

The background image depicts three fantasy warriors standing in a dark, smoky battlefield. The warrior on the left is clad in ornate plate armor with a blue and gold helmet, holding a sword. The central warrior is a large, muscular figure wearing a horned helmet and fur-trimmed armor, holding a massive battle-axe. The warrior on the right wears a helmet with a plume and chainmail, holding a sword. Behind them are banners with various symbols, including a crown and a skull. The scene is filled with smoke and debris, suggesting a recent battle.

JEU DE PLATEAU TOUR PAR TOUR

RÉALISÉ PAR KAPNIST NGANTAH
ENCADRÉ PAR NICOLAS RIVIN



Rapport de projet technologique

N°6: Jeu de plateau tour par tour

Sommaire

1 cadrage du projet

1.2 Résumé

1.3 Enjeux et objectifs

1.4 Livrables

2 présentation de mes objets

2,0 plateau

2.1 joueurs

2.2 armes

2.3 blocks

2.4 La carte du jeu

3 architecture du jeu

3,2 fonction principales

3.3 fonction utiles

4 Diagramme UML

4.1 Diagrammes de packages

4.2 diagrammes de cas d'utilisations

a)Diagramme de cas d'utilisation générer mes objet

b)Diagramme de cas d'utilisation Mouvement

c)Diagramme de cas d'utilisation Combat

4.3 Diagrammes de séquences

a)Diagramme de séquence Mouvement

b)Diagramme de séquence Combat

1 Cadrage du projet.

1.2 Résumé.

Dans ce projet, il est question pour nous de créer un jeu en ligne en Javascript, dans lequel deux joueurs évolueront chacun leur tour pour s'affronter. A l'issue de ce jeu, il ne pourra en rester qu'un seul.

Sur une carte, deux joueurs, les armes(minimum 4), et les cailloux(impasses) seront générés de façons aléatoire.

Ces armes pourront être récoltés par les joueurs qui passent dessus.

L'arme par défaut qui équipe le joueur doit infliger 10 points de dégât.

1.2 Enjeux et objectifs.

L'objectif de cette application est de:

- Réalise un jeu multi joueurs en ligne
- Mettre en place une application Javascript et jQuery

1.3 les livrables.

Pour développer le jeu de plateau, Kapnist Ngantah fournira les éléments suivants.

- l'architecture du jeu réutilisable
- les diagrammes UML de cette application
- code source complet du projet

2 Présentation de mes objets

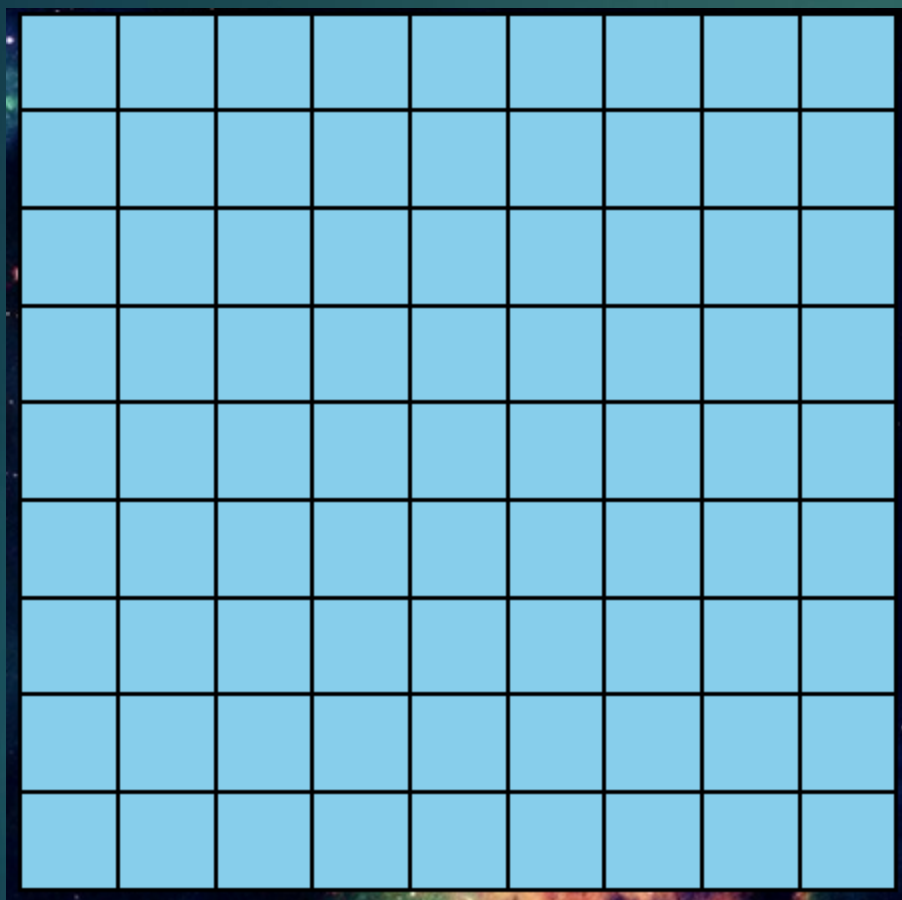
Afin d'assurer un fonctionnement optimal de notre jeu, il conviendrait de déterminer les différents éléments le constituant, ainsi nous pouvons noter:

- le plateau
- les blocks
- Kapnist(joueur 1)
- Genius(joueur 2)

les armes:

- le Sabre Egyptien
- Le Sabre magique
- la Hache de guerre
- Le Bouclier de Zelda

2 Présentation des objets



Plateau de jeu



Kapnist

Santé: excellente

Arme: Sabre Egyptien

Dégât arme: 10

Posture: attaque, défense

Point de vie: 100



Genius

Santé: excellente

Arme: Sabre Egyptien

Dégât arme: 10

Posture: attaque, défense

Point de vie: 100



Sabre Egyptien
(arme par default
de Kapnist)
défat: 10points



Sabre Magique
(arme par defat
de Genius)
degat: 10points



Bouclié
de Zelda
degat:
15points



Hâche de
guerre
degat:
20points

Nouvelle
partie

Jeu de plateau tour par tour



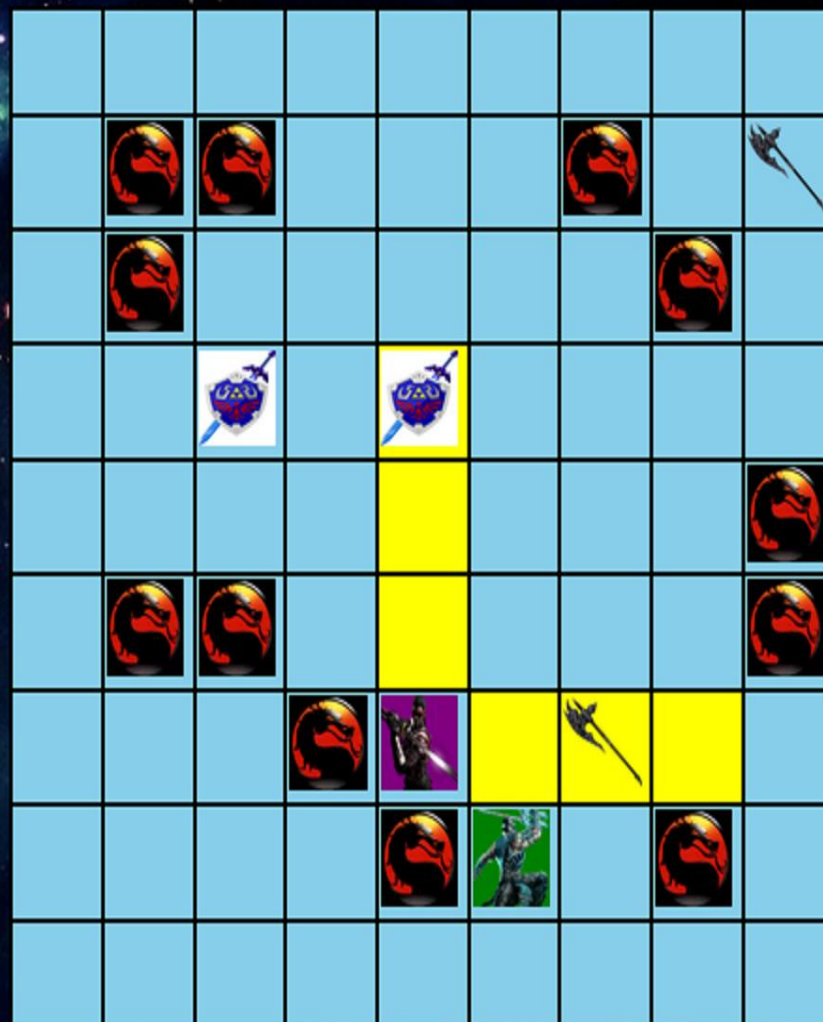
Sante: Excellente

Arme: Sarbre Magique

Degat arme: 10

Posture Attaque Défense

Point de vie: 100



Sante: Excellente

Arme: Sarbre Egyptien

Degat arme: 10

Posture Attaque Défense

Point de vie: 100

Block ou impasse



3 Architecture du jeu

Génération de mes objets

Fonction principales:

function armes() : permet de générer mes arme

Function placePlayer() qui permet de générer mes joueurs

Function block() qui permet de générer mes blocks(impasses)

Fonctions utiles:

getRandomNumber() permet de générer un nombre aleatoire entre un min et un max

Function ajoutClassCasesvides() qui permet d'ajouter la classe isAcces au cases vides,

function ajoutClassImpasses() qui permet d'ajouter la classe impasse à nos block,

Gestion des mouvements

- ▶ Fonctions principales:

- ▶ `eventPlayer1()` pour assurer le déplacement du joueur1 et le ramassage des armes.
- ▶ `eventPlayer2()` pour assurer le déplacement du joueur2 et le ramassage des armes.

- ▶ Fonctions utiles:

celles qui gèrent les cases cliquables

- ▶ `positionSpriteTop(posXJoueur1[0], posXJoueur1[1]);`
- ▶ `positionSpriteLeft(posXJoueur1[0], posXJoueur1[1]);`
- ▶ `positionSpriteRigth(posXJoueur1[0], posXJoueur1[1]);`
- ▶ `positionSpriteBottom(posXJoueur1[0], posXJoueur1[1]);`
- ▶ `Play()` qui gère le bruitage

Gestion du combat

- ▶ Fonctions principales

- ▶ figth() qui déclanche le combat si les deux joueurs sont cote à cote
- ▶ combat() qui gère le combat
- ▶ Attaque1 () et attaque2() qui permettent d'attaquer pour les joueurs1 et joueur2
- ▶ défense1 () et défense2() qui permettent de défendre pour les joueurs1 et joueur2

- ▶ Fonctions utilises

- ▶ getRandomNumber() qui permet de savoir qui engage le combat
- ▶ Play() qui gère le bruitage

4-Diagrammes UML

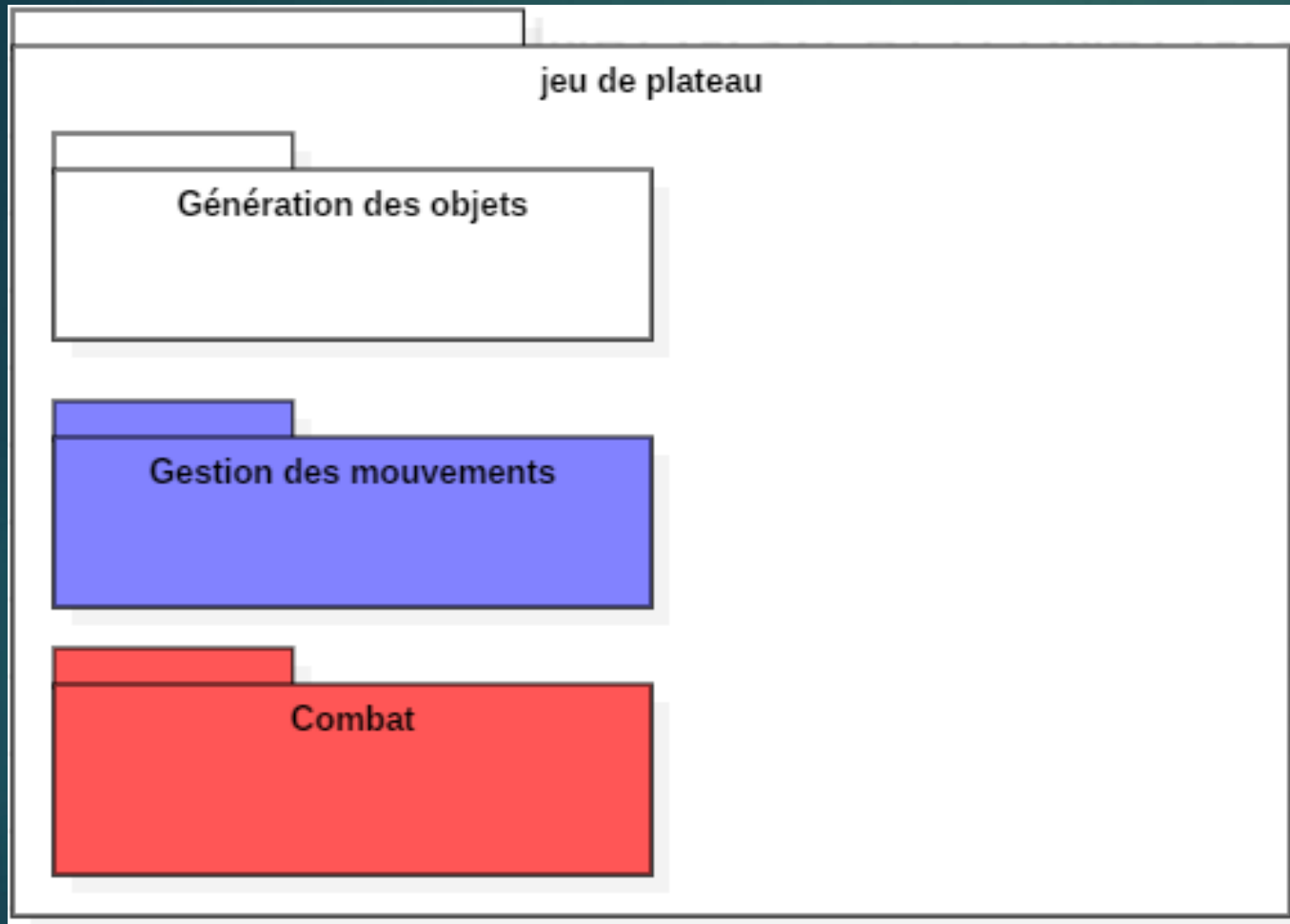


Diagramme de package

Génération objets

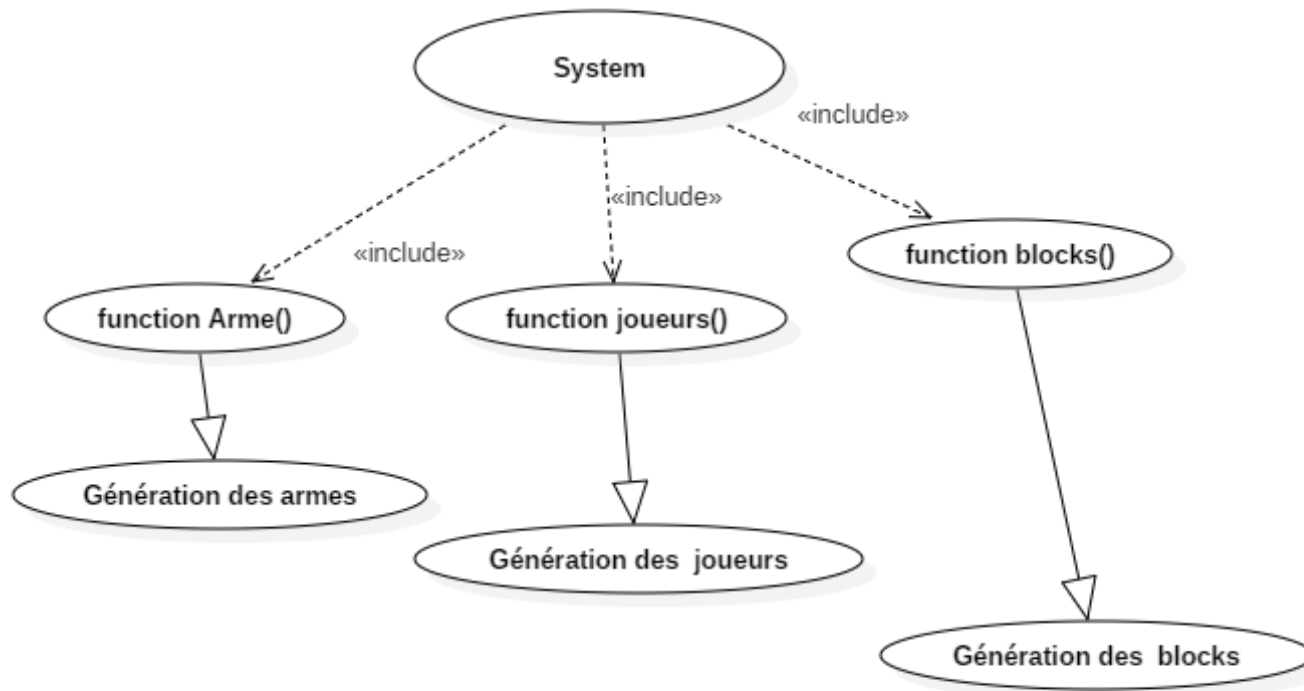


Diagramme du cas d'utilisation génération des objets

Diagramme du cas d'utilisation gestion des mouvements

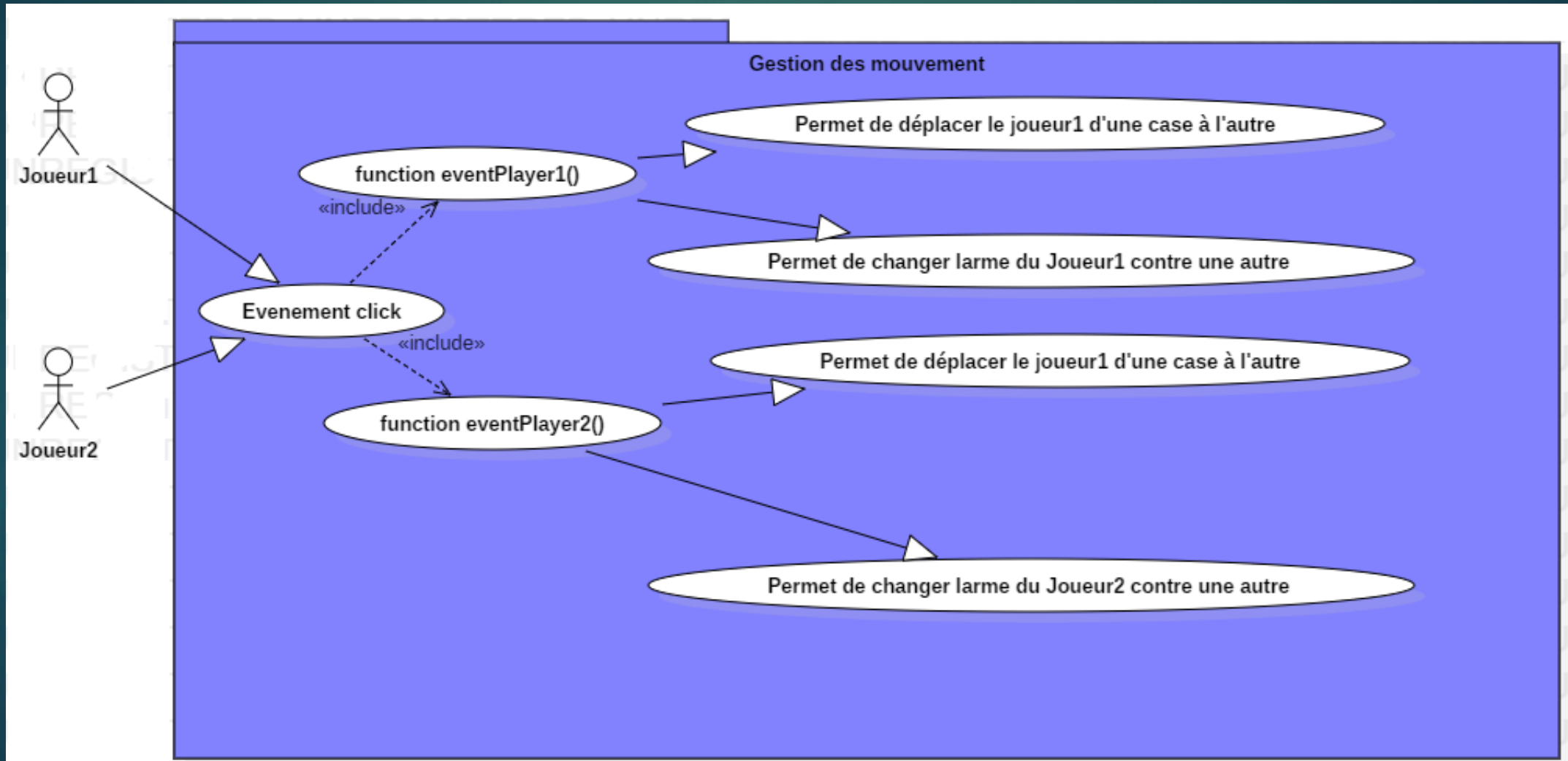
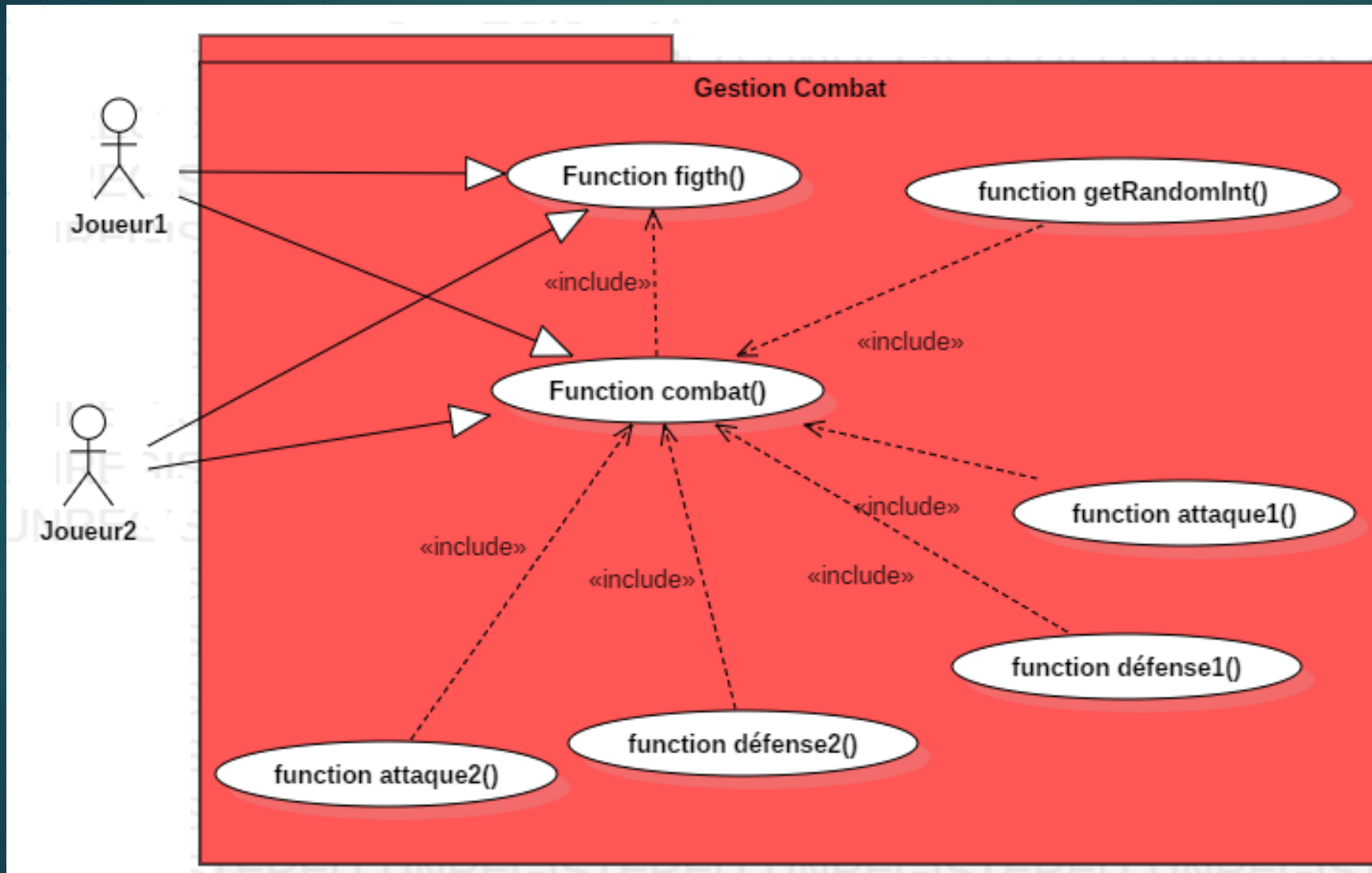


Diagramme du cas d'utilisation combat



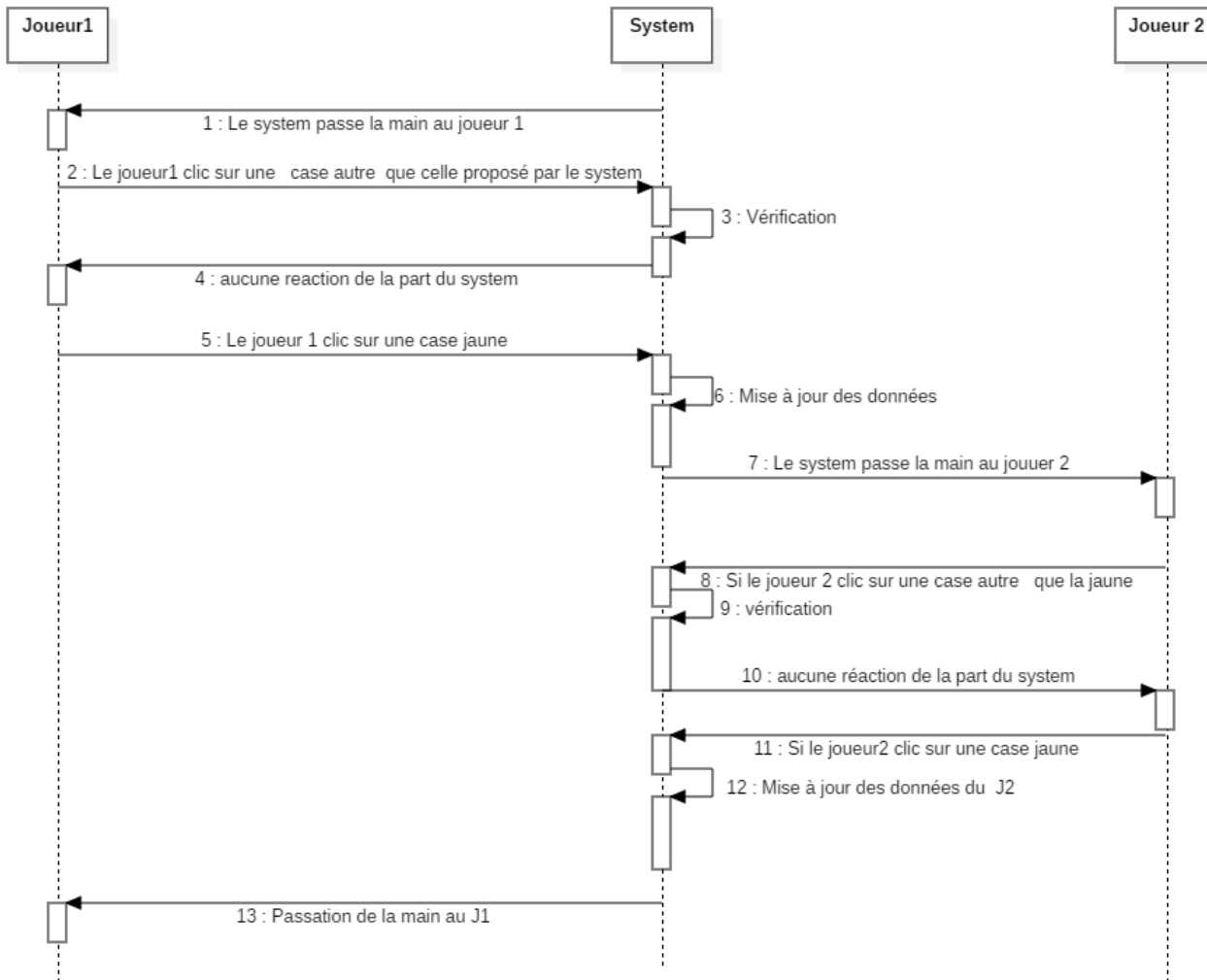


Diagramme de séquence Mouvement

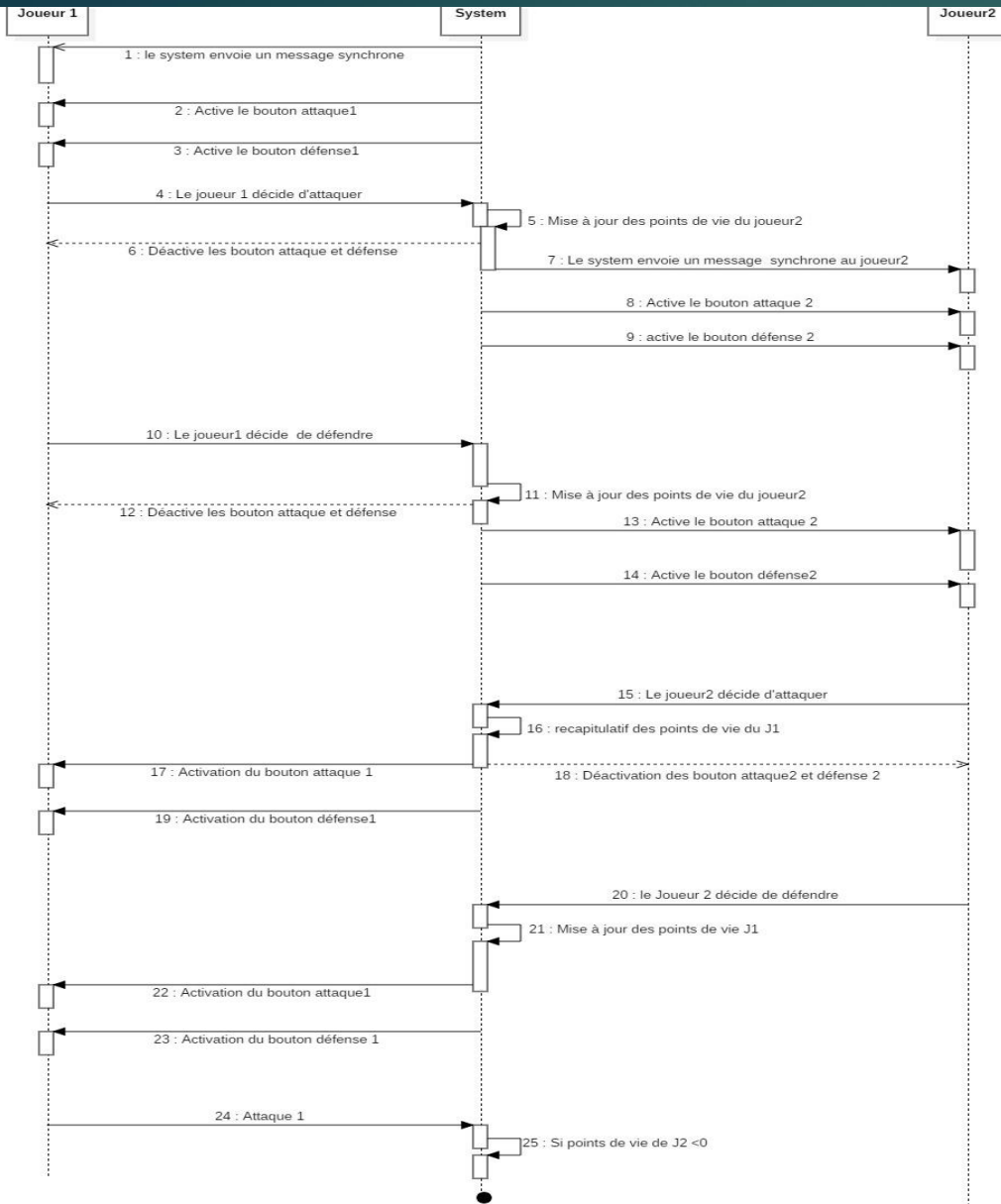


Diagramme de séquence combat



Présentation du code source et démonstration

Fin