et animé, l'aire de jeu n'est plus basique et abstraite mais plus proche à la compréhension de l'utilisateur et est plus intuitive.

7 Fonctionnalité : Quelques détails graphiques

7.1 Etape : Implémentation de la poussière dûe au mouvement

- Définition de l'objet *Particle* ayant comme attributs *position*, et *type* de mouvement pour lequel la poussière est dûe dans un nouveau fichier *particle.py*.
- Implémentation des fonctions *import_dust_run_particles* pour l'importation de l'animation de la poussière et *run_dust_animation* pour afficher l'animation sur l'écran de jeu. Nous implémentons ces deux fonctions dans le fichier *player.py*.
- Implémentation de la méthode *jump_particles* et *create_landing_dust* dans le fichier *level.py* pour importer et afficher la poussière dûe aux sauts et atterrissages.

7.2 Etape : Animation des particules de la poussière

- Implémentation des méthodes *animate* et *update* pour animer le display de ces particules selon l'état du joueur et mettre à jour en permanence leur type et affichage.

Commentaires et Résultats :

Nous avons dans cette partie ajouté des détails d'animation qui rajoute un aspect plus réaliste sur l'animation.

8 Fonctionnalité : L'audio du jeu

8.1 Etape : Choix des sons

- Dans un premier temps, nous choisissons les sons propres à chaque action exécutée durant le jeu, ainsi que la musique du jeu que l'on importe dans le répertoire du projet.

8.2 Etape : Intégration du son au jeu

- Création d'un nouveau fichier *audio.py*, et grâce au *mixer* de *pygame* nous développons la fonction *music* qui permet de gérer les sons au cours de la partie de jeu.

Commentaires et Résultats :

Maintenant que le son est intégré, le jeu paraît encore plus réaliste et plus intuitif et interactif.

9 Fonctionnalité : Configuration du jeu

Motivation :

Définir un *welcome_menu* de jeu qui donne une idée générale sur le jeu et offre la possibilité de jouer sur plusieurs niveaux. Il doit aussi détecter les échecs et victoires du joueur pour lancer le *game - over*

9.1 Etape : Organisation des composantes du jeu

- Création d'un fichier *setup.py* où on définit les méthodes suivantes : *main*, *welcome_menu* et *levels* pour implémenter les fonctionnalités ci-dessus.

9.2 Etape : Victoire, Echec et Game-Over

- Implémentation de la fonction *win_situation* dans le fichier *level.py* qui retourne l'état de victoire lorsque le joueur a gagné.
- Implémentation de la fonction *game_over* pour pouvoir relancer le jeu une fois le joueur est mort ou a gagné que l'on peut détecter par l'intermédiaire des fonctions *death* et *win_situation*.

Commentaires et Résultats :

Nous avons ainsi implémenté une interface de bienvenue au jeu, et donc le jeu et ses composantes sont plus organisées et mieux mises en avant, cependant il nécessite des boutons pour pouvoir agir sur cette interface nouvelle.

10 Fonctionnalité : Boutons de l'interface *Menu*

10.1 Etape : Définition des boutons

- Implémentation de la classe *Button* dans un nouveau fichier *button.py* à laquelle on attribut la position x , y , *image* et *scale*.

10.2 Etape : Affichage des boutons

- Définition de la méthode *draw* qui affiche le bouton dans sur l'écran lorsqu'on l'appelle.

Commentaires et Résultats :

Nous pouvons à présent agir sur le Menu de démarrage du jeu, l'interface est plus simple et plus intuitive, et le jeu est mieux organisé et bien présenté.

11 Fonctionnalité : Attaques par projectiles

11.1 Etape : Définition d'un projectile

- On se propose de considérer comme projectile qui rentre dans le thème choisi initialement un *tajine* marocain.
- Implémentation de la classe *Tajine* dans un nouveau fichier *tajine.py* ayant les attributs x , y et *direction* du projectile.

11.2 Etape : Lancée des tajines

- Définition de la méthode *shoot* qui permet de lancer le tajine dans la direction donnée.
- Implémentation de la méthode *update* qui prend comme argument aussi le *deplacement* pour mettre à jour l'état du projectile.

Commentaires et Résultats :

Nous avons ainsi implémenté les projectiles, et leur animations, cependant, nous n'avant pas encore codé l'effet de ces projectiles sur les ennemis.

12 Fonctionnalité : Quelques dernières fonctionnalités et leur affichage

12.1 Etape : Niveaux de vie

- Définition des niveaux de vie *health* et de la méthode *death* dans le fichier *level.py*, qui retransche au joueur un niveau de vie s'il touche des ennemis ou tous ses points s'il tombe des l'aire de jeu et qui donne au joueur le statut de mort si tous ses points de vie ont été retirés.
- Implémentation de la méthode *update* qui prend comme argument aussi le *deplacement* pour mettre à jour l'état du projectile.
- Implémentation de la fonction *show_health* qui affiche les niveaux de vie du joueur dans un nouveau fichier *gui.py*.

12.2 Etape : Interaction avec l'ennemi

- Définition de la fonction *hit_enemy* dans le fichier *level.py* qui donne au joueur la possibilité de tuer des ennemis en leur sautant dessus.
- Implémentation de la fonction *collision_tajine* qui permet d'attaquer des ennemis par les projectiles de tajines.
- Implémentation de la fonction *show_tajine* qui affiche le nombre de projectiles *tajines* que le joueur a en sa possession.

12.3 Etape : Collection de pièces

- Nous définissons les pièces *Coin* comme des *AnimatedTile* auxquelles on associe une animation au sein du jeu.
- Implémentation de la fonction *hit_coin* dans le fichier *level.py* qui permet de supprimer les pièces de l'espace de jeu lorsque le joueur les touche.
- Implémentation de la fonction *show_coins* qui affiche le nombre de pièces collectées.

12.4 Etape : Score du jeu

- Création d'un nouveau fichier *read_score* où l'on va implémenter les différentes fonction qui permettent de calculer le score et de l'afficher.
- Implémentation de la fonction *change_score* qui à partir d'un score initial, calcule les changements du score à chaque instant et événement dans le jeu.
- Implémentation de la fonction *read_highscore* qui donne le meilleur score déjà fait et le change lorsqu'un nouveau a été réussi.

Commentaires et Résultats :

Nous avons ainsi achevé le développement de notre jeu. Les éléments l'environnement interagissent bien entre eux, et toutes les fonctionnalités s'exécutent en harmonie et sans bug.

Conclusion

Au fil de ce rapport, nous avons partagé notre périple créatif, détaillant les étapes allant de l'idéation à la concrétisation du jeu, mettant en lumière les défis surmontés, les choix artistiques et techniques, ainsi que les moments de collaboration qui ont renforcé notre cohésion en tant qu'équipe.

Nous invitons chaleureusement les lecteurs à plonger dans les coulisses de cette aventure, où chaque détail, personnage et niveau ont été soigneusement conçus. Cette expérience interactive vise à offrir bien plus qu'un divertissement, elle aspire à immerger les participants dans l'innovation et la créativité.

—Fin du rapport—