Rapport - Document de Conception de Jeu

SQUIDGAME level 2

Présenté par Étudiant : belasri safia ***Encadré par : Pr lefdaoui youssef***

"Squid Game"

Présenté dans le cadre du cours de développement de jeux
Étudiant :

belasri safia :level 2

madiha nachid : level 1

Sommaire

1. Vision du projet

Concept général

"Squid Game: Escape 2D" est une adaptation vidéoludique en 2D inspirée de la série télévisée "Squid Game". Ce jeu de plateforme et de survie place le joueur dans la position d'un participant contraint de surmonter des épreuves mortelles.

Proposition de valeur

- **Genre :** Plateforme-action avec éléments de survie.
- **Public cible :** Adolescents et adultes familiers avec l'univers de la série.
- **Expérience visée :** Créer une tension constante où chaque décision peut être fatale
- **Objectif ludique :** Maîtriser le timing et la stratégie pour survivre à chaque épreuve

Univers narratif

Le joueur incarne un participant anonyme qui doit traverser des versions stylisées et déformées de jeux d'enfants, transformés en épreuves mortelles. L'ambiance oscillera entre nostalgie enfantine et danger omniprésent, fidèle à l'esprit de la série originale.

2. Mécaniques fondamentales

Système de déplacement

- **PC :** Touches directionnelles (gauche/droite) + barre d'espace pour sauter
- **Mobile :** Interface tactile adaptative avec boutons virtuels

- **Spécificité :** Bouton dédié "Avancer/Stop" pour le jeu"

Système de vie et d'échec

- **Vies :** 3 au départ de chaque niveau
- **Conditions d'échec :**
    - Perte de toutes les vies
    - Déplacement pendant une phase interdite

Interactions environnementales

- **Obstacles :** Structures pour se protéger des tirs
- **Zones de danger :** Aires à éviter sous peine d'élimination
- **Éléments interactifs :** Potentiellement des leviers, plateformes mobiles

3. Structure des niveaux

Niveau 2 : "2 Gardiens "

- **Concept :** Parcours d'obstacles avec des gardiens postés tirant à intervalles réguliers
- **Mécanisme central :** Synchronisation des déplacements entre les tirs des gardiens
- **Obstacles :** Rochers, murs et structures offrant une protection temporaire
- **Progression :** Trois sections avec difficulté croissante (nombre de gardiens, fréquence des tirs)

4. Systèmes d'intelligence artificielle

IA de la Poupée (Niveau 2)

- **Comportement :** Rotation programmée à intervalles prédéfinis
- **États :** Deux positions distinctes correspondant aux phases verte et rouge
- **Détection :** Analyse du mouvement du joueur pendant la phase rouge via raycasting
- **Variables d'ajustement :**
    - rotationInterval : durée entre les changements d'état

    - detectionPrecision : sensibilité à la détection de mouvement

IA des Gardiens (Niveau 2)

- **Comportement :** Rotation et tir à intervalles réguliers
- **Système de visée :** Détection de la présence du joueur sur la ligne de tir
- **Variables d'ajustement :**
    - rotationTime : fréquence de rotation
    - shootingDelay : temps entre chaque tir
    - bulletSpeed : vitesse des projectiles

Possibilités d'évolution

- Ajout d'une détection de trajectoire pour les gardiens
- Comportements adaptatifs basés sur les mouvements habituels du joueur
- Variantes d'adversaires avec des patterns de déplacement (patrouilles)

5. Interface utilisateur et expérience de jeu

HUD (Affichage tête haute)

- **Chronomètre :** Position supérieure centrale, dégradé de couleur selon l'urgence
- **Compteur de vies :** Coin supérieur gauche, représenté par des icônes

Écrans de navigation

- **Menu principal :** le jeu, sélection de niveau.
- **Écran de fin de niveau :** Statistiques, temps écoulé, score obtenu
- **Écran Game Over :**, option de réessaie ou retour au menu

Interfaces contextuelles

- Marqueurs de trajectoire pour les tirs des gardiens

6. Direction artistique et sonore

Style graphique

- Esthétique 2D avec influences pixel art et éléments vectoriels
- Palette de couleurs contrastée reprenant le code couleur de la série (rose, vert, rouge)

- Animation fluide pour le personnage principal et réactive pour les adversaires

Feedback visuel

- Effets de particules lors des éliminations
- Vibrations d'écran pour les événements importants
- Ralentis dramatiques lors des moments décisifs

7. Architecture technique

Moteur et environnement

- **Moteur :** Unity 2D (version 6000.0.40f1 LTS)
- **Langage :** C#
- **Format des assets :** Principalement PNG, animations en spritesheet .
- **Plateformes cibles :** Windows (primaire), Android (secondaire)

Organisation du code

- Architecture orientée composants suivant les principes SOLID
- Séparation claire entre logique de jeu et représentation
- Utilisation du système d'événements Unity pour la communication entre objets

Gestion des ressources

- Système de pooling pour les objets fréquemment instanciés (projectiles)
- Chargement asynchrone des niveaux pour fluidifier l'expérience
- Optimisation pour appareils mobiles (LOD, compression des textures)

8. Plan de développement

État d'avancement actuel

Fonctionnalité	État	Priorité	Commentaires
Contrôleur joueur	✅ Terminé	Haute	Fonctionnel avec animations
Système de chronomètre	⚠️ en test	Basse	Adaptable par niveau
IA Poupée (Niveau 1)	✅ Terminé	Haute	Tests de détection en cours

Fonctionnalité	État	Priorité	Commentaires
IA Gardiens (Niveau 2)	✅ Terminé	Haute	Optimisation du raycasting nécessaire
Système de collisions	⚠️ En test	Haute	Ajustement des hitboxes en cours
Interface principale	🔄 En cours	Moyenne	Maquettes finalisées
Version mobile	📝 Planifié	Basse	Adaptation UI nécessaire

Calendrier prévisionnel :

- **Phase 1 (Complétée) :** Prototype jouable des deux premiers niveaux
- **Phase 2 (Complétée) :** Finalisation des mécaniques, équilibrage et tests utilisateurs

9. Analyse du marché et positionnement

Contexte du marché

Les adaptations de séries populaires en jeux vidéo représentent un segment de marché significatif, particulièrement lorsqu'elles sont proposées dans un format accessible comme les jeux 2D multi-plateformes. L'univers de "Squid Game" n'a pas encore été largement exploité dans le domaine vidéoludique, offrant ainsi une opportunité de marché notable.

Analyse concurrentielle

Titre	Similarités	Différences	Enseignements
Fall Guys	Épreuves par élimination, ambiance colorée	Multijoueur, physique exagérée	Simplicité des contrôles comme facteur de succès
Inside	Ambiance oppressante, puzzles mortels	Narration plus profonde, graphismes sombres	Efficacité d'une direction artistique minimaliste
Little Nightmares	Jeu d'évasion, adversaires géants	Plus axé sur l'horreur, exploration	Importance du rythme entre tension et relâchement

Avantages compétitifs

- Exploitation d'une licence populaire avec une base de fans établie
- Adaptation fidèle de mécaniques de jeu déjà iconiques dans la culture populaire
- Format accessible (2D) permettant une distribution large et un développement optimisé

Public cible

- **Démographique primaire :** Jeunes adultes (18-35 ans) familiers avec la série
- **Démographique secondaire :** Joueurs occasionnels attirés par la reconnaissance de la licence
- **Marché géographique :** Distribution mondiale avec accent particulier sur les marchés asiatiques et occidentaux

10. Dimension pédagogique du projet

Objectifs d'apprentissage

Ce projet a été conçu comme un exercice complet de développement de jeu vidéo, couvrant les aspects suivants:

- **Programmation orientée objet** : Application des principes SOLID dans un contexte de jeu
- **Game design** : Équilibrage de la difficulté et conception de niveaux progressifs
- **Intelligence artificielle** : Création de comportements ennemis diversifiés
- **Game feel** : Travail sur les retours visuels et sonores pour améliorer l'expérience

Compétences développées

- Maîtrise de l'environnement Unity et du C#
- Gestion de projet avec méthodologie agile (sprints, itérations)
- Résolution de problèmes techniques spécifiques aux jeux 2D
- Optimisation des performances pour adaptation multi-plateformes

Applications pédagogiques potentielles

- Utilisation comme projet de référence dans des cours de développement de jeux
- Base extensible pour exercices (ajout de niveaux, modification de mécaniques)
- Étude de cas sur l'adaptation d'une propriété intellectuelle existante

11. Annexes techniques

Descriptions des scripts principaux

PlayerController.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class PlayerController : MonoBehaviour
{
    public float moveSpeed = 5f;
    public Sprite idleSprite;
    public Sprite runningSprite;
    public Sprite deadSprite;

    private Vector3 lastPosition;
    private bool isDead = false;
    private bool isBlocked = false; // Empêche le mouvement si bloqué
    private SpriteRenderer spriteRenderer;
    private GuardianController guardian;

    void Start()
    {
        spriteRenderer = GetComponent<SpriteRenderer>();
        guardian = FindFirstObjectByType<GuardianController>();
        lastPosition = transform.position;
    }

    void Update()
    {
```

```
    if (isDead || isBlocked) return; // Si mort ou bloqué, ne pas bouger

    HandleMovement();
  }

  void HandleMovement()
  {
    float moveX = Input.GetAxis("Horizontal");
    float moveY = Input.GetAxis("Vertical");
    Vector3 newPosition = transform.position + new Vector3(moveX * moveSpeed *
Time.deltaTime, moveY * moveSpeed * Time.deltaTime, 0);

    transform.position = newPosition;

    if (moveX != 0 || moveY != 0)
    {
      spriteRenderer.sprite = runningSprite;
    }
    else
    {
      spriteRenderer.sprite = idleSprite;
    }

    if (guardian.IsLooking() && (moveX != 0 || moveY != 0))
    {
      Die();
    }
  }
```

```
public void BlockMovement()
{
   isBlocked = true;
   Debug.Log("🚧 Mouvement bloqué par un Castle !");
}

// Ajoute cette méthode pour débloquer le joueur :
public void UnblockMovement()
{
   isBlocked = false;
   moveSpeed = 5f; // Remet la vitesse normale
}
   void Die()
   {
      isDead = true;
      spriteRenderer.sprite = deadSprite;
      Debug.Log("💀 LOSER ! Tu as bougé au mauvais moment !");
   }

   void OnCollisionEnter2D(Collision2D collision)
   {
      if (collision.gameObject.CompareTag("Castle"))
      {
         isBlocked = true; // Bloque le joueur
         transform.position = lastPosition; // Remet à la dernière position valide
         Debug.Log("🚧 Le Castle bloque le passage !");
      }
   }void OnCollisionExit2D(Collision2D collision)
```

```
  {
    if (collision.gameObject.CompareTag("Castle"))  {
      isBlocked = false; // Débloque le joueur
      lastPosition = transform.position; // Sauvegarde la nouvelle position
      Debug.Log("✅ Déblocage du joueur !");
    }
  }
  void LoadNextLevel() {
    int nextSceneIndex = SceneManager.GetActiveScene().buildIndex + 1; // Prend le niveau suivant
    if (nextSceneIndex < SceneManager.sceneCountInBuildSettings) {
      SceneManager.LoadScene(nextSceneIndex); // Charge le prochain niveau
    }
    else{
      Debug.Log("🏆 Félicitations ! Tu as terminé le jeu !");
    }
  }
}
```

PlayerAI.cs

```
using UnityEngine;
using UnityEngine.UI; // Pour changer l'image en cas de Game Over
public class PlayerAI : MonoBehaviour
{
  public float moveSpeed = 2f; // Vitesse du joueur
  public Sprite gameOverSprite; // Image à afficher en cas de Game Over
  private GuardianController guardian;
  private bool canMove = true;
  private int direction = 1; // Direction initiale (1 = droite, -1 = gauche, 0 = immobile)
```

```
  private float reactionTime; // Temps avant que le joueur s'arrête (certains sont plus lents)


  private SpriteRenderer spriteRenderer;
  void Start(){
    guardian = FindAnyObjectByType<GuardianController>(); // Trouve le Guardian dans la scène
    spriteRenderer = GetComponent<SpriteRenderer>(); // Récupère le SpriteRenderer
    reactionTime = Random.Range(0.1f, 1f); // Certains réagissent vite (0.1s), d'autres plus lentement (1s)
    InvokeRepeating("ChangeDirection", 2f, Random.Range(1f, 3f)); // Change la direction toutes les 1-3 sec
  }
 void Update() {
    if (!canMove) return; // Ne pas bouger si mort
    if (guardian.IsLooking() {
      Invoke("StopMoving", reactionTime); // Arrêt progressif après reactionTime
    }
    else{
      transform.position += new Vector3(moveSpeed * direction * Time.deltaTime, 0, 0);
    }
  }
void ChangeDirection()  {
    direction = Random.Range(-1, 2); // -1 = gauche, 1 = droite, 0 = stop
  }
 void StopMoving(){
    if (guardian.IsLooking() && direction != 0) // Si bouge encore => Game Over
    {
      GameOver();
```

```
        }
        else {
            direction = 0; // Arrête le mouvement
        }
    }
void GameOver() {
        canMove = false;
        direction = 0; // Arrête le joueur
        spriteRenderer.sprite = gameOverSprite; // Change l'image en "Game Over"
        Debug.Log(gameObject.name + " 💀 Game Over ! Il a bougé au mauvais moment
!");
    }
}
```

GuardianController.cs

```
using UnityEngine;
using System.Collections;
public class GuardianController : MonoBehaviour
{
    public float rotationTime = 3.0f;  // Temps total pour tourner
    private bool looking = false;
    private bool isRotating = false;
    void Start()    {
        InvokeRepeating("StartRotation", rotationTime, rotationTime);
    }
    void StartRotation(){
        if (!isRotating){
            StartCoroutine(RotateGuardian());
        }
```

```
  }
  IEnumerator RotateGuardian(){
    isRotating = true;
    float duration = 0.5f; // Temps de rotation
    float elapsed = 0f;
    Quaternion startRotation = transform.rotation;
    Quaternion targetRotation = looking ? Quaternion.Euler(0, 0, 0) : Quaternion.Euler(0, 180, 0);
while (elapsed < duration){
      transform.rotation = Quaternion.Slerp(startRotation, targetRotation, elapsed / duration);
      elapsed += Time.deltaTime;
      yield return null;
    }transform.rotation = targetRotation;
    looking = !looking;
    isRotating = false;
  }public bool IsLooking()
  {
    return !isRotating && looking; // Ne regarde que si la rotation est finie
  }
}
```

SceneTransition.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections;
[RequireComponent(typeof(Collider2D))]
public class SceneTransition : MonoBehaviour
{
  [Header("Scene Settings")]
```

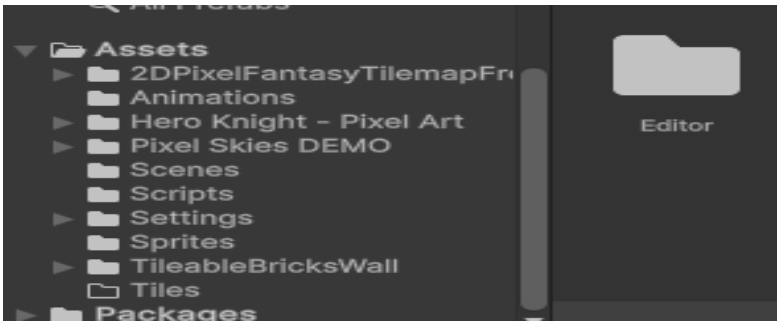
```
  [SerializeField] private string sceneToLoad = "level2"; // Scène cible
  [SerializeField] private float transitionDelay = 1.5f;

  [Header("Audio Settings")]
  [SerializeField] private AudioClip transitionSound;
  private AudioSource audioSource;
  [Header("Visual Effects")]
  [SerializeField] private Animator fadeAnimator; // Animator avec les triggers "Start" et "End"

  [Header("Optional")]
  [SerializeField] private GameObject playerToDisable;
 private void Start()
  {
    audioSource = GetComponent<AudioSource>();
    if (audioSource == null && transitionSound != null)
    {
      Debug.LogWarning("Aucun AudioSource trouvé, mais un son de transition est défini.");
    } if (fadeAnimator == null)
    {
      Debug.LogWarning("Aucun Animator de transition défini.");
    }
  }   private void OnTriggerEnter2D(Collider2D other) {
    if (other.CompareTag("Player")  {
      StartCoroutine(LoadSceneCoroutine());
    }
  }
private IEnumerator LoadSceneCoroutine() {
```

```
        if (playerToDisable != null)
            playerToDisable.SetActive(false); // Désactive le joueur si précisé
if (fadeAnimator != null)
            fadeAnimator.SetTrigger("Start"); // Début de l'animation
 if (audioSource != null && transitionSound != null)
            audioSource.PlayOneShot(transitionSound);
yield return new WaitForSeconds(transitionDelay); // Attente
    SceneManager.LoadScene(sceneToLoad); // Changement de scène
    }
}
```

Structure des dossiers du projet



12. Étude comparative et influences

Influences ludiques

Notre projet s'inscrit dans une tradition de jeux de plateformes à haute tension, et s'inspire de plusieurs œuvres marquantes

- **Limbo/Inside** : Pour l'ambiance oppressante et la mort comme mécanique d'apprentissage
- **Super Meat Boy** : Pour la précision des contrôles et le cycle rapide d'échec-réessai
- **Celeste** : Pour la progression de la difficulté et l'accessibilité des mécaniques

Influences narratives

L'atmosphère du jeu s'inspire directement de la série "Squid Game", mais incorpore également des éléments de:

- **Battle Royale** : Le concept de participants forcés à s'éliminer
- **Alice in Borderland** : L'idée de jeux mortels aux règles strictes
- **The Hunger Games** : La surveillance constante et la théâtralisation de la mort

Éléments d'innovation

Notre jeu se distingue par:

- La transformation fidèle des jeux d'enfants coréens en défis mortels interactifs
- L'alternance entre phases d'action et phases d'immobilité comme mécanisme central

13. Perspectives d'évolution et monétisation

Développement futur

- **Mode histoire étendu** : Ajout de séquences narratives entre les niveaux
- **Mode multijoueur** : Compétition en temps réel entre joueurs
- **Éditeur de niveaux** : Permettre aux joueurs de créer leurs propres défis

Conclusion :

Ce document de conception représente l'état actuel du projet et pourra être révisé selon les retours et l'évolution du développement. Il a été élaboré dans le cadre académique comme démonstration des compétences en game design et documentation technique.