Unshaped

« Création d’un jeu de survie 2D avec le moteur de jeu Godot. »

Par Vasco Valadares Semana – En 2023 – Projet Personnel

Table des matières

1. Introduction

Unshaped est un jeu d'action 2D de type "survival" où le joueur incarne un personnage de forme ronde. L'objectif est de survivre le plus longtemps possible face à des vagues d'ennemis aux formes géométriques. Pour y parvenir, le joueur peut tirer des projectiles, collecter des améliorations (power-ups) et augmenter les statistiques de son personnage afin de vaincre des adversaires de plus en plus redoutables.

2. Objectifs du Projet

2.1 Contexte de Développement

Le projet Unshaped est un jeu indépendant complet, développé dans le but de d'apprendre le moteur Godot ainsi que des mécaniques de jeu classiques telles que la survie, les systèmes d'amélioration et la gestion de score.

2.2 Objectifs Principaux

- **Création d'un Jeu de Survie :** Développer une expérience de jeu captivante avec des mécaniques de progression claires pour le joueur ;
- **Implémentation d'un Système d'Améliorations Complet :** Permettre au joueur d'améliorer les capacités de son personnage (attaque, vie, vitesse) ;
- **Développement d'un Gameplay Équilibré :** Concevoir des types d'ennemis variés avec des comportements distincts et une difficulté progressive ;
- **Fourniture d'une Expérience Utilisateur Soignée :** Offrir une interface utilisateur intuitive, des effets visuels et des retours clairs pour le joueur.

3. Architecture du Jeu

3.1. Style Architectural et Structure

Le jeu Unshaped est construit sur une architecture orientée scènes. Il utilise le pattern Singleton pour la gestion globale des paramètres du jeu (GameSettings via AutoLoad). Le projet adopte une approche modulaire avec une séparation des responsabilités entre les différents scripts et scènes.

3.2. Modules et Composants Clés

Le projet est organisé en plusieurs répertoires principaux contenant les scripts et scènes :

- **Scripts/Characters/ et Scenes/Characters/ :** Gestion des entités de jeu (Joueur, Ennemis) ;
- **Scripts/UI/ et Scenes/UI/ :** Logique et scènes de l'interface utilisateur (Menus, HUD) ;
- **Scripts/Collectibles/ et Scenes/Collectibles/ :** Objets ramassables et power-ups ;
- **Scripts/AutoLoads/ :** Singletons globaux pour les paramètres et la sauvegarde ;
- **Scripts/Projectiles/ :** Logique des projectiles ;
- **Scripts/GameProcess.gd :** Contrôleur principal de la logique de jeu.

3.3. Fichiers Essentiels et Rôles

- **Point d'Entrée Principal :** Scenes/MainMenu.tscn est la scène lancée au démarrage du jeu ;
- **Configuration du Projet :** project.godot contient les configurations globales du moteur Godot (inputs, autoloads, résolution) ;
- **Contrôleur de Jeu :** Scripts/GameProcess.gd orchestre les mécaniques de jeu principales (apparition d'ennemis, score, phases de difficulté) ;
- **Gestion des Données Persistantes :** Scripts/AutoLoads/GameSettings.gd gère la sauvegarde et le chargement des scores et paramètres.

3.4. Choix Technologiques

- **Moteur de Jeu :** Godot Engine 3.5.3 ;
- **Langage de Programmation :** GDScript (langage natif de Godot) ;
- **Systèmes Intégrés Godot :** Area2D pour les collisions, AnimationPlayer pour les animations, Timer pour la gestion temporelle, Input Map pour les contrôles.

4. Gestion et Structure des Données

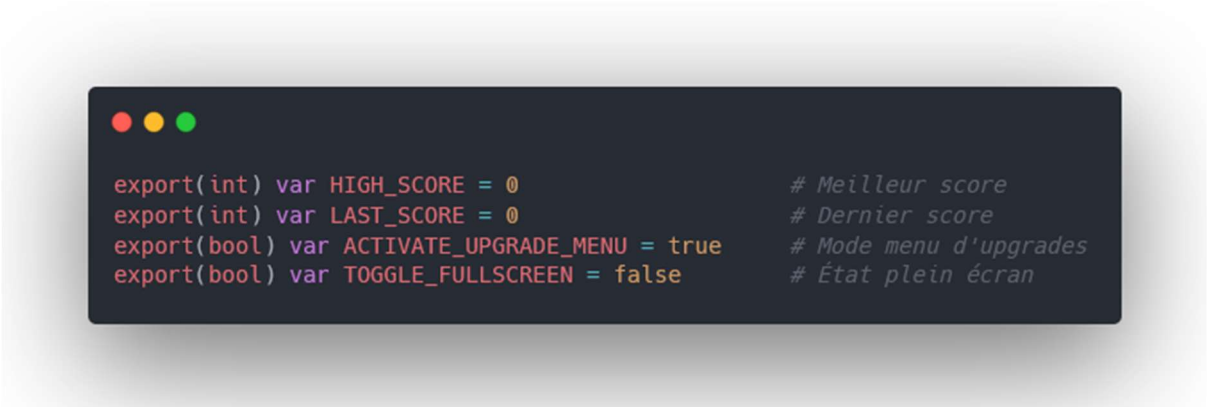
4.1. Structures de Données Clés

Les données principales du jeu sont gérées par des variables au sein des scripts GDScript.

- **Player.gd :**

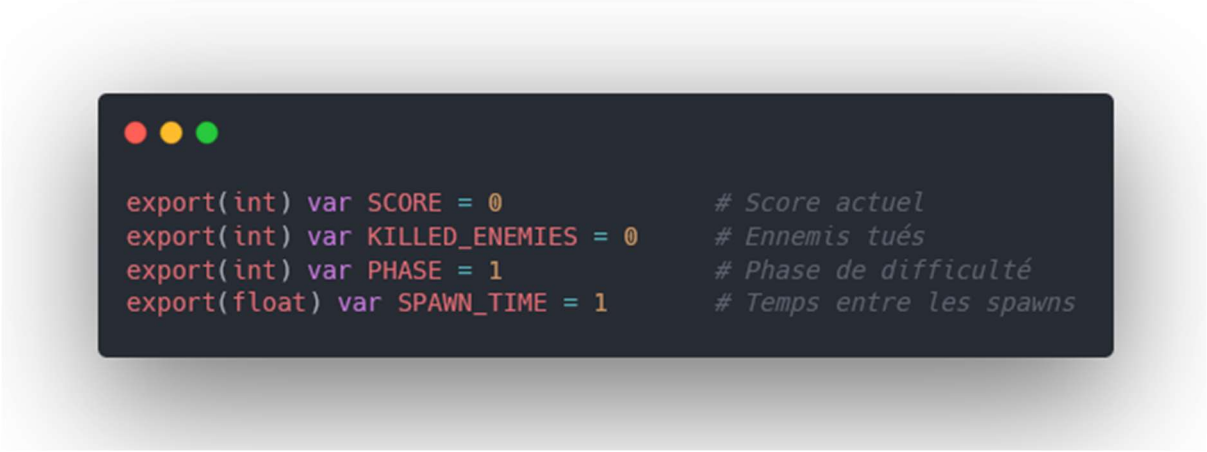
```
export(int) var SPEED = 200           # Vitesse de déplacement
export(int) var HEALTH = 3            # Points de vie actuels
export(int) var MAX_HEALTH = 3        # Points de vie maximum
export(int) var DAMAGE = 1            # Dégâts infligés
export(String) var BOOSTS = "None"    # Boost actif ("None", "Sword", "Dash")
```

- **GameSettings.gd (AutoLoad/Singleton) :**



```
export(int) var HIGH_SCORE = 0                    # Meilleur score
export(int) var LAST_SCORE = 0                    # Dernier score
export(bool) var ACTIVATE_UPGRADE_MENU = true     # Mode menu d'upgrades
export(bool) var TOGGLE_FULLSCREEN = false        # État plein écran
```

- **GameProcess.gd (Contrôleur de jeu) :**



```
export(int) var SCORE = 0                 # Score actuel
export(int) var KILLED_ENEMIES = 0        # Ennemis tués
export(int) var PHASE = 1                 # Phase de difficulté
export(float) var SPAWN_TIME = 1          # Temps entre les spawns
```

4.2. Mécanisme de Stockage

- **Données en Mémoire :** L'état courant du jeu (position du joueur, vie, score actuel, etc.) est stocké dans des variables au sein des scripts ;
- **Persistance des Données :** Les scores (meilleur score, dernier score) et les paramètres utilisateur (activation du menu d'upgrade, plein écran) sont sauvegardés de manière persistante dans un fichier binaire local (user://save.save). Cette sauvegarde utilise les fonctions File.store_var() et File.get_var() de Godot pour sérialiser un dictionnaire contenant les données.

4.3. Flux de Données Principal

1. **Initialisation :** Au lancement du jeu, GameSettings.gd (chargé automatiquement) tente de charger les données depuis user://save.save. Si le fichier n'existe pas, des valeurs par défaut sont utilisées et une première sauvegarde est effectuée ;
2. **Gameplay :** Durant une partie, GameProcess.gd met à jour le score, le nombre d'ennemis tués, la phase de difficulté, et gère l'apparition des ennemis. Les interactions (collisions, tirs) modifient les statistiques du joueur et des ennemis ;
3. **Fin de Partie :** Lorsque le joueur perd, le score final est comparé au meilleur score. Si un nouveau record est établi, HIGH_SCORE est mis à jour. LAST_SCORE est toujours mis à jour. GameSettings.saveData() est ensuite appelé pour écrire ces informations sur disque.

5. Fonctionnalités Principales et Implémentation

5.1. Système de Mouvement et Tir du Joueur

- **Description :** Le joueur contrôle un personnage circulaire qui se déplace librement dans un espace 2D et peut tirer des projectiles en direction du curseur de la souris. Deux types de projectiles (carré et triangle) sont disponibles et peuvent être alternés pour toucher les cibles carrées ou triangulaires ;
- **Fichiers Impliqués :** Scripts/Characters/Player.gd (méthodes _physics_process, switch, shoot), Scripts/Projectiles/onProjectileCreate.gd ;
- **Logique Clé :** Le mouvement est géré par la détection des actions d'input (ZQSD/flèches). La visée se fait avec la souris, et un système de cooldown limite la cadence de tir ;

```gdscript
# Dans Player.gd - Gestion du mouvement
func _physics_process(delta):
    var velocity = Vector2.ZERO
    if Input.is_action_pressed("up"): velocity.y -= 1
    # ... autres directions ...
    if velocity.length() > 0:
        velocity = velocity.normalized()
        $AnimatedSprite.animation = "Walk"
    else:
        $AnimatedSprite.animation = "Idle"
    position += velocity * delta * SPEED
    $Cursor.look_at(get_global_mouse_position()) # Orientation du curseur de tir
```

5.2. Système d'Ennemis et IA

- **Description :** Le jeu propose trois types d'ennemis (Triangle, Carré, Losange) avec des comportements et statistiques distincts. Ils poursuivent le joueur et leur difficulté augmente avec les phases de jeu ;
- **Fichiers Impliqués :** Scripts/Characters/TriangleEnemy.gd, Scripts/Characters/SquareEnemy.gd, Scripts/Characters/LozengeEnemy.gd ;
- **Logique Clé :** Les ennemis utilisent une IA de poursuite simple (calcul de la direction vers le joueur). Ils bénéficient d'une immunité temporaire à leur apparition. Leurs statistiques (vie) augmentent en fonction de la variable Game.PHASE.

```gdscript
# Dans un script d'ennemi (ex: SquareEnemy.gd) - Poursuite et scaling
func _physics_process(delta):
    var direction = (Player.position - position).normalized()
    position += direction * delta * SPEED
    $AnimatedSprite.flip_h = direction.x < 0

func _ready():
    if Game.PHASE > 2:
        HEALTH += Game.PHASE - 2 # Augmentation de la vie avec les phases
    # ... logique d'immunité avec Timer ...
```

5.3. Système d'Améliorations (Upgrades)

- **Description :** En tuant des ennemis, le joueur accumule des points qui peuvent être dépensés pour améliorer l'attaque, les points de vie maximum ou la vitesse de déplacement. Ces améliorations peuvent être choisies via un menu de sélection de cartes ou par des raccourcis clavier ;
- **Fichiers Impliqués :** Scripts/UI/CardSelector.gd, Scripts/GameProcess.gd, Scripts/Characters/Player.gd ;
- **Logique Clé :** Un point d'amélioration est octroyé tous les 10 ennemis tués (ce seuil augmente avec CARD_PHASE). Les statistiques du joueur sont directement modifiées lors de la sélection d'une amélioration ;

```gdscript
# Dans CardSelector.gd - Application d'une amélioration d'attaque
func _on_AttackIncrease_pressed():
    if $AnimationPlayer.is_playing(): return # Évite clics multiples
    Player.DAMAGE += 1
    Game.ATTACK_COUNT += 1
    Game.UPGRADE_COUNT -= 1
    cardSelected() # Logique de fermeture du menu
```

5.4. Système de Collectibles et Power-ups

- **Description :** Les ennemis éliminés ont une chance de laisser tomber des objets collectibles (cœurs pour la vie, épées pour un boost d'attaque, "dash" pour une capacité d'esquive, bombes pour des dégâts de zone) ;

- **Fichiers Impliqués :** Scripts dans Scripts/Collectibles/ (ex: Heart.gd, Sword.gd) ;
- **Logique Clé :** Le "drop" des collectibles est basé sur une probabilité (1 chance sur 23 pour chaque type de power-up standard). Les bombes apparaissent tous les 50 ennemis tués. Les power-ups modifient temporairement ou définitivement les capacités du joueur.

```gdscript
# Dans un script d'ennemi - Logique de drop de collectible
func spawnCollectible() -> void:
    var randomNumber = (randi() % 23 + 1) # Génère un nombre entre 1 et 23
    var generateHeart = (randomNumber == 21)
    # ... autres probabilités ...
    if generateHeart == true:
        var spawnedHeart = Heart.instance()
        get_tree().current_scene.add_child(spawnedHeart)
        spawnedHeart.global_position = global_position
```

5.5. Système de Sauvegarde et Gestion des Scores

- **Description :** Le jeu sauvegarde automatiquement le meilleur score, le dernier score réalisé, et certains paramètres de jeu (comme l'activation du menu d'upgrade ou le mode plein écran) ;
- **Fichiers Impliqués :** Scripts/AutoLoads/GameSettings.gd (méthodes saveData, loadData), Scripts/Characters/Player.gd (appel à la sauvegarde dans game_over) ;
- **Logique Clé :** Utilisation d'un AutoLoad (singleton) pour un accès global aux données. La sérialisation se fait dans un fichier binaire via File.store_var() avec un dictionnaire. Le chargement se fait au démarrage du jeu.

```gdscript
# Dans GameSettings.gd - Sauvegarde des données
func saveData():
    GameData = {
        "highScore": HIGH_SCORE,
        "lastScore": LAST_SCORE,
        "activateUpgradeMenu": ACTIVATE_UPGRADE_MENU,
        "toggleFullScreen": TOGGLE_FULLSCREEN,
    }
    var file = File.new()
    file.open(SAVE_FILE_PATH, file.WRITE)
    file.store_var(GameData)
    file.close()
```

6. Interface Utilisateur (UI)

6.1. Description des Écrans/Vues Principales

- **Menu Principal (MainMenu.tscn) :** Affiche le titre du jeu, les boutons "Play", "Options", "Quit", ainsi que le meilleur score et le dernier score ;
- **Écran de Jeu (Game.tscn) :** Comprend un HUD (Head-Up Display) affichant la barre de vie du joueur, le score actuel, et les compteurs d'améliorations. Des indicateurs visuels peuvent apparaître pour les power-ups actifs ;
- **Menu de Sélection d'Améliorations (CardSelector.tscn) :** Propose trois cartes au joueur pour choisir une amélioration (attaque, vie, vitesse). Le jeu est mis en pause pendant cette sélection ;
- **Menu de Mort (DeathMenu.tscn) :** S'affiche à la fin de la partie, montrant le score final et le meilleur score, avec des options pour "Recommencer", retourner au "Menu Principal" ou "Quitter" ;
- **Menu Pause (PauseMenu.tscn) :** Accessible en jeu, il permet de reprendre la partie, d'aller aux options, ou de quitter.

6.2. Définition de l'UI et Composants Clés

L'interface utilisateur est construite en utilisant les nœuds UI natifs de Godot :

- **Nœuds Control :** Utilisés comme conteneurs et pour organiser les layouts ;
- **Nœuds Button :** Pour toutes les actions interactives, souvent couplés à un AnimationPlayer pour des effets visuels (survol, clic) ;

- **Nœuds Label :** Pour afficher du texte (scores, titres, descriptions) ;
- **Nœud TextureProgress :** Utilisé pour la barre de vie.

Les animations des boutons (survol, clic) sont gérées par des nœuds AnimationPlayer associés à chaque bouton, améliorant le retour visuel. Le HUD est mis à jour dynamiquement dans la fonction _process du script de l'interface de jeu.

6.3. Flux d'Interaction Utilisateur Principaux

1. **Démarrage :** L'utilisateur navigue dans le Menu Principal, clique sur "Play" pour lancer une partie ;
2. **Gameplay :**
    - **Déplacement :** Touches ZQSD ou flèches directionnelles ;
    - **Tir :** Espace ou clic gauche de la souris ;
    - **Changement de projectile :** Touche E ou clic droit de la souris.
3. **Améliorations :** Soit via le menu de cartes qui apparaît automatiquement, soit par les raccourcis clavier (1, 2, 3) ;
4. **Pause :** Touche Échap pour ouvrir le menu de pause ;
5. **Fin de Partie :** Le Menu de Mort s'affiche, proposant de rejouer ou de quitter.

7. Points Techniques Notables et Solutions

7.1. Gestion des Collisions et Types d'Objets

- **Problème :** Distinguer efficacement les différents types d'entités (joueur, ennemis, projectiles) lors des événements de collision ;
- **Solution :** Les collisions sont gérées en vérifiant le nom du nœud ou d'un groupe auquel appartient le corps entrant en collision. Par exemple, if "SquareEnemyHitBox" in body.get_name().

7.2. Système d'Immunité Temporaire

- **Problème :** Empêcher que les ennemis soient détruits instantanément à leur apparition, leur laissant une chance d'agir ;
- **Solution :** Un Timer est initié pour chaque ennemi à sa création (_ready()), lui accordant une courte période d'immunité (0.25s) pendant laquelle il ne peut pas subir de dégâts.

7.3. Scaling de Difficulté

- **Problème :** Assurer une progression de la difficulté pour maintenir l'engagement du joueur ;
- **Solution :** Un système de "phases" (Game.PHASE) est implémenté, basé sur le nombre d'ennemis tués. Les statistiques des ennemis (notamment les points de vie) augmentent avec les phases.

7.4. Gestion de la Pause et des Animations

- **Problème :** Éviter les interactions non désirées avec l'interface utilisateur (clics multiples, actions pendant une transition) lorsque des animations sont en cours ;
- **Solution :** Vérification de l'état de l'AnimationPlayer (if $AnimationPlayer.is_playing(): return) avant de traiter une interaction sur un bouton ou un élément d'UI animé.

8. Stratégies de Gestion des Erreurs et de Journalisation

8.1 Gestion des Erreurs

- **Validation des États :** Des vérifications sont en place pour prévenir des actions non souhaitées, par exemple, empêcher les clics multiples sur les boutons d'upgrade pendant une animation (if $AnimationPlayer.is_playing(): return) ;
- **Gestion des Fichiers de Sauvegarde :** Lors du chargement des données (loadData() dans GameSettings.gd), le code vérifie si le fichier de sauvegarde existe. S'il n'existe pas, un fichier avec des valeurs par défaut est créé et sauvegardé, évitant un crash au premier lancement ;
- **Collisions :** La distinction des types d'objets lors des collisions se fait par la vérification du nom du nœud, ce qui est une forme de validation pour appliquer les effets corrects.

8.2 Journalisation

Le projet s'appuie principalement sur les outils de débogage intégrés à Godot Engine :

- **Instructions print() :** Utilisées durant le développement pour afficher des informations dans la console de sortie de Godot ;
- **Débogueur de Godot :** Permet de surveiller les variables, les performances et les erreurs d'exécution ;
- **Retours Visuels :** Des indicateurs en jeu (score, vie, effets d'impact) servent de "journalisation" implicite de l'état du jeu pour le joueur.

9. Perspectives d’Évolution et Améliorations Futures

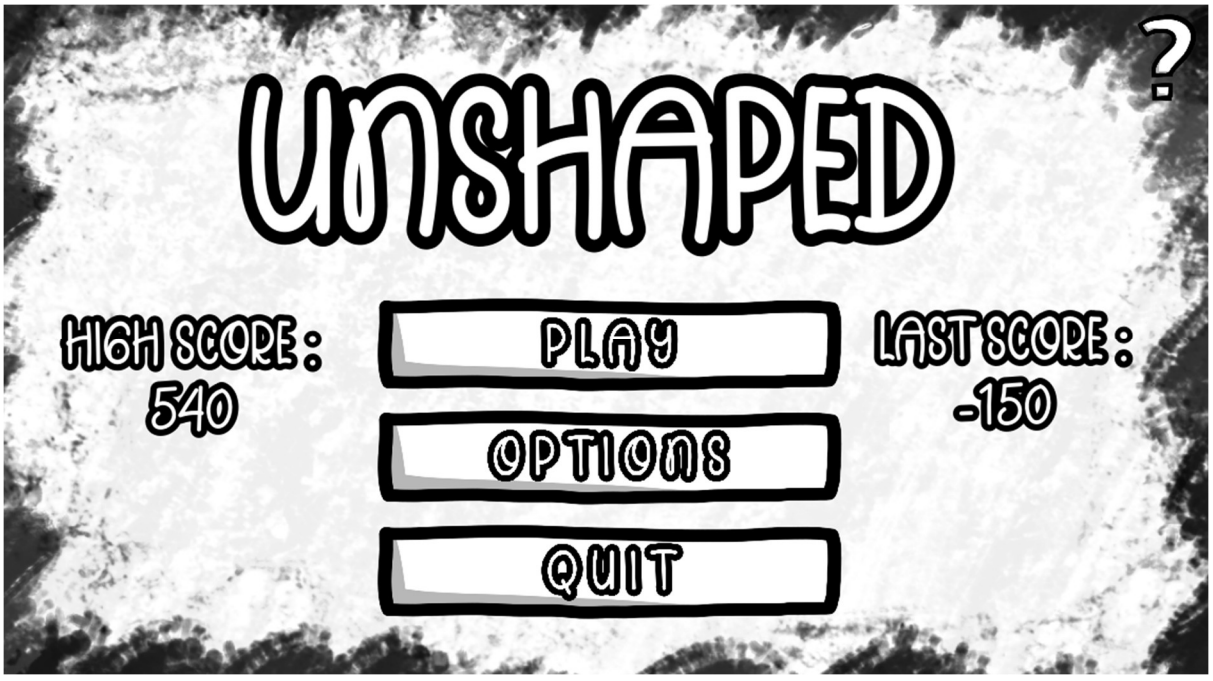
Bien que "Unshaped" soit un jeu terminé et fonctionnel, il comporte beaucoup d’améliorations possibles :

- **Variété d'Ennemis et Comportements :** Introduire de nouveaux types d'ennemis avec des schémas d'attaque plus complexes (projectiles ennemis, ennemis de soutien, pièges) ;
- **Boss Fights :** Ajouter des combats de boss à des intervalles clés pour varier le rythme et offrir des défis majeurs ;
- **Plus de Power-ups et d'Améliorations :** Diversifier les collectibles avec des effets uniques et étendre l'arbre des améliorations possibles pour le joueur ;

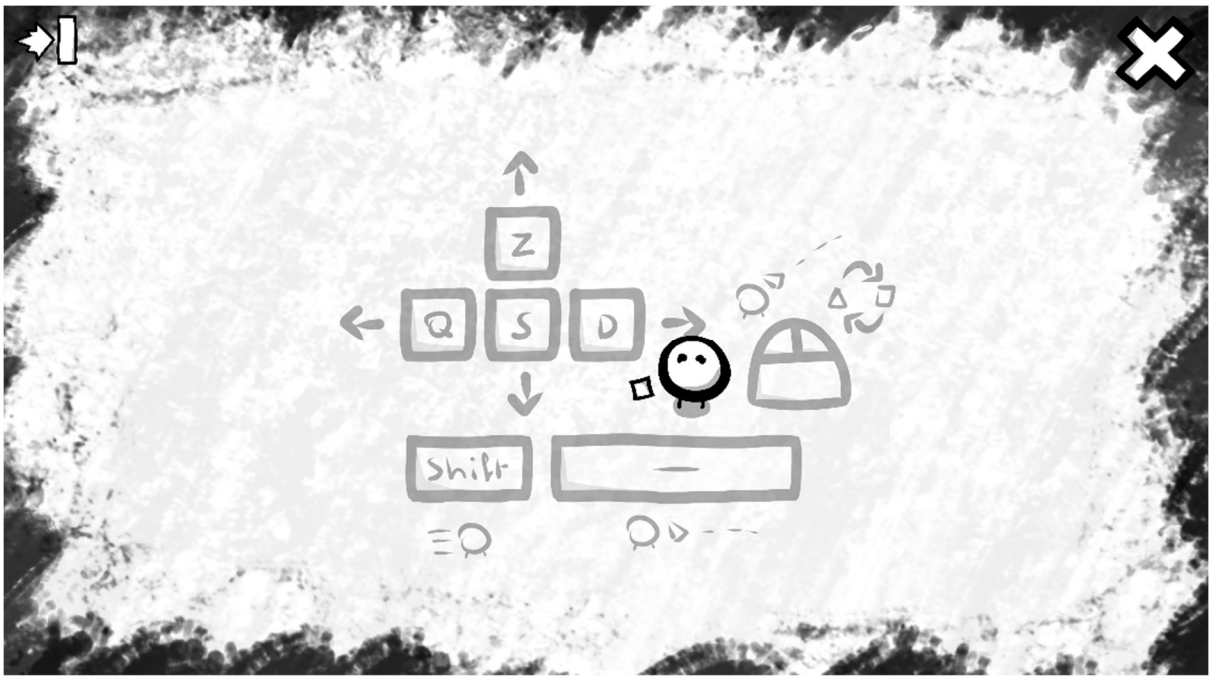
- **Environnements et Niveaux :** Si le jeu devait s'étendre au-delà d'une arène unique, concevoir différents niveaux avec des obstacles ou des caractéristiques environnementales spécifiques ;
- **Améliorations Graphiques et Sonores :** Ajouter plus d'effets visuels (particules, shaders), des animations plus détaillées, et une bande-son plus riche avec des effets sonores variés ;
- **Modes de Jeu Supplémentaires :** Envisager des modes comme un mode "challenge" avec des objectifs spécifiques, ou un mode "coopératif" local ;
- **Équilibrage Affiné :** Continuer à tester et ajuster la courbe de difficulté, les probabilités de drop, et l'impact des améliorations pour une expérience optimale ;
- **Localisation :** Adapter le jeu pour d'autres langues ;
- **Optimisation des Performances :** Pour des vagues d'ennemis très importantes, analyser et optimiser les performances (object pooling pour les projectiles et ennemis, par exemple).

10. Conclusion

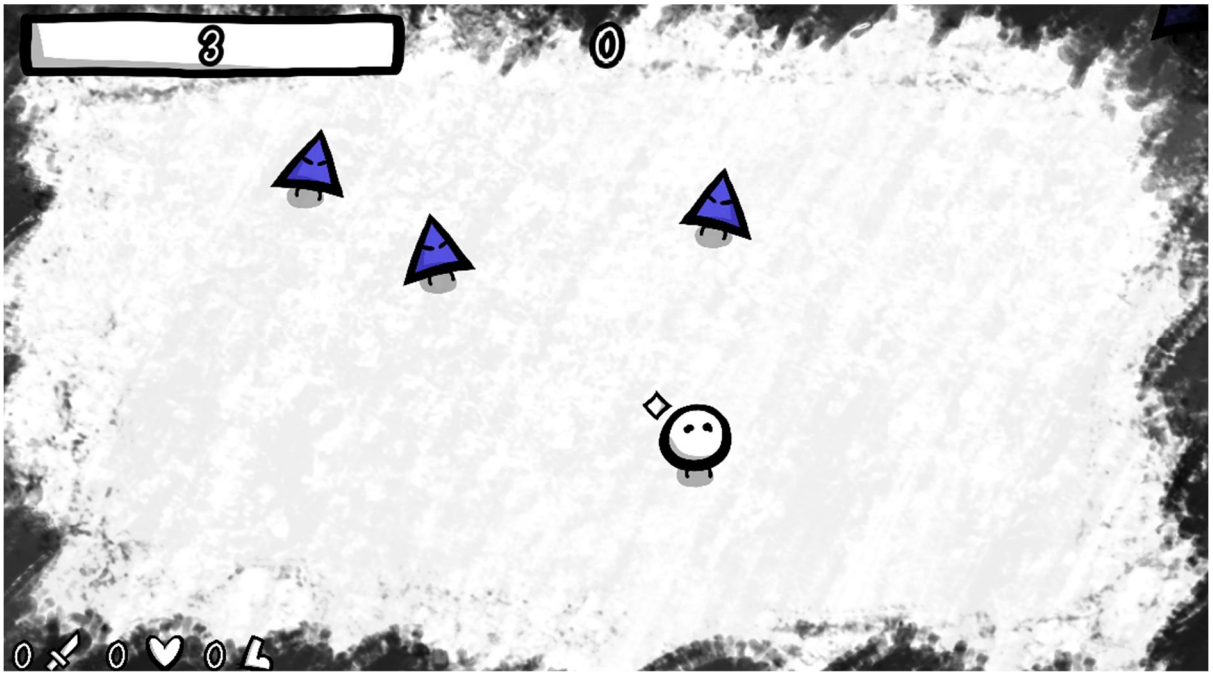
Avec le développement de ce jeu, j'ai pu mettre en application les fonctionnalités du moteur de jeu Godot et apprendre les principes de conception d'un jeu 2D. J'ai structuré le code de façon modulaire, en veillant à bien séparer les responsabilités entre les différentes scènes et scripts. J'ai implémenté les mécaniques de jeu essentielles, telles que le mouvement du personnage, le système de tir, l'intelligence artificielle des ennemis, le mécanisme d'améliorations, la gestion des objets collectibles et la persistance des données, en m'assurant de leur cohérence et de leur fonctionnalité. Ce projet a été pour moi l'occasion d'approfondir ma compréhension de la gestion des scènes, des signaux, des timers, des animations et de la sauvegarde de données au sein de l'écosystème Godot. Je considère "Unshaped" comme une première démonstration de ma capacité à mener à terme un projet.



(Page d’accueil du jeu)



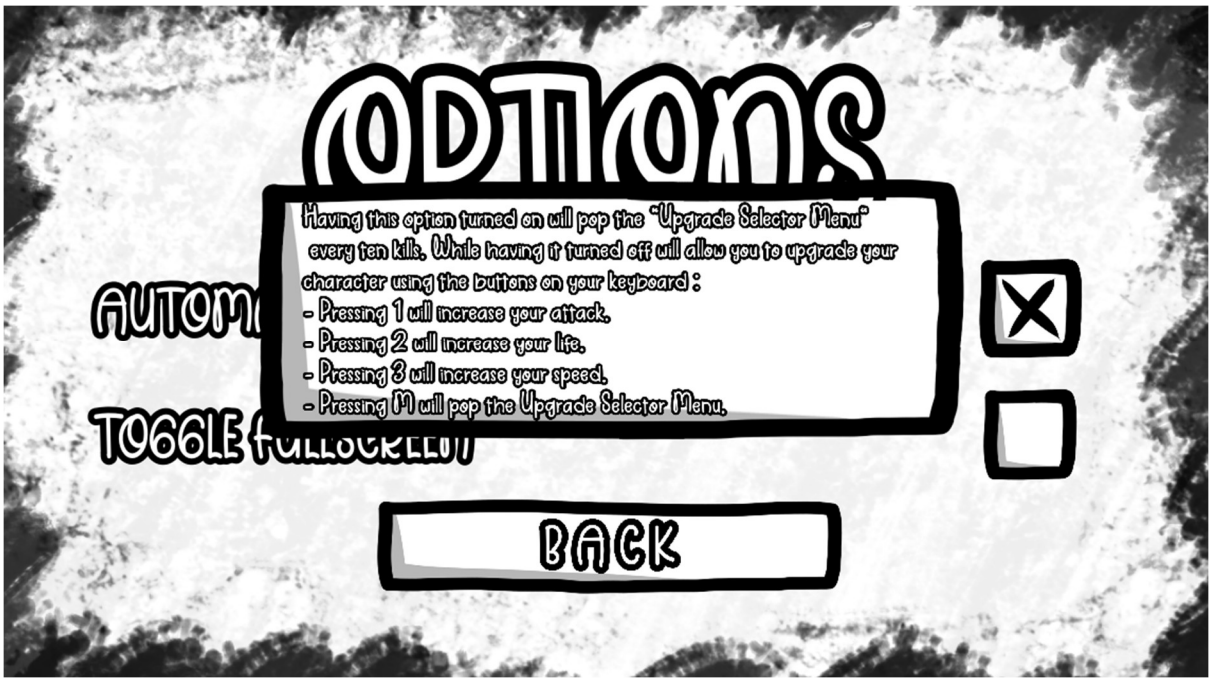
(Zone de tutoriel du jeu)



(Jeu)

(Menu de sélection d'améliorations)

(Menu de mort)



(Menu de configuration du jeu avec des tooltips)