Projet Logiciel Transversal

**Projet worms**

Grégoire de Faup – Antoine Delavoypierre



Figure 1 - Worms Armageddon

© Steam

Table des matières

[1 Objectif 3](#__RefHeading__1316_824097905)

[1.1 Présentation générale 3](#__RefHeading__1318_824097905)

[1.2 Règles du jeu 3](#__RefHeading__1320_824097905)

[1.3 Conception Logiciel 3](#__RefHeading__3406_1053578290)

[2 Description et conception des états 4](#__RefHeading__1322_824097905)

[2.1 Description des états 4](#__RefHeading__1480_1060408740)

[2.2 Conception logiciel 4](#__RefHeading__1484_1060408740)

[2.3 Conception logiciel : extension pour le rendu 4](#__RefHeading__1027_1053578290)

[2.4 Conception logiciel : extension pour le moteur de jeu 4](#__RefHeading__3408_1053578290)

[2.5 Ressources 4](#__RefHeading__2098_1060408740)

[3 Rendu : Stratégie et Conception 6](#__RefHeading__2100_1060408740)

[3.1 Stratégie de rendu d'un état 6](#__RefHeading__2104_1060408740)

[3.2 Conception logiciel 6](#__RefHeading__2118_1060408740)

[3.3 Conception logiciel : extension pour les animations 6](#__RefHeading__3410_1053578290)

[3.4 Ressources 6](#__RefHeading__2132_1060408740)

[3.5 Exemple de rendu 6](#__RefHeading__2136_1060408740)

[4 Règles de changement d'états et moteur de jeu 8](#__RefHeading__1336_824097905)

[4.1 Horloge globale 8](#__RefHeading__2079_824097905)

[4.2 Changements extérieurs 8](#__RefHeading__2081_824097905)

[4.3 Changements autonomes 8](#__RefHeading__2083_824097905)

[4.4 Conception logiciel 8](#__RefHeading__1029_1053578290)

[4.5 Conception logiciel : extension pour l'IA 8](#__RefHeading__3412_1053578290)

[4.6 Conception logiciel : extension pour la parallélisation 8](#__RefHeading__1829_2123130103)

[5 Intelligence Artificielle 10](#__RefHeading__1031_1053578290)

[5.1 Stratégies 10](#__RefHeading__2973_1544052078)

[5.1.1 Intelligence minimale 10](#__RefHeading__2975_1544052078)

[5.1.2 Intelligence basée sur des heuristiques 10](#__RefHeading__2977_1544052078)

[5.1.3 Intelligence basée sur les arbres de recherche 10](#__RefHeading__2480_1544052078)

[5.2 Conception logiciel 10](#__RefHeading__2979_1544052078)

[5.3 Conception logiciel : extension pour l'IA composée 10](#__RefHeading__2981_1544052078)

[5.4 Conception logiciel : extension pour IA avancée 10](#__RefHeading__2983_1544052078)

[5.5 Conception logiciel : extension pour la parallélisation 10](#__RefHeading__2985_1544052078)

[6 Modularisation 11](#__RefHeading__2987_1544052078)

[6.1 Organisation des modules 11](#__RefHeading__1831_2123130103)

[6.1.1 Répartition sur différents threads 11](#__RefHeading__1833_2123130103)

[6.1.2 Répartition sur différentes machines 11](#__RefHeading__1835_2123130103)

[6.2 Conception logiciel 11](#__RefHeading__1837_2123130103)

[6.3 Conception logiciel : extension réseau 11](#__RefHeading__1966_2123130103)

[6.4 Conception logiciel : client Android 11](#__RefHeading__1968_2123130103)

# Présentation générale

## Archétype

L’objectif que nous nous sommes fixé est de créer un jeu de type worms 2D mais bien sûr avec des fonctionnalités qui nous seront propres. C'est-à-dire :

* Génération d’une nouvelle carte à chaque partie
* Possibilité de détruire les éléments se trouvant sur notre carte
* Possibilité de jouer avec des amis ou contre un IA
* Création d’une équipe comprenant 1 à 5 personnages (peut varier selon le nombre de joueurs)
* Les positions de départ sont choisis par les joueurs sans qu’ils puissent savoir où se placent les autres
* Chaque personnage possède ses propres statistiques et peut être équipé avec une arme et un atout

Si cela est possible nous aimerions :

* Sauvegarder des données à chaque partie qui permettraient de faire progresser les personnages (choix de plus d’armes/atouts, meilleures statistiques etc)

Un personnage aura les statistiques suivantes :

* Vie
* Points de déplacements
* Nombre d’attaques/tour

Les atouts permettent aux personnages de booster ces caractéristiques : plus de vie ou résistance aux coups, plus de déplacements et plus d’attaques/tour.

## Règles du jeu

Le joueur commence sa partie après avoir composé une équipe. Une carte de jeu est alors créée. Une fois la nouvelle carte affichée le joueur pourra placer un à un tous ses personnages en partant du haut de la carte et ce en un temps donné. Un joueur aura par exemple 15s pour déplacer ses personnages depuis le haut et les placer là où il le souhaite. Passé ce délai les personnages tomberont en chute libre.

Le jeu peut alors vraiment commencer. Puisqu’il s’agît d’un jeu tour par tour les joueurs joue l’un après l’autre et non simultanément. L’ordre de jeu est choisi aléatoirement au début de la partie.

* Pendant son tour le joueur sélectionne **un seul** de ses personnages
  + Son personnage peut alors se déplacer, utiliser son arme et son atout
* Une fois ses points de mouvements et d’attaques le tour de ce joueur est automatiquement terminé (toutefois un joueur est libre de terminer son tour avant).
* Lorsqu’un personnage a perdu toute sa vie à cause des attaques des autres joueurs (ou de chutes) alors il disparaît.
* Le joueur possédant le dernier personnage encore en jeu gagne la partie.

## Ressources

# Description et conception des états

L'objectif de cette section est une description très fine des états dans le projet. Plusieurs niveaux de descriptions sont attendus. Le premier doit être général, afin que le lecteur puisse comprendre les éléments et principes en jeux. Le niveau suivant est celui de la conception logiciel. Pour ce faire, on présente à la fois un diagramme des classes, ainsi qu'un commentaire détaillé de ce diagramme. Indiquer l'utilisation de patron de conception sera très appriécé. Notez bien que les règles de changement d'état ne sont pas attendues dans cette section, même s'il n'est pas interdit d'illustrer de temps à autre des états par leur possibles changements.

## Description des états

## Conception logiciel

## Conception logiciel : extension pour le rendu

## Conception logiciel : extension pour le moteur de jeu

## Ressources

Illustration 1: Diagramme des classes d'état

# Rendu : Stratégie et Conception

Présentez ici la stratégie générale que vous comptez suivre pour rendre un état. Cela doit tenir compte des problématiques de synchronisation entre les changements d'états et la vitesse d'affichage à l'écran. Puis, lorsque vous serez rendu à la partie client/serveur, expliquez comment vous aller gérer les problèmes liés à la latence. Après cette description, présentez la conception logicielle. Pour celle-ci, il est fortement recommandé de former une première partie indépendante de toute librairie graphique, puis de présenter d'autres parties qui l'implémente pour une librairie particulière. Enfin, toutes les classes de la première partie doivent avoir pour unique dépendance les classes d'état de la section précédente.

## Stratégie de rendu d'un état

## Conception logiciel

## Conception logiciel : extension pour les animations

## Ressources

## Exemple de rendu

Illustration 2: Diagramme de classes pour le rendu

# Règles de changement d'états et moteur de jeu

Dans cette section, il faut présenter les événements qui peuvent faire passer d'un état à un autre. Il faut également décrire les aspects lié au temps, comme la chronologie des événements et les aspects de synchronisation. Une fois ceci présenté, on propose une conception logiciel pour pouvoir mettre en œuvre ces règles, autrement dit le moteur de jeu.

## Horloge globale

## Changements extérieurs

## Changements autonomes

## Conception logiciel

## Conception logiciel : extension pour l'IA

## Conception logiciel : extension pour la parallélisation

Illustration 3: Diagrammes des classes pour le moteur de jeu

# Intelligence Artificielle

Cette section est dédiée aux stratégies et outils développés pour créer un joueur artificiel. Ce robot doit utiliser les mêmes commandes qu'un joueur humain, ie utiliser les mêmes actions/ordres que ceux produit par le clavier ou la souris. Le robot ne doit pas avoir accès à plus information qu'un joueur humain. Comme pour les autres sections, commencez par présenter la stratégie, puis la conception logicielle.

## Stratégies

### Intelligence minimale

### Intelligence basée sur des heuristiques

### Intelligence basée sur les arbres de recherche

## Conception logiciel

## Conception logiciel : extension pour l'IA composée

## Conception logiciel : extension pour IA avancée

## Conception logiciel : extension pour la parallélisation

# Modularisation

Cette section se concentre sur la répartition des différents modules du jeu dans différents processus. Deux niveaux doivent être considérés. Le premier est la répartition des modules sur différents threads. Notons bien que ce qui est attendu est un parallélisation maximale des traitements: il faut bien démontrer que l'intersection des processus communs ou bloquant est minimale. Le deuxième niveau est la répartition des modules sur différentes machines, via une interface réseau. Dans tous les cas, motivez vos choix, et indiquez également les latences qui en résulte.

## Organisation des modules

### Répartition sur différents threads

### Répartition sur différentes machines

## Conception logiciel

## Conception logiciel : extension réseau

## Conception logiciel : client Android

Illustration 4: Diagramme de classes pour la modularisation