Projet : Création d'un jeu vidéo 2D

Version du document : final

Propriétaire du document : Equipe 22

Date de Création : 25/11/2023

Product Backlog

Id	Fonctionnalités	Priorité
0	**En tant que** développeur, **je souhaite** définir l'architecture de mon projet et l'initialiser **pour que** je puisse démarrer le développement de manière structurée et efficace.	**Haute**
1	**En tant que** développeur de jeu, **je veux** pouvoir créer, supprimer, modifier des objets et que chaque objet ait des dimensions **pour** définir les éléments principaux de mon jeu.	**Haute**
2	**En tant que** développeur de jeu, **je veux** que mes objets puissent se déplacer ou pas **pour** implémenter le mouvement dans le jeu	**Haute**
3	**En tant que** développeur de jeu**, je veux** pouvoir définir et déclencher des forces externes **qui peuvent** influencer le mouvement / position du serpent et des fruits.	**Moyenne**
4	**En tant que** développeur de jeu, **je veux** que mes objets aient une masse **pour** que leur mouvement soit affecté par les forces appliquées.	**Moyenne**
5	**En tant que** développeur de jeu, **je veux** un système de gestion de collision, **pour que** les objets du jeu interagissent d'une manière cohérente et réaliste.	**Haute**
6	**En tant que** développeur de jeu, **je veux** afficher des sprites et des formes sur l'écran **pour** avoir une représentation graphique des objets.	**Haute**

7	**En tant que** développeur de jeu, **je veux** avoir un système de couches dans l'affichage, **pour** distinguer les éléments du background et décider de l'objet à afficher en premier.	**Haute**
8	**En tant que** développeur de jeu, **je veux** pouvoir animer les objets **pour** rendre le jeu plus vivant et voir la conséquence des actions sur les objets.	**Moyenne**
9	**En tant que** développeur de jeu, **je veux** positionner la caméra **pour** décider quelle partie de l'environnement on veut afficher sur l'écran.	**Moyenne**
10	**En tant que** développeur de jeu, **je veux** qu'il y ait une communication entre les moteurs pour lier les différents moteurs indépendants.	**Moyenne**
11	**En tant que** développeur de jeu, **je veux** que le core-Kernel puisse synchroniser les événements de la couche Gameplay avec les événements des moteurs pour invoquer les actions correspondantes dans chaque moteur.	**Moyenne**
12	**En tant que** développeur, **je veux** implémenter la gestion des entrées clavier pour détecter les touches pressées, **pour** que les joueurs puissent interagir avec le jeu à l'aide du clavier .	**Moyenne**
13	**En tant que** développeur, **je veux** intégrer la gestion des événements de la souris, y compris la détection des clics, des survols et des mouvements, **pour** permettre aux joueurs d'interagir avec le jeu à l'aide de la souris.	**Moyenne**
14	**En tant que** développeur de jeu, **je veux** un moteur d'inférence logique **pour** l'interaction des PNJ avec les différents événements du jeu.	**Basse**
15	**En tant que** développeur de jeu, **je veux** avoir un système de "pathfinding" **pour** que les PNJ puissent se déplacer dans l'environnement.	**Basse**
16	**En tant que** développeur de jeu, **je veux** ajouter du son **pour** la musique d'ambiance et ajouter des effets sonores aux actions qui se passent dans le jeu (collision, interaction, bruit de pas…).	**Basse**
17	**En tant que** développeur de jeu, **je veux** préciser de quelle direction vient le son, **pour** donner au joueur une conscience spatiale dans l'environnement.	**Basse**
18	**En tant qu'**utilisateur de jeu, **je veux** avoir un serpent que je peux manipuler **pour** changer sa direction.	**Haute**
19	**En tant qu'**utilisateur, **je veux** pouvoir manger des fruit qui apparaissent aléatoirement avec le serpent **pour** agrandir sa taille	**Haute**

20	**En tant qu'**utilisateur **je veux** que les sprites du serpent changent selon sa direction	**Haute**
21	**En tant qu'**utilisateur **je veux** qu'il y ait des murs à éviter par le serpent.	**Haute**
22	**En tant qu'**utilisateur **je veux** avoir un mode de jeu **afin** que le serpent bouge tout seul.	**Basse**
23	**En tant qu'**utilisateur, **je veux** que le jeu se termine lorsque le serpent entre en collision avec lui-même ou avec les bords de l'écran.	**Basse**
24	**En tant qu'** utilisateur, **je veux** que le serpent puisse traverser d'un bord de l'écran à l'autre lorsque sa tête atteint les bords.	**Basse**

- Rouge : US terminé
- Vert : US en cours
- Bleu : US annulé

Projet : Creation d'un jeu vidéo 2D

Version du document : V1

Propriétaire du document : Equipe 22

Date de Création : 10/10/2023

Sprint Backlog : Version finalisée

1. Liste des fonctionnalités (items)

- 0 : **En tant que** développeur, **je souhaite** définir l'architecture de mon projet et l'initialiser **pour** que je puisse démarrer le développement de manière structurée et efficace.
- 1 : **En tant que** développeur de jeu, **je veux** pouvoir créer, supprimer et modifier des objets et que chaque objet ait des dimensions **pour** définir les éléments principaux de mon jeu.

2. Liste des tâches

- 001 : Conception d'un diagramme de classe général de l'application .
- 002 : Conception de diagramme de package
- 003 : Initialisation de l'environnement du projet
- 101 : Établir la structure / méthode des objets physique (création / suppression / modification)

3. Planification et affectation des tâches

Id	Nom de tâche	Date de début	Date de fin	Etat de la tâche
001	Conception d'un diagramme de class général de l'application	10/10/2023	10/10/2023	Terminée
002	Conception de diagramme de package	11/10/2023	11/10/2023	En cours
003	Initialisation de l'environnement du projet	12/10/2023	12/10/2023	Terminée

101	Établir la structure / méthode des objets physique (création / suppression / modification)	13/10/2023	15/10/2023	En cours

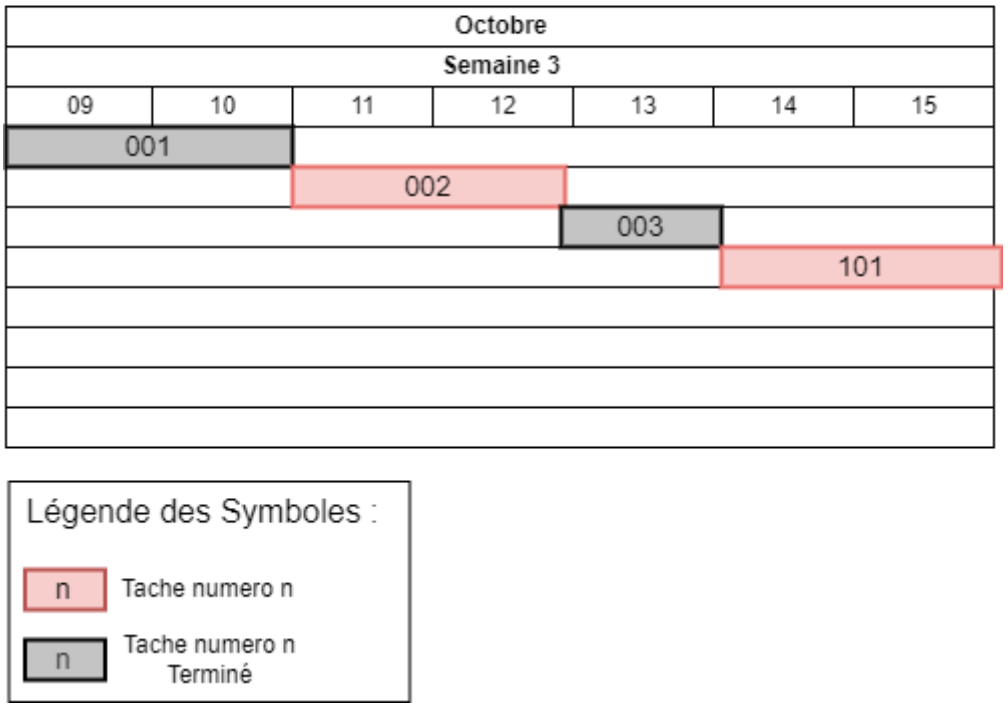
4. Diagramme de Gantt prévisionnel

Diagramme De Gantt

Projet : GL-TP2

Version : 1 - V1

Date : 10/10/2023



Projet : Création d'un jeu vidéo 2D

Version du document : V2

Propriétaire du document : Equipe 22

Date de Création : 22/10/2023

Sprint Backlog : Version finalisée

1. Liste des fonctionnalités (items)

- 0 : **En tant que** développeur, **je souhaite** définir l'architecture de mon projet et l'initialiser **pour que** je puisse démarrer le développement de manière structurée et efficace.
- 1 : **En tant que** développeur de jeu, **je veux** pouvoir créer, supprimer et modifier des objets et que chaque objet ait des dimensions **pour** définir les éléments principaux de mon jeu.
- 2 : **En tant que** développeur de jeu, **je veux** que mes objets puissent se déplacer ou pas **pour** implémenter le mouvement dans le jeu
- 6 : **En tant que** développeur de jeu, **je veux** afficher des sprites et des formes sur l'écran **pour** avoir une représentation graphique des objets.

2. Liste des tâches

- 002 : Conception de diagramme de package
- 101 : Établir la structure / méthode des objets physique (création / suppression / modification)
- 601 : Implémenter l'environnement ou afficher les objets (la scène)
- 201 : Implémenter le mouvement des objets
- 602 : Associer des textures (sprites) aux objets et les afficher dans la scène

3. Planification et affectation des tâches

Id	Nom de tâche	Date de début	Date de fin	Etat de la tâche
101	Établir la structure / méthode des objets physique (création / suppression / modification)	16/10/2023	18/10/2023	Terminée
601	Implémenter l'environnement ou afficher les objet (la scène)	19/10/2023	21/10/2023	Terminée
201	Implémenter le mouvement des objets	19/10/2023	22/10/2023	Terminée
002	Conception de diagramme de package	21/10/2023	-/10/2023	Annulée
602	Associer des textures (sprites) aux objets et les afficher dans la scène	21/10/2023	-/10/2023	En cours

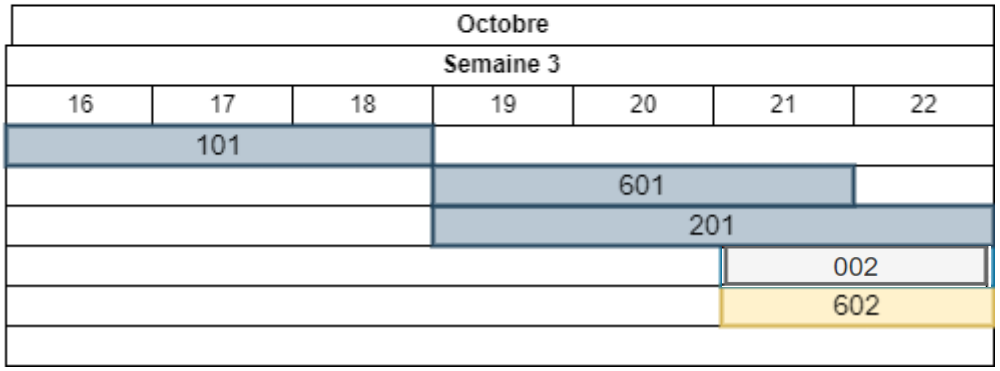
4. Diagramme de Gantt réel

Diagramme De Gantt

Projet : GL-TP2

Version : 2

Date : 22/10/2023



Projet : Création d'un jeu vidéo 2D

Version du document : 3

Propriétaire du document : Equipe 22

Date de Création : 28/10/2023

Sprint Backlog : Version finalisée

1. Liste des fonctionnalités (items)

- 5 : **En tant que** développeur de jeu, **je veux** un système de gestion de collision, **pour que** les objets du jeu interagissent d'une manière cohérente et réaliste
- 6 : **En tant que** développeur de jeu, **je veux** afficher des spirites et des formes sur l'écran **pour** avoir une représentation graphique des objets.
- 12 : **En tant que** développeur, **je veux** implémenter la gestion des entrées clavier pour détecter les touches pressées, **pour** que les joueurs puissent interagir avec le jeu à l'aide du clavier .
- 18 : **En tant qu'**utilisateur de jeu, **je veux** avoir un serpent que je peux manipuler **pour** changer sa direction.

2. Liste des tâches

- 502 : Définir les règles de collision pour chaque type d'objet.
- 503 : Gérer les événements déclenchés par les collisions.
- 504 : Implémenter des mécanismes de résolution de collision pour éviter que les objets ne se superposent de manière incohérente.
- 601 : Implémenter l'environnement ou afficher les objets (la scène)
- 602 : Associer des textures (spirites) aux objets et les afficher dans la scène
- 603 : Gérer la rotation des spirites pour une représentation visuelle.
- 1201 : Détecter les touches pressées
- 1202 : Associer des actions aux touches pressées
- 1801 : Manipulation du serpent dans les 4 directions

3. Planification et affectation des tâches

Id	Nom de tâche	Date de début	Date de fin	Etat de la tâche
602	Associer des textures (spirites) aux objets et les afficher dans la scène	21/10/2023	28/10/2023	Terminée
603	Gérer la rotation des spirites pour une représentation visuelle.	21/10/2023	28/10/2023	Terminée
1201	Détecter les touches pressées	28/10/2023	05/11/2023	Terminée
1202	Associer des actions aux touches pressées	28/10/2023	05/11/2023	Terminée
1801	Manipulation du serpent dans les directions	28/10/2023	05/11/2023	Terminée
502	Mettre en place un système de collision pour les objets du jeu.	25/10/2023	26/10/2023	Retardé
503	Gérer les événements déclenchés par les collisions.	26/10/2023	28/10/2023	Retardé
504	Implémenter des mécanismes de résolution de collision pour éviter	28/10/2023	30/10/2023	Annulé

Id	Nom de tâche	Date de début	Date de fin	Etat de la tâche
	que les objets ne se superposent de manière incohérente.			

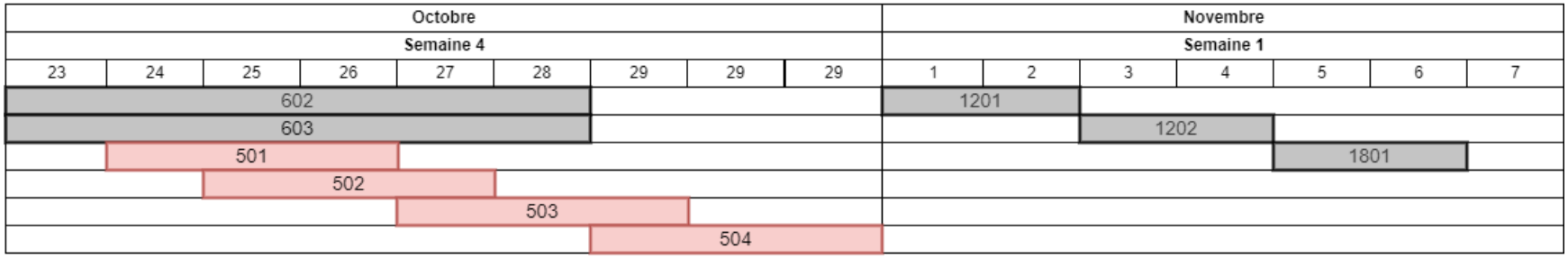
4. Diagramme de Gantt réel

Diagramme De Gantt

Projet : GL-TP2

Version : 3 - V2

Date : 05/11/2023



Légende des Symboles :

n — Tache numero n

n — Tache numero n Terminé

Projet : Création d'un jeu vidéo 2D

Version du document : V4

Propriétaire du document : Equipe 22

Date de Création : 11/11/2023

Sprint Backlog : Version finalisée

1. Liste des fonctionnalités (items)

- 5: **En tant que** développeur de jeu, **je veux** un système de gestion de collision, **pour que** les objets du jeu interagissent d'une manière cohérente et réaliste.
- 19 : **En tant qu'**utilisateur, je veux pouvoir manger des fruits qui apparaissent aléatoirement avec le serpent **pour** agrandir sa taille.
- 20 : **En tant qu'**utilisateur **je veux** que les sprites du serpent changent selon sa direction.
- 21 : **En tant qu'**utilisateur **je veux** qu'il y ait des murs à éviter par le serpent.

2. Liste des tâches

- 502 : Définir les règles de collision pour chaque type d'objet.
- 503 : Gérer les événements déclenchés par les collisions.
- 504 : Implémenter des mécanismes de résolution de collision pour éviter que les objets ne se superposent de manière incohérente.
- 1901 **:** faire apparaître un fruit de façon aléatoire dans une position qui ne soit pas la même que celle du serpent.
- 1902 : ajouter un segment au serpent une fois le fruit a été mangé pour agrandir sa taille et faire apparaître un autre fruit dans une autre position.
- 2001 : définir et découper les différents sprites du serpent.
- 2002 : changer les sprites selon les mouvements du serpent.
- 2101 : Placer les murs.
- 2102 : déclencher la fin du jeu une fois le mur touché par le serpent en détectant la collision.

3. Planification et affectation des tâches

Id	Nom de tâche	Date de début	Date de fin	Etat de la tâche
502	Mettre en place un système de collision pour les objets du jeu.	06/11/2023	7/11/2023	Terminée
503	Gérer les événements déclenchés par les collisions.	07/11/2023	8/11/2023	En cours
1901	Faire apparaître un fruit de façon aléatoire dans une position qui ne soit pas la même que celle du serpent.	08/11/2023	9/11/2023	Terminée
1902	Ajouter un segment au serpent une fois le fruit a été mangé pour agrandir sa taille et faire apparaître un autre fruit dans une autre position.	08/11/2023	11/11/2023	Terminée
2001	Définir et découper les différents sprites du serpent.	06/11/2023	10-/11/2023	Terminée
2002	changer les sprites selon les mouvements du serpent.	06/11/2023	11-/11/2023	Terminée
2101	Placer les murs.	06/11/2023	12/11/2023	Terminée

2102	Déclencher la fin du jeu une fois le mur touché par le serpent en détectant la collision.	06/11/2023	-/11/2023	En cours

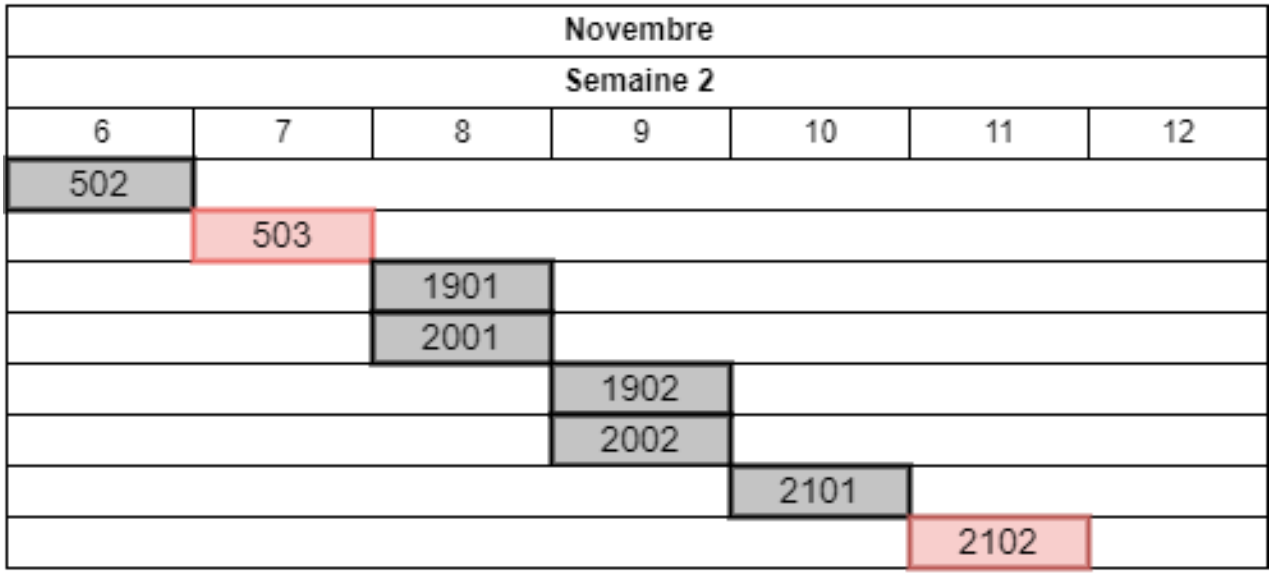
4. Diagramme de Gantt réel

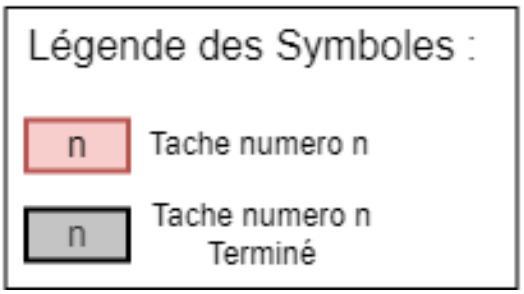
Diagramme De Gantt

Projet : GL-TP2

Version : 4 - V2

Date : 10/11/2023





Projet : Création d'un jeu vidéo 2D

Version du document : V5

Propriétaire du document : Equipe 22

Date de Création : 12/11/2023

Sprint Backlog : Version finalisée

1. Liste des fonctionnalités (items)

- 5 : **En tant que** développeur de jeu, **je veux** un système de gestion de collision, **pour que** les objets du jeu interagissent d'une manière cohérente et réaliste
- 7 : **En tant que** développeur de jeu, **je veux** avoir un système de couches dans l'affichage, **pour** distinguer les éléments du background et décider de l'objet à afficher en premier.
- 23 : **En tant qu'**utilisateur, **je veux** que le jeu se termine lorsque le serpent entre en collision avec lui-même ou avec les bords de l'écran.
- 24 : **En tant qu'** utilisateur, **je veux** que le serpent puisse traverser d'un bord de l'écran à l'autre lorsque sa tête atteint les bords.

2. Liste des tâches

- 503 : Gérer les événements déclenchés par les collisions.
- 701 : Développer un système de gestion des couches d'affichage pour distinguer les éléments du background et décider de l'ordre d'affichage des objets.
- 2301 : mettre en place la logique de collision entre le serpent et les murs et lui même.
- 2302 : déclencher automatiquement la fin du jeu lorsque l'événement de collision est détecté.
- 2401 : Implémenter un mécanisme pour gérer le déplacement du serpent d'un bord de l'écran à l'autre lorsque sa tête atteint les bords.

3. Planification et affectation des tâches

Id	Nom de tâche	Date de début	Date de fin	Etat de la tâche
503	Gérer les événements déclenchés par les collisions.	07/11/2023	16/11/2023	Terminée
701	Développer un système de gestion des couches d'affichage pour distinguer les éléments du background et décider de l'ordre d'affichage des objets.	13/11/2023	14/11/2023	Annulé
2301	Mettre en place la logique de collision entre le serpent et les murs et lui même.	15/11/2023	16/11/2023	Terminée
2302	Déclencher automatiquement la fin du jeu lorsque l'événement de collision est détecté.	16/11/2023	17/11/2023	En cours
2401	Implémenter un mécanisme pour gérer le déplacement du serpent d'un bord de l'écran à l'autre lorsque sa tête atteint les bords.	17/11/2023	018/11/2023	En cours

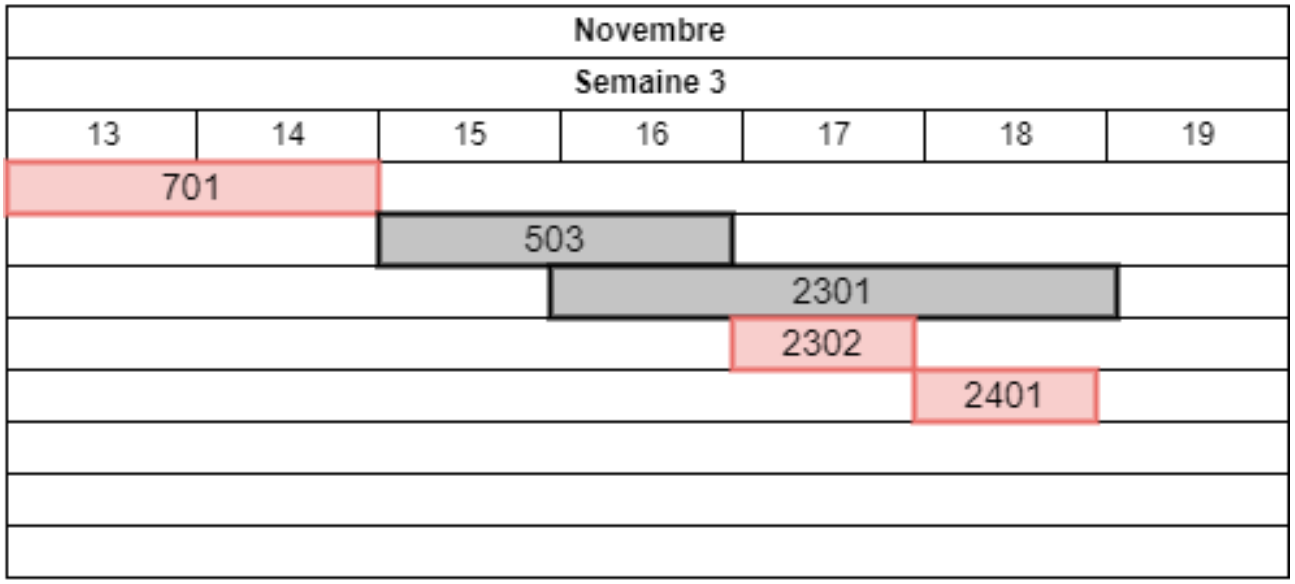
4. Diagramme de Gantt prévisionnel :

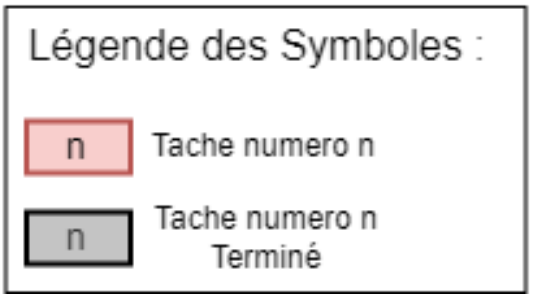
Diagramme De Gantt

Projet : GL-TP2

Version : 5

Date : 19/11/2023





Projet : Création d’un jeu vidéo 2D

Version du document : V6

Propriétaire du document : Equipe 22

Date de Création : 25/11/2023

Sprint Backlog : Version finalisée

1. Liste des fonctionnalités (items)

- 23 : **En tant qu’**utilisateur, **je veux** que le jeu se termine lorsque le serpent entre en collision avec lui-même ou avec les bords de l'écran.
- 24 : **En tant qu'** utilisateur, **je veux** que le serpent puisse traverser d'un bord de l'écran à l'autre lorsque sa tête atteint les bords.
- 16 : **En tant que** développeur de jeu, **je veux** ajouter du son **pour** la musique d’ambiance et ajouter des effets sonores aux actions qui se passent dans le jeu (collision, interaction, bruit de pas…).

2. Liste des tâches

- 2301 : mettre en place la logique de collision entre le serpent et les murs et lui même.
- 2302 : Déclencher automatiquement la fin du jeu lorsqu'une collision est détectée.
- 2303 : Établir un mécanisme de collision entre le serpent et les fruits.
- 2304 : Gérer l'apparition de nouveaux fruits lorsque le serpent consomme un fruit.
- 2401 : Mettre en œuvre un mécanisme permettant au serpent de se déplacer d'un bord de l'écran à l'autre lorsque sa tête atteint les limites.
- 1601 : Intégrer un fond sonore pour l'ambiance du jeu.
- 1602 : Ajouter un son spécifique lorsque le serpent consomme un fruit.
- 1603 : Implanter un son distinctif pour signaler la fin du jeu.

3. Planification et affectation des tâches

Id	Nom de tâche	Date de début	Date de fin	Etat de la tâche
2301	Mettre en place la logique de collision entre le serpent et les murs et lui même.	15/11/2023	20/11/2023	Terminée
2302	Déclencher automatiquement la fin du jeu lorsqu'une collision est détectée	16/11/2023	16/11/2023	Terminée
2303	Établir un mécanisme de collision entre le serpent et les fruits.	20/11/2023	21/11/2023	Terminée
2304	Gérer l'apparition de nouveaux fruits lorsque le serpent consomme un fruit.	21/11/2023	22/11/2023	Terminée
2401	Mettre en œuvre un mécanisme permettant au serpent de se déplacer d'un bord de l'écran à l'autre lorsque sa tête atteint les limites.	17/11/2023	26/11/2023	Annulé
1601	Intégrer un fond sonore pour l'ambiance du jeu.	23/11/2023	24/11/2023	Terminée
1602	Ajouter un son spécifique lorsque le serpent consomme un fruit.	24/11/2023	25/11/2023	Terminée

Id	Nom de tâche	Date de début	Date de fin	Etat de la tâche
1603	Implanter un son distinctif pour signaler la fin du jeu.	24/11/2023	25/11/2023	Terminée

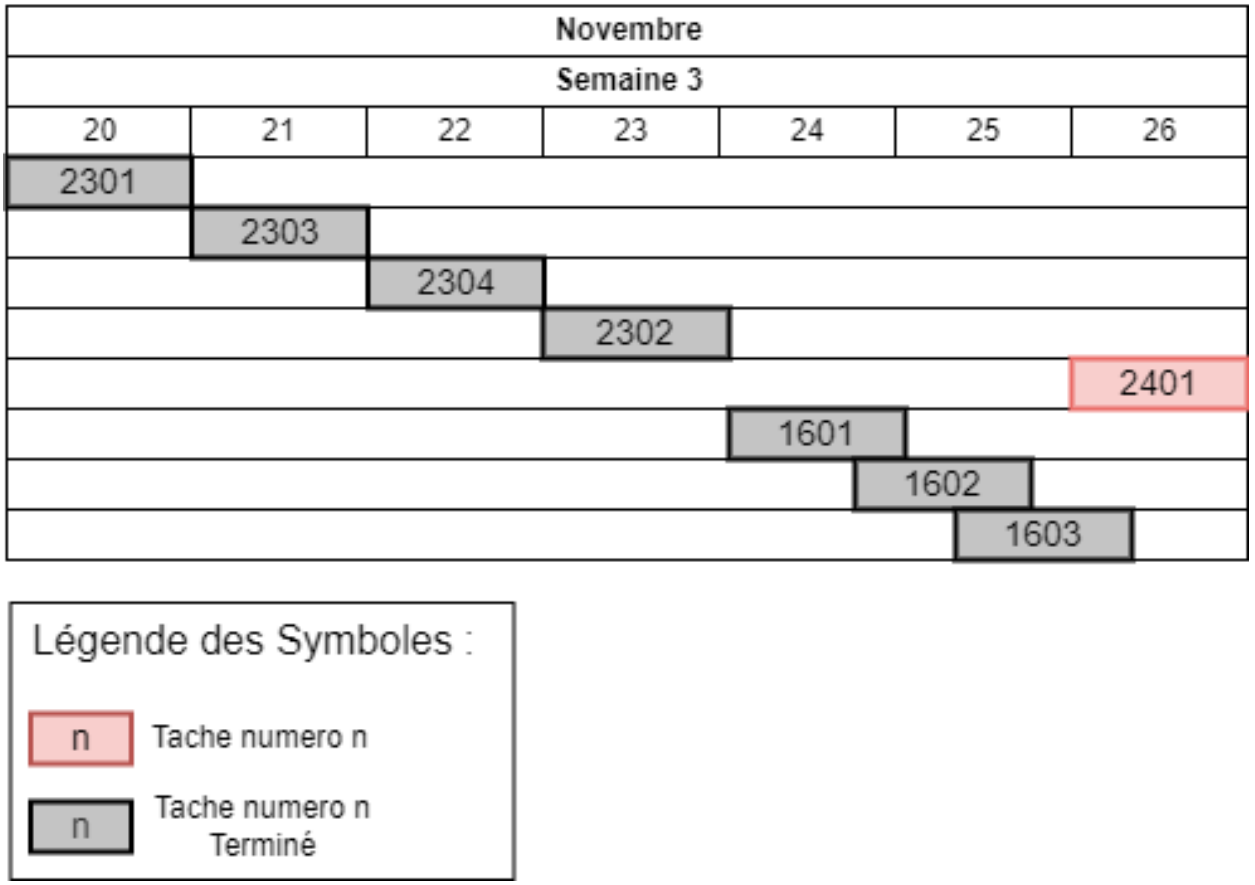
4. Diagramme de Gantt prévisionnel :

Diagramme De Gantt

Projet : GL-TP2

Version : 6

Date : 25/11/2023



Projet : Création d'un jeu vidéo 2D Snake

Version du document : 3

Propriétaire du document : Equipe 22

Date de Création : 26/11/2023

Document de Spécification de moteurs de Jeux Vidéos.

Introduction

Ce projet consiste en le développement d'un jeu vidéo en deux dimensions (2D), mettant en place les moteurs physique et graphique. Ces moteurs permettent de simuler le déplacement des objets du jeu et d'afficher les éléments visuels du jeu, en utilisant des principes physiques tels que la position, la vélocité, et l'accélération.

Ce document a pour but de définir les spécifications détaillées du moteur physique, graphique, entrée et noyau du jeu vidéo 2D, ainsi que la couche GamePlay dédiée spécialement au jeu Snake.

I. Moteur physique :

Un moteur physique dans le développement d'un jeu est un composant logiciel qui simule les lois de la physique dans le contexte du jeu. Pour gérer les mouvements, les collisions et les réactions des objets et des entités du jeu, en tenant compte de principes physiques tels que la gravité, la vélocité, l'accélération, et les forces.

1. Espace du jeu :

L'espace du jeu est un monde en deux dimensions où les événements se déroulent en suivant les règles.

2. Objets et Entités :

Les objets et les entités représentent les éléments du jeu, tels que les personnages, les ennemis, les objets, murs, obstables. Chaque objet est caractérisé par sa position dans un espace bidimensionnel, sa vitesse, son accélération, sa largeur, sa hauteur et sa masse. Les objets peuvent se déplacer en mettant à jour leur position en fonction de leur vitesse et de leur accélération.

3. Dynamique des Objets :

Le moteur doit permettre la simulation de mouvement des objets, en prenant en compte la position, la vitesse, l'accélération et les dimensions.

- **Position :** L'objet a une position dimensionnelle (x, y) dans l'espace du jeu, représentée par un vecteur.
- **Vitesse :** Il possède également une vitesse (x, y),présentée par un vecteur.
- **Accélération** : L'accélération (x, y) est utilisée pour modifier la vitesse de l'objet et est stockée dans un vecteur.
- **Dimensions :** L'objet a une largeur et une hauteur qui définissent sa taille dans l'espace du jeu.
- **Masse :** Une masse (non visible dans la classe actuelle) peut être ajoutée pour des calculs liés à la physique).

Le déplacement des objets s'effectue en suivant un processus simple :

Mise à Jour de la Vitesse :

Une méthode ajuste la vitesse de l'objet en fonction de l'accélération. Elle ajoute la composante x de l'accélération à la vitesse x de l'objet et la composante y de l'accélération à la vitesse y de l'objet.

Mathématiquement : `velocity$_x$ += acceleration$_x$` et `velocity.y += acceleration.y`.

Mise à Jour de la Position :

La méthode met à jour la position de l'objet en fonction de sa vitesse. Elle ajoute la composante x de la vitesse à la position x de l'objet et la composante y de la vitesse à la position y de l'objet.

Mathématiquement : position.x += velocity.x et position.y += velocity.y.

Ces opérations sont effectuées à chaque itération du moteur du jeu, permettant ainsi de simuler le mouvement de l'objet en fonction de sa vitesse et de son accélération.

Cela signifie que l'accélération influe sur la vitesse, qui à son tour influence la position. En d'autres termes, l'accélération représente le changement de vitesse au fil du temps. Ce processus est itératif, ce qui signifie qu'il se répète à chaque itération du jeu, permettant ainsi aux objets de se déplacer de manière dans l'espace 2D.

4. Réaction aux Forces Extérieures :

Les objets doivent réagir aux forces extérieures, par exemple une rafale de vents, forçant les objets à changer de direction.

5. Gestion de Collision :

Le moteur doit gérer efficacement les collisions entre objets. Lorsque deux entités se superposent ou entrent en contact :

- *Détection de Collision :* Le moteur doit être capable de détecter les collisions entre les objets du jeu, tels que des objets traversable ou pas.

- *Réponse aux Collisions :* Le moteur doit fournir des mécanismes pour gérer la réaction aux collisions, y compris la résolution de la collision (rebond, pénétration minimale) et la gestion des dégâts.

6. Gestion de la gravité:

La gravité est un élément clé dans la simulation physique. Le moteur applique la force de gravité pour faire en sorte que les objets tombent vers le bas, créant ainsi un réalisme dans le mouvement.

II. Moteur graphique :

Le moteur graphique a pour objectif principal de convertir les données du jeu en une représentation visuelle attrayante et cohérente. Il se charge de traduire les informations sur les objets, les décors, et les actions du jeu en images affichées à l'écran, contribuant ainsi à créer un univers visuel captivant pour les joueurs.

1. **Création d'objets graphiques:** Le moteur permet la création et la gestion d'objets graphiques, définis par leurs dimensions, positions, couleurs, et éventuellement des textures.

2. **Animations :** Le moteur doit prendre en charge des systèmes pour la création d'effets spéciaux. Elles sont utilisées dans une variété de contextes, tels que le mouvement des personnages, les actions spéciales, les réactions aux événements.

3. **Affichage :** Ce moteur se charge d'afficher les objets sur une fenêtre.

4. **Gestion des textures :** Le moteur prend en charge l'application de textures aux objets graphiques, permettant d'ajouter des détails visuels réalistes.

III. Moteur d'Entrées :

Le module d'entrée a pour objectif de gérer les interactions utilisateur avec le jeu en capturant et traitant les entrées, principalement à partir du clavier. Il offre une interface pour mapper des événements d'entrée à des actions spécifiques dans le jeu.

1. **Gestion des Événements Clavier :** Le module prend en charge la gestion des événements liés au clavier, notamment les pressions de touches.

2. **Mappage d'Événements :** Il permet le mappage d'événements clavier spécifiques à des actions du jeu, facilitant la personnalisation des commandes.

3. **Structure d'Événements :** Le module utilise une structure d'événements pour encapsuler les informations sur les entrées utilisateur, fournissant des détails tels que la touche pressée.

IV. Moteur de Son :

Le moteur sonore du jeu a pour but de gérer les éléments audio, renforçant ainsi l'aspect immersif de l'expérience de jeu. Cette composante spécifique est responsable du chargement, de la lecture, de la pause, de la reprise, de la réinitialisation et de l'arrêt des pistes sonores.

V. Noyau :

Le noyau (Kernel) a pour objectif de coordonner les différents composants du jeu, notamment les moteurs graphique et physique, le gestionnaire d'entrée, et les objets de jeu, pour assurer le bon fonctionnement et l'interaction cohérente entre ces éléments.

Le noyau donc est responsable de la gestion des aspects essentiels du jeu, la physique, le graphique et la gestion des entrées du jeu pour :

- Fournir une infrastructure solide pour la création d'objets.
- Faciliter l'intégration de la logique du jeu, des interactions utilisateur et des graphismes.

1. Composants:

Le noyau d'un jeu est composé des éléments suivants :

- Moteur physique
- Moteur graphique
- **InputHandler :** Un gestionnaire d'entrées pour gérer les actions de l'utilisateur.

2. Fonctionnalités:

Le noyau permet :

- La création et la gestion d'objets.
- La mise à jour de l'état du jeu.
- D'assurer la gestion des entrées utilisateur, incluant la prise en charge des commandes pour déplacer un objet.
- De mettre en pause et de reprendre le jeu en modifiant l'échelle temporelle.

Mais également :

- ✓ *Gestion des Composants :* Le noyau gère les instances des moteurs graphique et physique, le gestionnaire d'entrée, ainsi que les objets de jeu.
- ✓ *Intégration des Moteurs :* Il intègre les moteurs graphique et physique pour synchroniser les aspects visuels et physiques du jeu.
- ✓ *Gestion des Entrées :* Le noyau prend en charge la gestion des entrées utilisateur en utilisant le gestionnaire d'entrée, permettant la liaison d'événements à des actions spécifiques.
- ✓ *Coordination des Objets de Jeu :* Il coordonne la création, la mise à jour et la gestion des objets de jeu, assurant une interaction harmonieuse entre les composants.

3. Couche Game Play – Le jeu Serpent :

La couche gameplay du jeu Snake a pour objectif de mettre en œuvre la logique spécifique du jeu, notamment le déplacement du serpent, la gestion du corps du serpent, les interactions avec l'utilisateur, et le lancement du jeu dans le contexte du noyau.

- ✓ *Initialisation du Serpent :* La couche gameplay initialise le serpent avec une tête colorée, des textures spécifiques, et une vélocité initiale.
- ✓ *Gestion du Corps du Serpent :* Elle gère la croissance du serpent en ajoutant de nouveaux segments au corps à chaque mise à jour.
- ✓ *Gestion des collisions du Serpent :* Le serpent doit réagir à la collision avec d'autres objets, tels que lui-même, qui forcera le jeu à se terminer, ou un objet à points, qui lui permettra de gagner des points.

Conclusion :

Cette spécification fournit un aperçu des caractéristiques, de l'architecture, du fonctionnement des différents moteurs d'un jeu en 2D ainsi que la couche gameplay du jeu Snake, détaillant sa contribution à l'expérience de jeu globale.

Projet : Création d’un jeu vidéo 2D Snake
Version du document : 3

Propriétaire du document : Equipe 22

Date de Création : 24/11/2023

Document de Conception de moteurs de Jeux Vidéos.

I. Introduction :

Concevoir les moteurs d'un jeu vidéo équivaut à sculpter l'intelligence même du jeu, une tâche à la fois cruciale et créative. C'est ici que les méthodes agiles, en particulier Scrum, se révèlent indispensables. Scrum offre une flexibilité précieuse pour travailler sur les aspects clés du jeu, tels que les graphismes, les mouvements, et l'interaction utilisateur. Ces moteurs agissent comme les maîtres d'œuvre du jeu, façonnant chaque détail pour créer une expérience immersive et innovante. Son agilité permet également de s'ajuster aux changements tout au long du projet.

Un document de conception des moteurs et du noyau est incontournable pour planifier et détailler l'architecture, les fonctionnalités, et les composants clés du jeu.

Ce présent document détaille par conséquent la conception des moteurs nécessaires pour le développement d’un jeu vidéo. Les moteurs inclus dans cette conception sont le moteur physique, le moteur graphique, le moteur d’input, le moteur de son et le noyau.

II. Architecture Générale :

L'architecture générale d'un jeu vidéo repose sur un ensemble de moteurs essentiels, chacun ayant un rôle spécifique. Le moteur physique gouverne les lois du mouvement et les interactions physiques, assurant un réalisme saisissant. Le moteur graphique façonne l'aspect visuel du jeu, des graphismes aux animations, créant un univers visuel captivant. Le moteur d'Input permet une interaction fluide entre le joueur et le jeu en gérant les commandes. Intégrer le moteur de son à cette structure complexe enrichit davantage l'expérience de jeu en gérant tous les aspects sonores. Ce moteur, dédié à la reproduction des effets sonores et des musiques d'ambiance joue un rôle crucial pour créer une atmosphère immersive. Enfin, le kernel, cœur opérationnel, coordonne l'ensemble, assurant la cohérence globale de l'expérience de jeu. Chaque moteur, spécialisé dans sa fonction, collabore étroitement avec le kernel, créant une architecture qui offre une immersion complète, où chaque composant contribue de manière cruciale à l'expérience globale du joueur.

III. Moteur Physique :

- ✓ **Objectif :**

Le moteur physique constitue le fondement du réalisme dans un jeu vidéo. Son objectif global est de créer un environnement de jeu où les objets interagissent physiquement de manière cohérente avec les lois du monde réel.

Sa principale fonction est de simuler les lois de la physique, apportant une dimension tangible aux mouvements des objets. Son rôle ainsi est de calculer les mouvements, les collisions, et les réponses à ces dernières, contribuant ainsi à créer une expérience de jeu immersive et crédible.

En fournissant une simulation physique réaliste, le moteur physique contribue à l'immersion du joueur en rendant les mouvements, les collisions et les interactions visuellement et mécaniquement plausibles.

- ✓ **Architecture :**

Le moteur physique constitue le pilier fondamental d'un jeu vidéo, modélisant les mouvements et les interactions entre objets. Il est construit autour d'une architecture modulaire comprenant plusieurs classes, offrant une flexibilité et une extensibilité remarquables. Chaque classe est conçue pour s'intégrer de manière transparente dans le moteur physique, facilitant la maintenance et les futures extensions du système. Ces composants travaillent en tandem pour offrir une simulation physique réaliste, formant ainsi la base solide de l'environnement du jeu.

1) Classe Vector2 :

La classe `Vector2` représente un vecteur bidimensionnel, utilisé pour définir les coordonnées (x, y) de la position, de la vitesse et de l'accélération des objets physiques.

2) Classe PhysicObject :

La classe `PhysicObject` représente les objets physiques dans le jeu. Elle détient des variables telles que la position, la vitesse, l'accélération, la largeur, la hauteur et la masse.

Cette classe offre des méthodes pour mettre à jour la position en fonction de la vitesse et ajuster la vitesse en fonction de l'accélération. Chaque objet physique est également associé à une instance de la classe `Bounds` pour gérer ses limites spatiales.

La classe principale, `PhysicObject`, encapsule les caractéristiques fondamentales des objets physiques tels que la position, la vitesse, et l'accélération. Cette classe expose des méthodes telles que `update()` qui, lors de son appel, ajuste dynamiquement la vitesse et la position en fonction de l'accélération, assurant ainsi un réalisme dans les mouvements des objets.

3) Classe Bounds :

La classe `Bounds` est responsable de la gestion des limites spatiales d'un objet physique.

Elle détermine les limites d'un objet en fonction de sa largeur, de sa hauteur et de sa position, fournit des méthodes pour obtenir les coordonnées des points centraux et des bords de l'objet, ainsi qu'une méthode `updateBounds()` pour actualiser dynamiquement ces limites en fonction des propriétés de l'objet physique.

4) Classe PhysicEngine :

Le moteur physique est géré par la classe `PhysicEngine`, qui maintient une liste d'objets physiques. Cette classe permet la création de nouveaux objets physiques via la méthode `createPhysicObject` et assure la mise à jour de la physique de l'ensemble des objets par l'intermédiaire de la méthode `updateEngine`. Elle illustre l'application du principe de modularité avec une séparation claire des responsabilités.

La classe assure également la gestion des collisions avec les méthodes `checkCollisionSelf` (entre un objet et lui-même) et `checkCollision`, facilitant la détection de collections entre les objets physiques. Elle maintient une collection d'objets physiques à travers la liste `physicObjects`, offrant des opérations telles que la suppression d'un objet physique avec la méthode `removePhysicObject`

Le `PhysicEngine` agit ainsi comme le gestionnaire central de tous les objets physiques du jeu.

✓ Diagramme de classe :

Le diagramme suivant, un diagramme de classe propre au moteur physique, représente l'architecture adoptée pour le mettre en place :

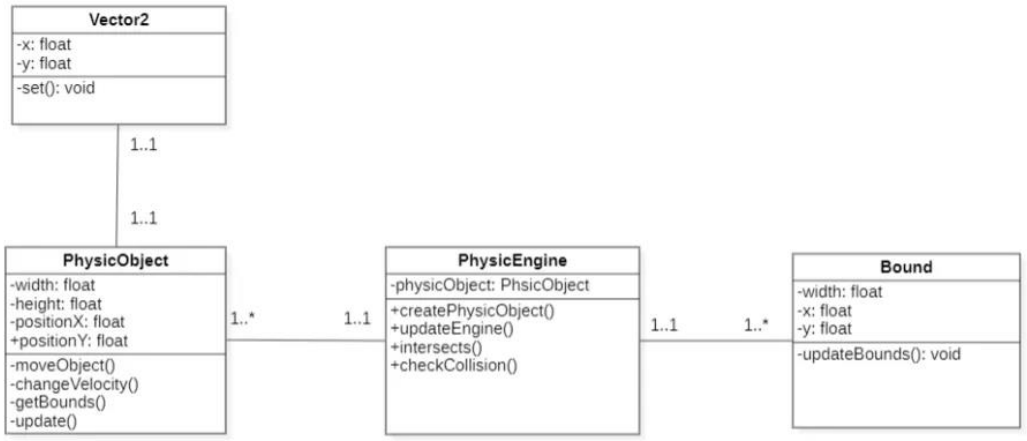


Figure 1: Diagramme de classe - Moteur Physique.

IV. Moteur Graphique :

✓ Objectif :

Le moteur graphique, pilier essentiel d'un jeu vidéo, vise à donner vie à l'esthétique visuelle de l'expérience ludique. Son objectif fondamental consiste à traduire les données du jeu en une représentation visuelle cohérente et attrayante. En d'autres termes, il transforme les informations sur les objets, les décors et les actions du jeu en images affichées à l'écran. À travers l'utilisation de techniques avancées de rendu graphique, ce moteur orchestre l'illumination, les ombres, les textures et les effets spéciaux pour créer un univers visuel captivant. En résumé, le moteur graphique contribue

fondamentalement à l'aspect visuel du jeu, permettant aux joueurs de s'immerger pleinement dans un monde visuellement vivant.

- ✓ **Architecture :**

L'architecture du moteur graphique présenté suit un modèle de conception modulaire basé sur des classes distinctes, chacune ayant une responsabilité spécifique. Il se compose des classes suivantes :

1) Classe Vector2 :

La classe Vector2 joue un rôle fondamental dans la représentation des positions dans l'espace bidimensionnel du moteur graphique. En tant que conteneur pour les coordonnées x et y, elle offre une solution simple et efficace pour manipuler les emplacements des objets graphiques. Cette classe, bien qu'apparemment modeste, fournit une base essentielle pour la localisation précise des éléments visuels au sein du jeu.

2) Classe GraphicObject :

Le cœur visuel du moteur graphique réside dans la classe GraphicObject. Responsable de la représentation graphique d'objets dans le jeu, elle encapsule des attributs tels que la largeur, la hauteur, la position, la couleur, et éventuellement, la texture. L'objet peut être représenté soit par un rectangle coloré, soit par une image chargée à partir d'une texture. La classe permet de mettre à jour la position de l'objet, de créer une représentation visuelle de celui-ci, et de redimensionner cette représentation. En utilisant des composants graphiques de JavaFX, cette classe offre une flexibilité permettant la création d'objets visuels diversifiés, allant des simples formes géométriques aux images texturées plus complexes.

3) Classe GraphicEngine :

La classe GraphicEngine agit comme le gestionnaire central de tous les objets graphiques du jeu. Elle est responsable de leur création, de leur stockage et de leur mise à jour régulière. Cette classe favorise une approche modulaire en permettant d'ajouter, de retirer et de mettre à jour des GraphicObjects de manière dynamique. Grâce à cette organisation, le moteur graphique peut évoluer efficacement avec une variété d'objets tout en maintenant une structure claire et organisée.

4) Classe GameFrame :

La fenêtre de jeu est orchestrée par la classe GameFrame. Celle-ci offre une interface utilisateur graphique via JavaFX, intégrant la scène où les objets graphiques sont affichés. En tant que point d'entrée de l'application, elle crée une fenêtre JavaFX, initialise la scène, et lance une boucle de jeu basée sur le temps pour maintenir un taux d'images par seconde constant. La classe permet également d'ajouter des objets graphiques à la scène. La classe GameFrame agit comme une interface entre le moteur graphique et l'aspect visuel du jeu.

5) Boucle de Jeu :

Au cœur du moteur graphique, la boucle de jeu orchestre le rythme du rafraîchissement visuel. En gérant le temps et en déclenchant la mise à jour régulière du moteur graphique, elle maintient la cohérence temporelle du jeu. La boucle de jeu dans GameFrame s'assure que le moteur graphique est mis à jour régulièrement, fournissant une animation fluide et constante.

Cette architecture favorise une séparation claire des responsabilités, rendant chaque classe indépendante dans son domaine. La classe GameFrame agit comme un point d'entrée de l'application, coordonnant l'initialisation de la fenêtre et le lancement de la boucle de jeu. La boucle de jeu assure

un rafraîchissement régulier du moteur graphique, qui, à son tour, gère la création et la mise à jour des objets graphiques.

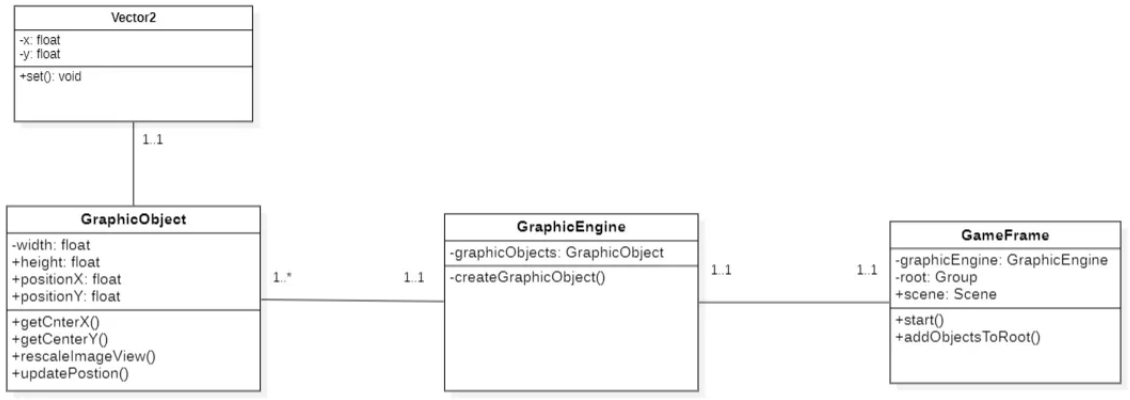


Figure 2: Diagramme de classe - Moteur Graphique.

V. Moteur d'Entrées (Input) :

- ✓ **Objectif :**

Le module d'entrée a pour but de rendre le jeu facile à contrôler. Il capture les touches du clavier, puis les transforme en commandes compréhensibles par le jeu. Il permet au joueur de diriger et d'interagir avec le jeu de manière naturelle, offrant une expérience de jeu agréable et personnalisable.

- ✓ **Architecture :**

L'architecture du module d'entrée repose sur deux classes principales : `InputHandler` et `KeyBinds`.

- La classe `InputHandler` agit comme le gestionnaire central des entrées. Elle utilise une structure de données `HashMap` pour mapper les codes d'événements (les touches du clavier) à des actions spécifiques définies par des objets `Runnable`. Ainsi, chaque événement d'entrée est associé à une action particulière, permettant une gestion flexible et personnalisée des commandes.

- La classe `KeyBinds` étend la classe `KeyEvent` de JavaFX et représente une liaison clé personnalisée. Elle encapsule les informations relatives à un événement clavier, telles que le code de touche et le caractère associé. Cette classe est utilisée pour créer des objets `KeyEvent` personnalisés dans le contexte du jeu.

Cette architecture permet une gestion flexible et réactive des commandes de jeu.

VI. Moteur de Son :

- ✓ **Objectif :**

Le moteur de son vise à gérer la lecture et le contrôle d'effets sonores dans un environnement de jeu. Il est conçu pour être intégré dans un système de jeu afin d'améliorer l'expérience immersive en fournissant une gestion efficace des effets sonores en réponse aux événements du jeu.

✓ **Architecture :**

Le moteur de son `SoundEngine` utilise la bibliothèque Java Sound pour créer et manipuler des clips audio, représentés par des instances de la classe `Clip`. L'architecture modulaire du moteur se compose de différentes fonctions de contrôle telles que `play`, `pause`, `resumeAudio`, `restart`, et `stop`, offrant un contrôle précis sur la lecture des clips sonores. La méthode `load` permet de charger de nouveaux fichiers audio, assurant une polyvalence dans le choix des sources sonores. La réinitialisation du flux audio à l'aide de la méthode `resetAudioStream` facilite le chargement de nouveaux clips sans avoir à recréer l'instance du moteur. Cette conception modulaire favorise l'intégration facile du moteur de son dans divers contextes de jeu, fournissant ainsi une expérience sonore immersive et adaptable.

VII. Noyau (Kernel) :

✓ **Objectif :**

L'objectif du noyau (`Kernel`) dans un jeu vidéo est d'assurer la cohérence et la synchronisation entre les différents moteurs du jeu, notamment le moteur physique, le moteur graphique et le moteur d'entrée. Le noyau agit comme une entité centrale qui orchestre l'ensemble du système, permettant une interaction fluide entre les éléments du jeu.

Le noyau sert de point de rencontre pour les moteurs individuels. Il gère l'initialisation, la mise à jour et la communication entre le moteur physique, le moteur graphique et le moteur d'entrée.

En réceptionnant les événements du moteur d'entrée, le noyau déclenche des actions appropriées dans les autres moteurs. Par exemple, la pression d'une touche peut déclencher un mouvement dans le moteur physique et une réaction visuelle dans le moteur graphique.

✓ **Architecture :**

L'architecture du noyau se compose de plusieurs classes qui interagissent pour assurer le fonctionnement global du jeu.

1) Classe CoreKernel :

Le `CoreKernel` sert de noyau central du jeu, coordonnant les moteurs graphiques et physiques, la gestion des entrées et la boucle principale du jeu.

Le `CoreKernel` lui-même agit comme une entité centrale qui orchestre le fonctionnement global du jeu. Il intègre le `physicEngine` pour gérer la simulation des lois physiques, le `graphicEngine` chargé de représenter visuellement le jeu, et l'`inputHandler` pour capturer les événements du clavier. Le `timeScale` ajustable offre un moyen de réguler la vitesse du jeu, permettant de le mettre en pause ou de le reprendre.

Cette classe constitue le cœur du système de jeu, coordonnant les différentes composantes telles que le moteur physique (`physicEngine`), le moteur graphique (`graphicEngine`), et le gestionnaire

d'entrées (`inputHandler`). Avec la capacité de mettre en pause le jeu grâce à la propriété `timeScale`, le `CoreKernel` offre un contrôle sur la temporalité du jeu (de réguler la vitesse du jeu, permettant de le mettre en pause ou de le reprendre).

La méthode `gameLoop()` représente le moteur du jeu, orchestrant la séquence d'actions à chaque frame. La création d'objets de jeu est simplifiée par les méthodes `createGameObject(...)`, établissant une liaison cohérente entre les composants physique et graphique.

2) Classe GameObject :

Les objets du jeu sont représentés par la classe `GameObject`, qui combine un `PhysicObject` pour la simulation physique et un `GraphicObject` pour la représentation graphique. Cette conception favorise une séparation claire des responsabilités entre les moteurs physique et graphique, facilitant la maintenance et l'extension du système.

La méthode `updateGameObject()` assure la synchronisation entre les aspects physique et graphique de l'objet, contribuant à maintenir une cohérence entre ces deux dimensions essentielles du jeu.

VIII. Couche GamePlay:

✓ **Objectif**

L'objectif général de la couche gameplay est de définir et mettre en œuvre les mécanismes fondamentaux qui créent l'expérience interactive pour les joueurs. Cette couche est au cœur du jeu, déterminant comment les différentes composantes du jeu interagissent entre elles pour offrir une expérience ludique et captivante.

La couche gameplay forme l'épine dorsale du jeu, transformant des éléments statiques tels que les graphismes et les moteurs en une expérience interactive et dynamique.

✓ **Architecture :**

La couche gameplay du jeu Snake, implémentée dans la classe `Main`, démontre une conception organisée et orientée objet. La classe coordonne l'initialisation du noyau via l'objet `coreKernel`, la création du serpent avec des textures spécifiques et une vélocité initiale, la gestion des entrées utilisateur avec des mappages clavier, et le lancement du jeu. Une fonction `update` encapsule la logique de déplacement du serpent, implémentée par le noyau, et la mise à jour de la position de chaque segment de son corps. L'ajout de nouveaux segments par la fonction `addBodySegment` et le positionnement sur une grille avec `moveToGridPosition` contribuent à la mécanique de jeu du serpent. L'ensemble reflète une structuration claire des composants de jeu, assurant une interaction cohérente avec le noyau du jeu `CoreKernel' et une expérience de jeu plaisante. L'utilisation du noyau permet une séparation des préoccupations, favorisant la modularité et la maintenabilité du code. Chaque fonction est définie de manière à encapsuler une partie spécifique du comportement du serpent, illustrant ainsi une approche modulaire et orientée objet pour le développement du jeu Snake.

IX. Diagrammes de conception :

- **Diagramme de packages :**

Le diagramme de packages suivant résume l'architecture suivie :

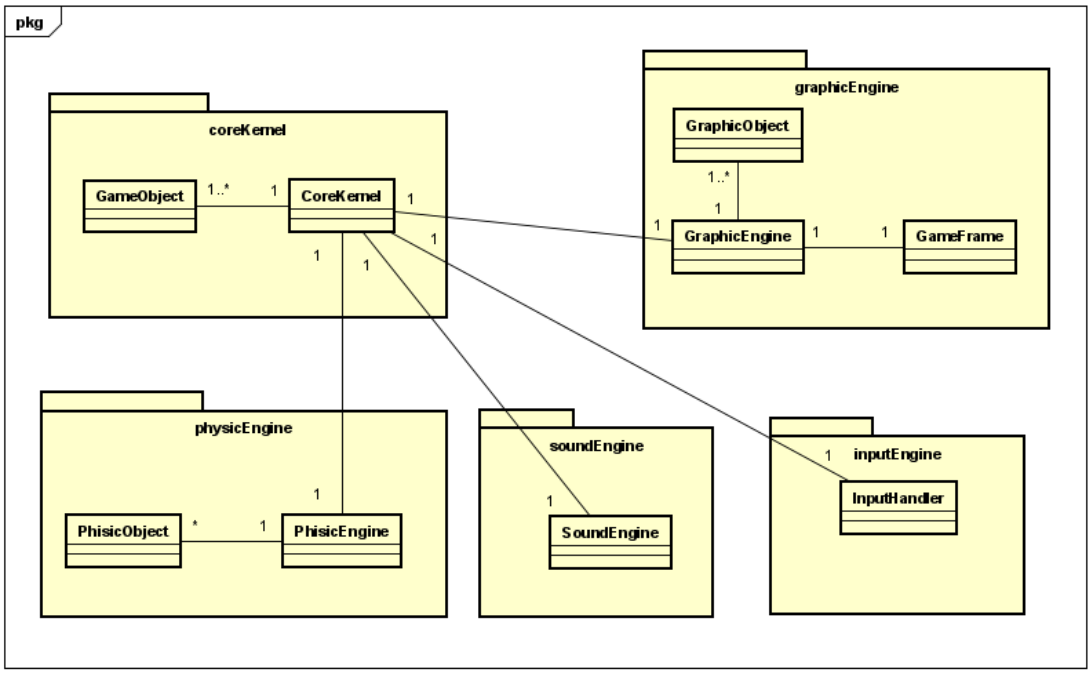


Figure 3: Diagramme de package.

Chaque package représente un moteur, tels que tous les moteurs sont reliés au noyau, et sont indépendants les uns des autres.

- **Diagramme de séquence :**

Le diagramme de séquence suivant illustre le lien entre le noyau et les moteurs : une instanciation du core Kernel mène à une instanciation du moteu graphique, physique et d'entrées. Cette meme instanciagion du noyau sera appelée dans le programme principale, la couche gameplay (notre main) pour l'exploiter dans le but d'implémenter le jeu.

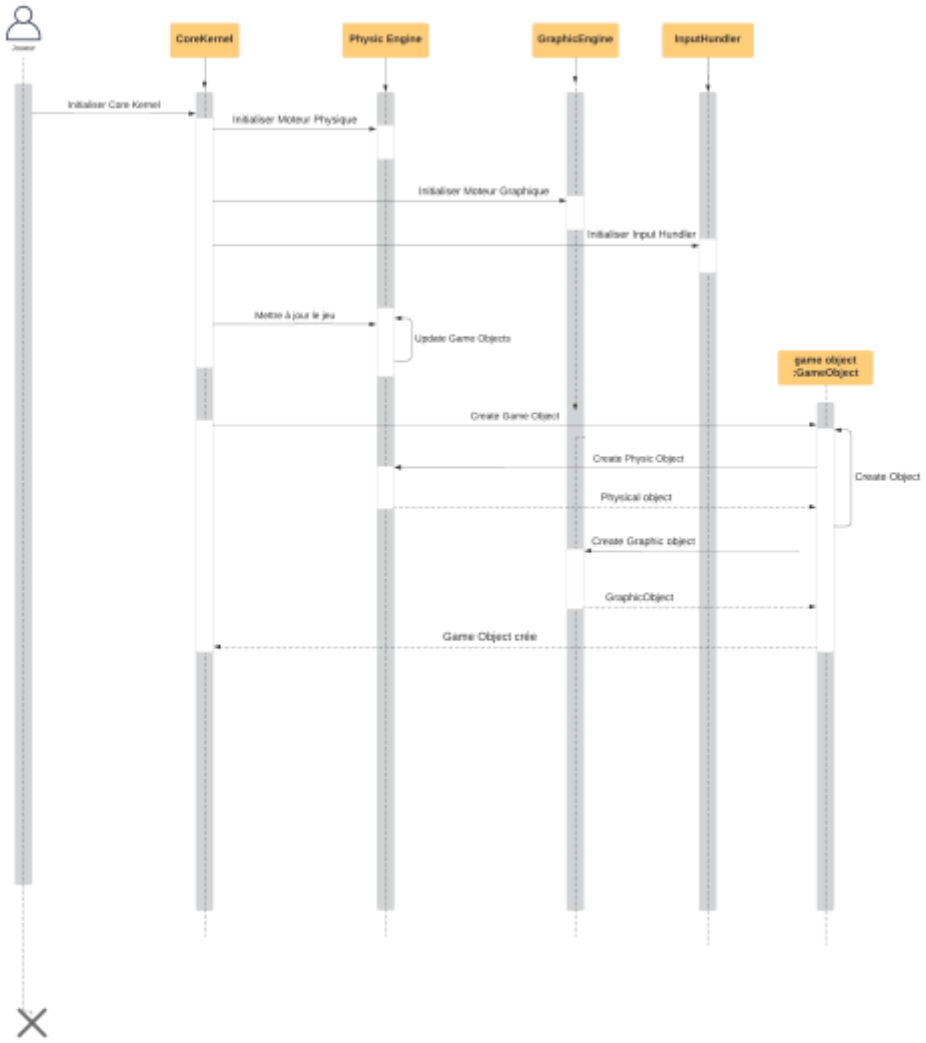


Figure 4: Diagramme de séquence - Lancement du jeu.

- **Diagramme de classe :**

Le diagramme de classe ci-dessous offre une vue synthétique de l'architecture du système de jeu vidéo, détaillant la relation entre les moteurs spécialisés et le noyau central. Chaque moteur, tel que le moteur physique, le moteur graphique et le moteur d'Input, est représenté par une collection de classes dédiées, illustrant ainsi la modularité du système. Ces moteurs interagissent de manière directe avec le noyau, coordonnant leurs activités sans établir de liaisons directes entre eux. Cette approche favorise la séparation des préoccupations et permet une évolutivité du système sans compromettre sa cohérence globale. Chaque classe au sein des moteurs expose des fonctionnalités spécifiques, contribuant ainsi à la richesse fonctionnelle de l'ensemble. La clarté des relations entre les moteurs et le noyau, ainsi que l'absence de connexions directes entre les moteurs, démontrent une conception réfléchie orientée vers la modularité et la maintenabilité du système.

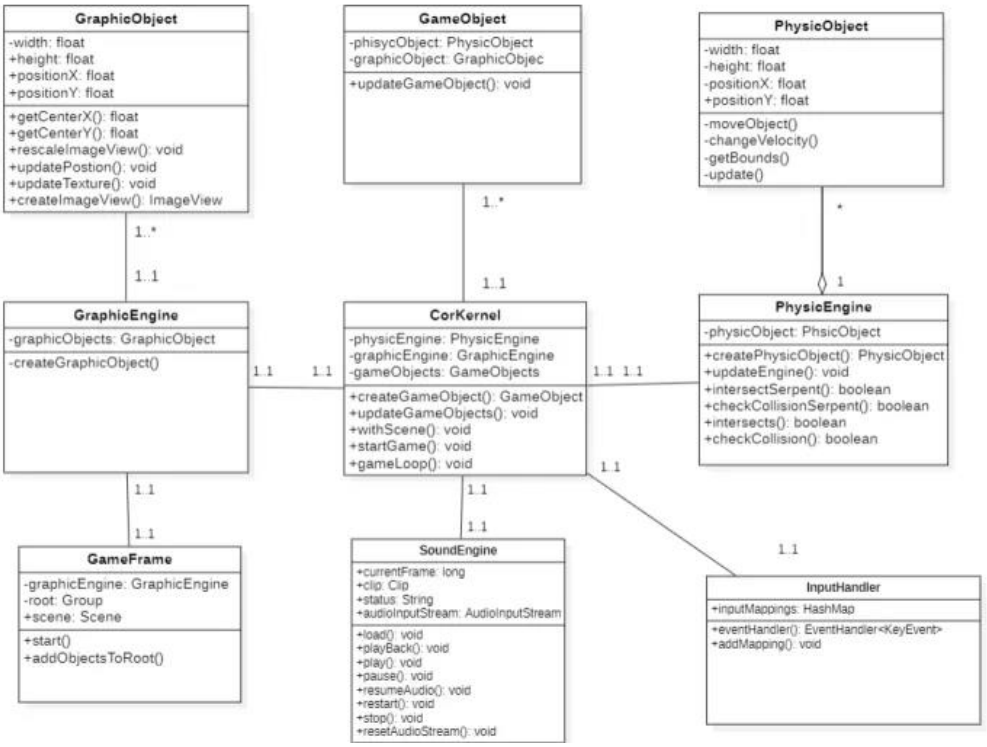


Figure 5: Diagramme de classe globale.

X. Conclusion :

Le jeu Snake présente une architecture soigneusement conçue, divisée en plusieurs couches fonctionnelles, chacune remplissant un rôle spécifique dans l'expérience de jeu. Le noyau du jeu (CoreKernel) agit comme une pièce maîtresse, gérant la physique, les graphiques et les entrées utilisateur. La couche gameplay, illustrée dans la classe `Main`, montre une implémentation élégante du comportement du serpent, avec des mécaniques claires de déplacement, de croissance et de mise à jour. L'utilisation de textures spécifiques et la coordination entre les objets graphiques et physiques contribuent à une représentation visuelle cohérente. L'approche modulaire permet une extensibilité facile du jeu, offrant la possibilité d'ajouter de nouvelles fonctionnalités ou de personnaliser le gameplay. Cette conception réfléchie facilite également la maintenance du code, en assurant une séparation claire des responsabilités entre les différentes couches du jeu. En somme, l'architecture du jeu Snake allie efficacement la simplicité et la flexibilité pour offrir une expérience de jeu captivante.

Projet : Création d'un jeu vidéo 2D

Version du document : finale.

Propriétaire du document : Equipe 22

Date de Création : 26/11/2023

Fiche de tests

L'objectif de cette fiche est d'archiver les tests qui ne peuvent pas être sauvegardés sous Junit. A moindre degré, elle peut servir de fiche de préparation des tests exécutables sous Junit.

1. Liste de fonctionnalités à tester du *product backlog*

- 1 : **En tant que** développeur de jeu, **je veux** pouvoir créer, supprimer, modifier des objets et que chaque objet ait des dimensions **pour** définir les éléments principaux de mon jeu.
- 2 : **En tant que** développeur de jeu, **je veux** que mes objets puissent se déplacer ou pas **pour** implémenter le mouvement dans le jeu.
- 5 : **En tant que** développeur de jeu, **je veux** un système de gestion de collision, **pour que** les objets du jeu interagissent d'une manière cohérente et réaliste.
- 6 : **En tant que** développeur de jeu, **je veux** afficher des sprites et des formes sur l'écran **pour** avoir une représentation graphique des objets.
- 12 : **En tant que** développeur, **je veux** implémenter la gestion des entrées clavier pour détecter les touches pressées, **pour** que les joueurs puissent interagir avec le jeu à l'aide du clavier.

2. Description des tests

ID	N°Action	Description de l'action	Résultat attendu	Résultat du test	Commentaires
1	101	Pour créer un objet physique.	Taille de la liste des objets physique est 1	S	Rien à signaler (RAS)

2	201	Mise à jour des paramètres d'un objet physique après avoir défini une vélocité et une accélération spécifiques.	Les paramètres des objets sont égaux aux nouveaux. paramètres affectés	S	
5	502	Détecter une collision entre deux objets physiques dont les rectangles de collision se chevauchent partiellement	checkcollision renvoi true pour le test de collision de deux objets ayant une même position	S	Rien à signaler (RAS)
6	602	Associer des textures (spirites) aux objets et les afficher dans la scène		Test non abouti.	
	603	Gérer la rotation des spirites pour une représentation visuelle.	les nouvelles positions de l'objet sont égaux aux positions qui lui sont affectées	S	Rien à signaler (RAS)
12	1201	Détecter les touches pressées	Renvoie True si le bouton est pressé.	S	Rien à signaler (RAS)
	1202	Associer des actions aux touches pressées.	Assurer que l'ajout d'un mapping pour une touche déjà mappée remplace correctement l'action	S	

			associée à cette touche.		