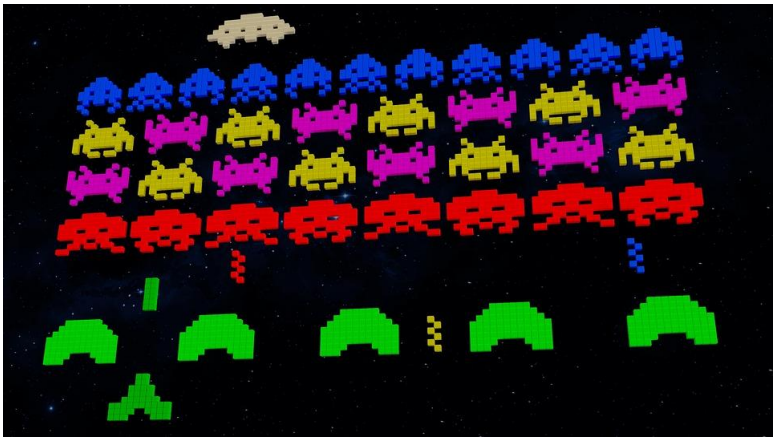


Mini-projet M2103/M2104

CONCEPTION ET DEVELOPPEMENT D'UN PETIT JEU VIDEO

M2103 – M2104 | 2020-2021

Contexte

Le mini-projet se déroule durant les TP 8 et 9 du module M2104 et 13-14-15 du module M2103. Le but est de concevoir puis réaliser un petit jeu vidéo en Java, en utilisant le *framework* JavaGame fourni, en équipe de 3-4 étudiants.

La conception sera réalisée à l'aide du logiciel Visual Paradigm.

Le développement sera réalisé à l'aide de l'EDI Netbeans et du *framework* JavaGame. **Attention : tous les membres de l'équipe doivent utiliser la même version de Netbeans et du JDK.**

Le déroulement sera effectué en utilisant des méthodes « agiles » : respectez bien le déroulement imposé par le sujet et les étapes intermédiaires. Ne passez pas à l'étape N+1 sans avoir fait valider l'étape N à votre enseignant !

A la fin de chaque étape il faudra faire valider votre progression à l'enseignant. Le nombre d'étapes correspondant au nombre de séances, vous finirez normalement une étape vers la fin d'une séance pensez bien à ne pas attendre les 5 dernières minutes de la séance, l'enseignant ne peux pas être partout à la fois ! Si vous êtes en avance, vous pouvez commencer le travail de la séance suivante. Si vous êtes en retard… accélérez !

Le rendu final de l'application doit être :

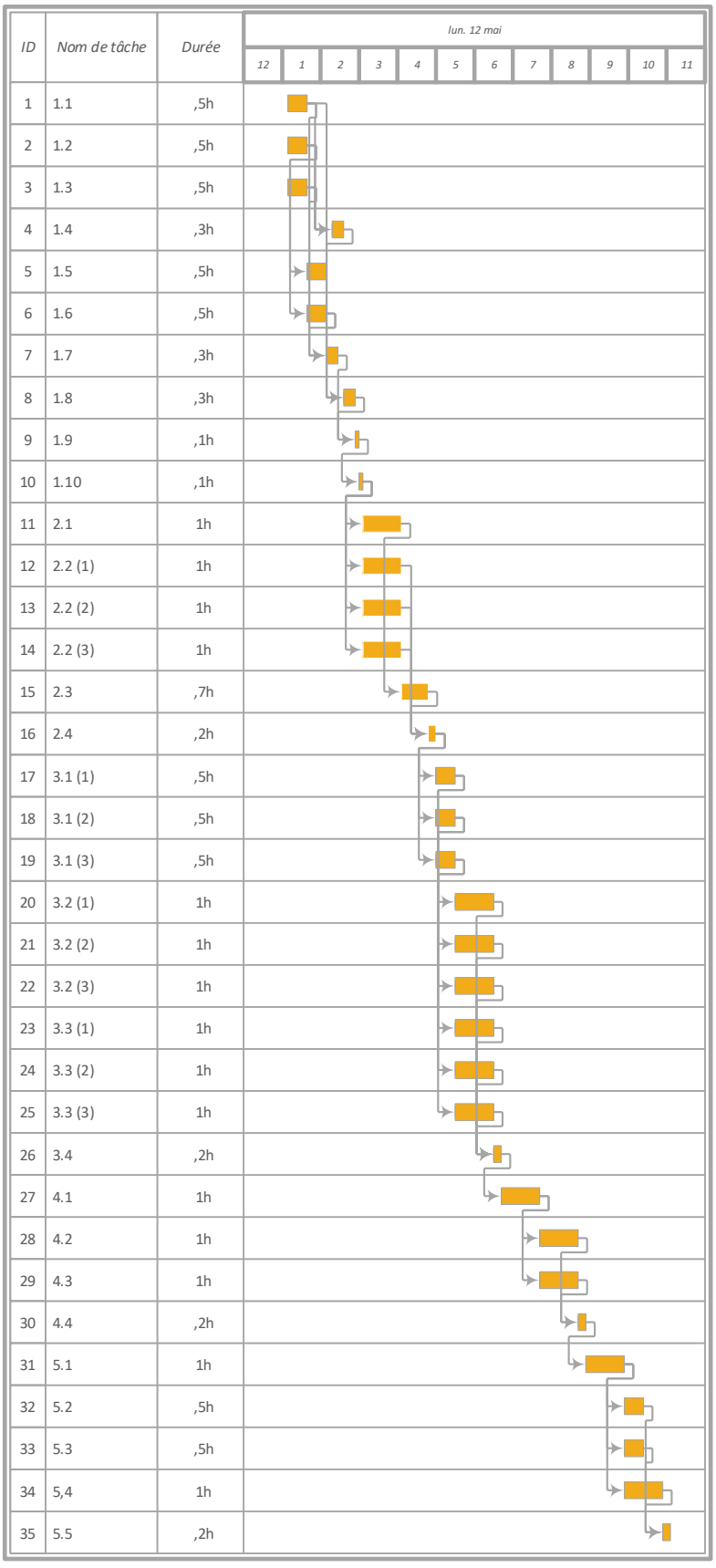
- Un **rapport de conception**, au format PDF, présentant tous les diagrammes du jeu (§ les TP M2104 pour le rapport Visual Paradigm) en version finale (dernière révision validée)
- **L'URL de votre dépôt SVN contenant les codes sources** (exemple https://wasat.u-bourgogne.fr/svn/pacman2019) et le numéro de la révision à considérer si ce n'est pas la dernière
- Une **archive au format 7z** contenant une version **déployable** de votre jeu (dossier à copier, exécutable autoinstallant, fichier MSI…) et un petit document au format PDF indiquant comment jouer (touches du clavier utilisées, règles du jeu, etc.)

Table des matières

Contexte 1
Diagramme de Gantt des différentes tâches 3
Etape 1 : la préparation 4
Tâche n°1 : charte graphique et sonore 4
Tâche n°2 : base de la conception 4
Tâche n°3 : base du développement 5
Tâche n°4 : intégration des ressources 5
Tâche n°5 : création de la classe du jeu (conception) 5
Tâche n°6 : création de la classe du joueur (conception) 5
Tâche n°7 : création de la classe de jeu (code) 6
Tâche n°8 : création de la classe du joueur (code) 6
Tâche n°9 : intégration du joueur 6
Tâche n°10 : intégration 7
Etape 2 : Première version du jeu 8
Tâche n°1 : le déplacement du joueur 8
Tâche n°2 : les autres objets 8
Tâche n°3 : les actions du joueur 9
Tâche n°4 : intégration 9
Etape 3 : les mouvements des items 10
Tâche n°1 : conception 10
Tâche n°2 : développement 10
Tache n°3 : test unitaire 10
Tâche n°4 : Intégration 11
Etape 4 : la gestion des collisions 12
Tâche n°1 : Conception 12
Tâche n°2 : codage 12
Tache n°3 : test unitaire 13
Tâche n°4 : Intégration 13
Etape 5 : création dynamique d'items 14
Tâche n°1 : conception 14
Tâche n°2 : codage évolution 14
Tâche n°3 : codage collision 14
Tâche n°4 : tests unitaires. 15
Tâche n°5 : intégration 15
Etapes suivantes 16

Diagramme de Gantt des différentes tâches

Le diagramme suivant indique l'ensemble des tâches du projet, avec une numérotation E.T avec E=étape et T=tâche. La tâche notée 4.3 est donc la tâche n°3 de l'étape 4. Les contraintes de précédence sont indiquées. Les tâches en parallèle peuvent être confiées à plusieurs étudiants en même temps.



ID	Nom de tâche	Durée
1	1.1	,5h
2	1.2	,5h
3	1.3	,5h
4	1.4	,3h
5	1.5	,5h
6	1.6	,5h
7	1.7	,3h
8	1.8	,3h
9	1.9	,1h
10	1.10	,1h
11	2.1	1h
12	2.2 (1)	1h
13	2.2 (2)	1h
14	2.2 (3)	1h
15	2.3	,7h
16	2.4	,2h
17	3.1 (1)	,5h
18	3.1 (2)	,5h
19	3.1 (3)	,5h
20	3.2 (1)	1h
21	3.2 (2)	1h
22	3.2 (3)	1h
23	3.3 (1)	1h
24	3.3 (2)	1h
25	3.3 (3)	1h
26	3.4	,2h
27	4.1	1h
28	4.2	1h
29	4.3	1h
30	4.4	,2h
31	5.1	1h
32	5.2	,5h
33	5.3	,5h
34	5,4	1h
35	5.5	,2h

Etape 1 : la préparation

Objectif de l'étape : avoir tout mis en place pour le développement du jeu. Pouvoir exécuter le jeu (sans interaction ni affichage, mais sans erreur).

Vous avez normalement déjà obtenu un dépôt SVN pour votre équipe (§ TP n°7 M2104).

Choisissez un jeu parmi la liste suivante. Vous pouvez discuter avec votre enseignant le choix du jeu.

Les jeux possibles sont les suivants (avec des variantes possibles) :

- Casse-briques
- Pac-man
- Space invaders
- Asteroids
- Rat Splat

Pour un autre jeu, discutez avec votre enseignant pour qu'il puisse juger de sa difficulté voire, même, de sa réalisabilité. N'oubliez pas que dans une équipe, il faut travailler en parallèle. Répartissez-vous donc bien les différentes tâches en faisant attention aux contraintes de précédence !

TACHE 1.1 : CHARTE GRAPHIQUE ET SONORE

Déterminer les différents graphismes nécessaires dans votre jeu : *sprites* des personnages, décor d'arrière-plan, etc. Réunir les images nécessaires, les retravailler au besoin pour qu'elles soient au format PNG avec transparence (32 bits) de la taille souhaitée. Vous pouvez aussi récupérer des sons (au format .wav non compressé) pour le jeu.

TACHE 1.2 : BASE DE LA CONCEPTION

Il faut créer un fichier *Visual Paradigm* pour l'équipe. Vous ne pouvez pas travailler à plusieurs sur le même fichier ! Il est par contre possible de travailler dessus « en série » : coordonnez-vous (par Teams ou tout autre moyen de dialogue en direct) pour éviter que plusieurs personnes n'ouvrent le même fichier à la fois. Placez ce fichier soit dans votre dépôt SVN doit dans un « drive » partage (Onedrive, GDrive, ou même avec une équipe dans Teams) afin que toute l'équipe puisse y accéder.

Chaque membre de l'équipe peut se connecter sur le projet et travailler sur l'analyse/conception. Le projet doit absolument être documenté correctement : les classes, opérations, associations, cas d'utilisations, messages doivent être documentés (exceptions : les membres privés, les membres redéfinis ne sont pas obligatoirement documentés).

Importez dans votre projet le diagramme de classes du *framework* « Javagame » fourni dans `Javagame.vpp`.

Créez un **diagramme de classes** portant le nom de votre jeu (ex : `tetris`). Glissez-déposez dans ce diagramme les classes suivantes issues de `Javagame.vpp` : `Game`, `GameItem`, `BoxGameItem` ainsi que les interfaces nécessaires pour jouer au clavier (`KeyListener`) et à la souris (`MouseListener`, `MouseMotionListener`).

Créez un **diagramme de cas d'utilisation** comportant un acteur (le joueur) et un système (donnez-lui le nom de votre jeu). Laissez-le vide pour l'instant.

TACHE 1.3 : BASE DU DEVELOPPEMENT

Normalement le dépôt SVN (§ TP4) pour votre projet a déjà été créé (si vous n'avez pas oublié d'envoyer le mail prévu à la fin du TP7…). Sur votre bureau (par exemple), reliez-vous à ce dépôt (*SVN checkout*).

Créez un projet *Netbeans* dans ce dépôt (attention à bien utiliser la même version de Netbeans dans tout le groupe). Recopiez le fichier `JavaGame.jar` fourni sur le commun. Référencez-le comme bibliothèque de votre projet Netbeans. Faites bien *enregistrer tout* sous Netbeans.

Dans l'explorateur windows, ajoutez (clic droit, *TortoiseSVN→Add*) tout le contenu du projet Netbeans (sauf le dossier `private` qui ne doit pas être dans le dépôt).

Publiez votre projet. Chaque membre de l'équipe peut à présent travailler sur le projet.

TACHE 1.4 : INTEGRATION DES RESSOURCES

Cette tâche nécessite que les tâches 1 et 3 soient terminées.

Reliez-vous au dépôt SVN du projet. Ajoutez dans ce dépôt les images dans un dossier `images` et les sons dans un dossier `sons`. Pensez bien à les intégrer au dépôt (commande *TortoiseSVN→Add*) et non simplement les copier dans le dossier. Ouvrez le projet Netbeans et ajoutez les deux dossiers de ressources comme **bibliothèques**. Enregistrez tout.

Publiez le projet.

TACHE 1.5 : CREATION DE LA CLASSE DU JEU (CONCEPTION)

Cette tâche nécessite que la tâche 2 soit terminée.

Ouvrez le fichier *Visual Paradigm* (voir Tâche 1.2 : base de la conception) et éditez le diagramme de classes portant le nom de votre jeu. Créez une classe héritant de `iut.Game` et représentant votre jeu. Faites implémenter les opérations abstraites (clic-droit sur la classe, *Related Eléments→réaliser toutes les interfaces*).

Mettez à jour et publiez le projet.

TACHE 1.6 : CREATION DE LA CLASSE DU JOUEUR (CONCEPTION)

Cette tâche nécessite que la tâche 2 soit terminée.

Ouvrez le fichier *Visual Paradigm* (voir Tâche 1.2 : base de la conception) et éditez le diagramme de classes portant le nom de votre jeu.

Créez une classe héritant de `iut.GameItem` ou `iut.BoxGameItem` et représentant votre joueur. Si le jeu se fait au clavier, implémentez `KeyListener`. S'il se fait à la souris, implémentez `MouseListener` et `MouseMotionListener`. Faites implémenter les opérations abstraites (clic-droit sur la classe, *Related Elements*, réaliser toutes les interfaces).

Mettez à jour et publiez le projet.

TACHE 1.7 : CREATION DE LA CLASSE DE JEU (CODE)

Cette tâche nécessite que les tâche n°1, n°3 et n°6 soient terminées.

Ouvrez le fichier *Visual Paradigm* (voir Tâche 1.2 : base de la conception) et regardez la classe représentant le jeu telle que conçue dans la tâche n°6 (respectez notamment son **nom**).

Connectez-vous au dépôt SVN du jeu (voir Tâche 1.3 : base du développement). Créez une nouvelle classe correspondante (elle doit hériter de `iut.Game`). Netbeans vous propose de corriger les erreurs en implémentant les opérations abstraites et le constructeur, faites-le. N'oubliez pas d'ajouter le nouveau fichier Java au dépôt (commande *TortoiseSVN→Add*).

Modifiez la fonction qui dessine l'arrière-plan pour qu'elle dessine un **fond noir** sur toute la fenêtre. Supprimez les « `throw new NotImplementedException` » des autres opérations pour pouvoir exécuter le projet.

Dans la classe principale (dans l'opération `main`) créez une instance de cette classe et appelez sa fonction `play`.

Testez : votre projet doit s'exécuter en présentant un écran noir.

Enregistrez et **publiez** votre projet.

TACHE 1.8 : CREATION DE LA CLASSE DU JOUEUR (CODE)

Cette tâche nécessite que les tâches n°1, n°4 et n°6 soient terminées.

Ouvrez le fichier Visual Paradigm du serveur et observez le diagramme de classes : une classe doit correspondre au joueur (elle a été conçue à la tâche 6).

Reliez-vous au projet SVN du jeu (sous Netbeans). Créez une nouvelle classe correspondant à celle vue dans la conception (respectez bien son nom, ses ancêtres et interfaces). Netbeans vous propose de corriger les erreurs en implémentant les opérations abstraites : acceptez. Il vous propose également de créer un constructeur : acceptez.
Modifiez le constructeur pour utiliser le fichier .png fourni par la tâche 1 comme *sprite* du joueur. Supprimez toutes les lignes « `throw new NotimplementedException` » du fichier.

Ajoutez le nouveau fichier `.java` ainsi créé au dépôt SVN.

TACHE 1.9 : INTEGRATION DU JOUEUR

Cette tâche nécessite que les tâches 7 et 8 soient terminées.

Ouvrez le projet Netbeans (n'oubliez pas les mises à jour SVN) et éditez la classe correspondant à votre jeu (celle qui hérite de `iut.Game`). Dans l'opération `createObjects`, faites ajouter au jeu une instance de la classe représentant le joueur (créée à la tâche 8). Réfléchissez bien à sa position initiale.

TACHE 1.10 : INTEGRATION

Cette tâche nécessite que toutes les tâches précédentes soient finalisées. Elle doit se faire sur le poste de votre enseignant pour validation. Demandez-lui, par Teams, en lui rappelant l'URL de votre dépôt.

Le jeu doit compiler et se lancer, avec un écran noir et l'image du joueur à sa position initiale. Corrigez au besoin ! En cas de manque de fichiers, il faut que l'étudiant qui a oublié de l'inclure dans le dépôt le fasse et refasse un *commit*.

En cas de modifications, n'oubliez pas de **publier** votre projet (*TortoiseSVN→Commit*).

Etape 2 : Première version du jeu

Objectif de la séance : avoir une première version très simple du jeu.

Les différentes tâches sont à adapter (voir avec votre enseignant) en fonction du jeu choisi.

TACHE 2.1 : LE DEPLACEMENT DU JOUEUR

Ouvrez la conception avec Visual Paradigm et le projet (présent sur le serveur SVN) avec Netbeans. N'oubliez pas les *update* au besoin.

Editez le **diagramme de cas d'utilisation** et ajoutez le(s) cas nécessaire(s) pour le déplacement du joueur (cela peut être un seul cas ou plusieurs, suivant le jeu). Réalisez le **diagramme de séquence** de ce cas (vous pouvez être amené à ajouter de ce fait des opérations, attributs et propriétés dans la classe correspondant au joueur).

Faites **valider** ce diagramme à l'enseignant.

Codez ensuite sous Netbeans la/les opération(s) nécessaire(s) au **déplacement** (suivant le type d'interaction choisie, clavier ou souris) en suivant les consignes de votre diagramme de séquence. Vous pouvez ajouter des membres à votre classe (attributs, opérations) en restant en cohérence avec votre diagramme de conception.

Exécutez le jeu : vous pouvez déplacer le joueur. Vérifiez que les règles de déplacement soient bien respectées (limites, orientation, changement d'image, etc...).

Enregistrez et **publiez** la conception et le développement.

TACHE 2.2 : LES AUTRES OBJETS

Cette tâche peut être décomposée en plusieurs autres tâches suivant le jeu choisi et le nombre d'items présents dans le jeu (nous parlons pour l'instant des items présents dès le début du jeu).

Exemples d'items : les vaisseaux ennemis, le tir du joueur et le tir des vaisseaux ennemis dans *Space Invaders*, les fantômes, les murs et les pilules (*Pac Man*), les astéroides (*Astéroids*), les souris (*Rat Splat*), les briques et la balle (*Casse-briques*) ...

Ouvrez la conception avec Visual Paradigm et le projet (présent sur le serveur SVN) avec Netbeans. N'oubliez pas les *update* au besoin.

Pour chaque *item*, créez (dans la conception) une classe héritant de `iut.GameItem` (ou `iut.BoxGameItem`) et faites implémenter les opérations abstraites (clic droit, *related éléments, realize all interfaces*).

Ajoutez dans le diagramme de cas d'utilisation le(s) cas rendus nécessaires (il peut ne pas en avoir, tout dépend du jeu).

Dans le projet Netbeans, créez une classe correspondant à la classe crée dans la conception. Faites implémenter les opérations abstraites et le constructeur. Supprimez les `throw new …` ajoutés automatiquement. N'oubliez pas d'ajouter ce nouveau fichier au dépôt SVN.

Modifiez le constructeur pour utiliser comme nom de *sprite* le nom du fichier .png utilisé pour l'item.

Modifiez la fonction `createObjects` de la classe du jeu pour intégrer le nouvel item au jeu. Exécutez : l'item doit apparaitre sur un fond noir (mais il ne se déplace pas).

Publiez les projets (conception et développement).

TACHE 2.3 : LES ACTIONS DU JOUEUR

Suivant le jeu, le joueur peut avoir des actions (différentes du déplacement) : tirer par exemple (space invaders, asteroids). Cette tâche nécessite que la tâche 1 soit terminée.

Ouvrez la conception (présente sur le serveur VP) avec Visual Paradigm et le projet (présent sur le serveur SVN) avec Netbeans. N'oubliez pas les *update* au besoin.

Modifiez le **diagramme de cas d'utilisation** pour ajouter les actions du joueur. Pour chaque cas, décrire l'action dans un diagramme de séquence (exemple : si le joueur tire, alors un item doit être créé et ajouté au jeu).

Modifiez dans Netbeans les fonctions de la classe représentant le joueur (suivant le type d'interaction, clavier ou souris) et implémentez la séquence décrite. Vous pouvez être amené à ajouter des membres à votre classe, faites le bien en cohérence entre le diagramme et le code.

Testez en exécutant le jeu (il n'y a pour l'instant aucun déplacement).

Publiez l'ensemble.

TACHE 2.4 : INTEGRATION

Cette tâche nécessite que toutes les tâches précédentes soient finalisées. Elle doit se faire sur le poste de votre enseignant, avec lui, pour validation.

Ouvrez le dépôt SVN dans Netbeans, exécutez le jeu. Il doit compiler et se lancer, avec le joueur déplaçable et les autres items immobiles. Les tests unitaires doivent tous être valides.

Corrigez au besoin, et en cas de modifications, n'oubliez pas de **republier** votre projet (*TortoiseSVN→Commit*).

Etape 3 : les mouvements des items

Objectif de l'étape : avoir les mouvements corrects des différents items du jeu (ennemis, tirs, etc.)

Ouvrez la conception (présente sur le serveur VP) avec Visual Paradigm et le projet (présent sur le serveur SVN) avec Netbeans. N'oubliez pas les *update* au besoin.

TACHE 3.1 : CONCEPTION

Cette tâche est nécessaire pour chaque *item* du jeu (elle peut donc être multipliée par 2, 3, 4…)

Ouvrez la conception (présente sur le serveur VP) avec Visual Paradigm. N'oubliez pas les *update* au besoin.

Ouvrez le diagramme de classes. Sélectionnez l'opération `evolve` de votre item et liez-là à un nouveau diagramme de séquence. Réfléchissez bien au **déplacement** de votre item (ne vous occupez pas pour l'instant des collisions), limites, etc. Complétez le diagramme de séquence pour implémenter l'évolution de l'objet. Si des messages à d'autres objets sont nécessaires, il sera peut-être pertinent d'ajouter ceux-ci dans le diagramme des classes. De même, si des attributs sont utiles pour réaliser l'opération.

Vous pouvez également essayer de faire **l'algorithme** dans un diagramme d'activité ou dans un diagramme d'états (non décrits en cours, mais vous pouvez demander à l'enseignant…).

Faites valider votre/vos diagrammes à l'enseignant. **Publiez** la conception.

TACHE 3.2 : DEVELOPPEMENT

Cette tâche nécessite que la tâche 1 soit terminée.

Cette tâche se répète pour chaque item du jeu (indépendamment des autres). Il n'est pas nécessaire que l'étudiant qui code une fonction soit le même que celui qui ait conçu la fonction.

Ouvrez la conception (présente sur le serveur VP) avec Visual Paradigm et le projet (présent sur le serveur SVN) avec Netbeans. N'oubliez pas les *update* au besoin.

En vous aidant que la conception réalisée en tâche 1, codez la fonction `evolve` de votre item. N'oubliez pas d'ajouter les éventuels attributs ou opérations rajoutés dans le diagramme de classe par la tâche 2. Si vous avez besoin d'ajouter de nouveaux attributs et/ou opérations, modifiez également le diagramme de classes.

Testez en exécutant si le mouvement parait correct. Publiez la conception et le développement.

TACHE 3.3 : TEST UNITAIRE

Cette tâche nécessite que la tâche 1 soit terminée. Elle n'est pas obligatoire au fonctionnement du jeu mais permet de mieux intégrer les fonctionnalités et est donc fortement recommandée.

Cette tâche se répète pour chaque item du jeu (indépendamment des autres). Il n'est pas obligatoire que l'étudiant qui réalise le test soit le même que celui qui a réalisé la conception. Il n'est pas utile d'attendre que le codage soit fait.

Ouvrez la conception avec Visual Paradigm et le projet (présent sur le serveur SVN) avec Netbeans. N'oubliez pas les *update* au besoin.

Créez dans le projet Netbeans un cas de test pour votre item de jeu. Créez une fonction de test pour la fonction `evolve`. Aidez-vous de la conception (diagramme de séquence, documentation, etc.) pour bien comprendre la fonctionnalité et réaliser un test cohérent.

N'oubliez pas d'ajouter au dépôt (*TortoiseSVN→Add*) les nouveaux dossiers/fichiers nécessaires au test.

TACHE 3.4 : INTEGRATION

Cette tâche nécessite que les tâches 1, 2 et 3 soient terminées et doit se faire en présence de l'enseignant, sur son propre poste.

Ouvrez le projet (présent sur le serveur SVN) avec Netbeans. N'oubliez pas les *update* au besoin.

Exécutez les tests unitaires : ils doivent être valides.

Exécutez l'application : vous devez voir se déplacer les items prévus au départ du jeu ou créés par le joueur (tirs, etc.). Corrigez bien évidemment au besoin et republiez !

Etape 4 : la gestion des collisions

Objectif de l'étape : gérer les collisions des items présents dès le début du jeu et réagir à celles-ci.

Il est rappelé que, dans le *framework* JavaGame, deux opérations abstraites de `GameItem` sont présentes pour la gestion des collisions : `isCollide` qui indique si un item « touche » un autre, et `collideEffect` qui réagit à la collision. `isCollide` est implémentée dans `BoxGameItem` par une simple *hitbox* rectangulaire.

On suppose que `isCollide` est implémentée (c'est le cas si vous héritez de `BoxGameItem` : dans le cas où vous héritez de `GameItem`, il faut bien sûr coder cette opération).

TACHE 4.1 : CONCEPTION

La tâche est répétée pour chaque item du jeu, y compris le joueur.

Ouvrez la conception (présente sur le serveur VP) avec Visual Paradigm. N'oubliez pas les *update* au besoin.

Créez un diagramme de séquence relié à l'opération `collideEffect` de votre item (à faire pour chaque item, joueur compris).

Réfléchissez bien à ce qu'il se passe pour l'item (pas pour l'autre qui est passé en paramètre) en cas de collision. Si vous avez besoin de différencier en fonction de l'item touché (on ne réagit pas de la même manière à la collision sur un bonus que sur un tir ennemi), vous pouvez utiliser l'opération `getType` de la classe `GameItem` (abstraite, donc obligatoirement définie dans chaque item).

Une collision peut déclencher un certain nombre de choses :

- Modification de l'état (valeur des attributs) pour l'item (par exemple, incrémenter un score, décrémenter les points de vie…)
- Création dans le jeu d'un nouvel item (exemple : toucher un bonus qui fait créer une balle, explosion, etc…). Il faut donc bien créer l'objet (allocation par new) et l'ajouter au jeu (`getGame().add`)
- Suppression de l'item du jeu (exemple : être touché par un tir pour un ennemi, toucher un ennemi pour le tir, etc..). Il faut donc dans ce cas penser à le retirer du jeu (`getGame().remove`)
- Modification de l'état d'un item du jeu (exemple : changement de score, jouer u son…) il faut donc dans ce cas envoyer un message à l'item, donc rajouter une opération dans celui-ci et l'associer à l'item

Faites valider vos diagrammes avant de les publier.

TACHE 4.2 : CODAGE

Cette tâche nécessite que la tâche 1 soit terminée.

La tâche est répétée pour chaque item du jeu, y compris le joueur.

Ouvrez la conception avec Visual Paradigm et le projet (présent sur le serveur SVN) avec Netbeans. N'oubliez pas les *update* au besoin.

Implémentez les modifications notamment du diagramme de séquence de la fonction `collideEffect` de l'item (et des éventuelles opérations rajoutées par celui-ci dans d'autres classes).

Il peut être difficile de tester directement dans certains cas (si votre collision dépend d'un item non encore finalisé). Vérifiez en revanche bien que les tests unitaires continuent de fonctionner.

N'oubliez pas de rajouter aux dépôts tout nouveau fichier du projet, et publiez le projet et la conception.

TACHE 4.3 : TEST UNITAIRE

Cette tâche nécessite que la tâche 1 soit terminée. Elle n'est pas obligatoire au fonctionnement du jeu mais permet de mieux intégrer les fonctionnalités et est donc fortement recommandée.

Cette tâche se répète pour chaque item du jeu (indépendamment des autres).

Ouvrez la conception (présente sur le serveur VP) avec Visual Paradigm et le projet (présent sur le serveur SVN) avec Netbeans. N'oubliez pas les *update* au besoin.

Créez dans le projet Netbeans un cas de test pour votre item de jeu. Créez une fonction de test pour la fonction `collideEffect`. Aidez-vous de la conception (diagramme de séquence, documentation, etc.) pour bien comprendre la fonctionnalité et réaliser un test cohérent. Cette opération étant une procédure et non une fonction, elle sera délicate à tester : utilisez pour ce faire les accesseurs disponibles (vous n'aurez pas forcément ce que vous cherchez, mais ne rajoutez aucune opération dans les classes) dans les items ou dans le jeu. Vous pouvez également utiliser des *mocks objects* (objets factices) pour faciliter les tests (exemple : une classe `FakeGame` au lieu de votre jeu, une classe `FakeItem` au lieu d'un item…).

N'oubliez pas d'ajouter au dépôt (*TortoiseSVN→Add*) les nouveaux dossiers/fichiers nécessaires au test avant de publier le projet.

TACHE 4.4 : INTEGRATION

Cette tâche nécessite que les tâches 1, 2 et 3 soient terminées.

Ouvrez le projet (présent sur le serveur SVN) avec Netbeans. N'oubliez pas les *update* au besoin.

Exécutez les tests unitaires : ils doivent être valides.

Exécutez l'application : vous devez voir se déplacer les items prévus au départ du jeu ou créés par le joueur (tirs, etc.). Corrigez bien évidemment au besoin !

Etape 5 : création dynamique d'items

Objectif de l'étape : gestion des items de jeu non présents au début du jeu et non déclenchés par le joueur (tirs des ennemis, nouveaux ennemis, apparition de bonus/malus, etc.).

TACHE 5.1 : CONCEPTION

Déterminez quels sont les items qui doivent apparaître au cours du jeu et quel est l'évènement déclencheur.

Si le déclenchement est lié à un intervalle de temps, la fonction `evolve` d'un item (créé pour l'occasion, comme le `GénérateurBalle` du petit jeu exemple) est appropriée.

La tâche est répétée pour chaque item apparaissant.

Ouvrez la conception (présente sur le serveur VP) avec Visual Paradigm et le projet (présent sur le serveur SVN) avec Netbeans. N'oubliez pas les *update* au besoin.

Ajoutez dans le diagramme de classe le nouvel item, et faites-le hériter de `GameItem`, `BoxGameItem` ou un autre item déjà créé. Implémentez les opérations abstraites (*related éléments → realize all interfaces*).

Définissez pour l'opération `evolve` un diagramme de séquence, ainsi que pour l'opération `collideEffect` (les deux peuvent être faites en parallèle par deux personnes). Ajoutez au besoin des membres ou des classes. N'oubliez pas que votre nouvel item peut avoir besoin d'une nouvelle image et/ou d'un nouveau son.

Ajoutez dans le projet Netbeans la classe correspondante, son constructeur et ses éventuels attributs/opérations.

Faites valider par l'enseignant et publiez les modifications.

TACHE 5.2 : CODAGE EVOLUTION

Cette tâche nécessite que la tâche 1 soit terminée.

La tâche est répétée pour chaque item du jeu rajouté lors de cette étape.

Ouvrez la conception (présente sur le serveur VP) avec Visual Paradigm et le projet (présent sur le serveur SVN) avec Netbeans. N'oubliez pas les *update* au besoin.

Implémentez le code de la fonction `evolve` de votre item (suivre les diagrammes UML).

Testez et publiez les modifications.

TACHE 5.3 : CODAGE COLLISION

Cette tâche nécessite que la tâche 1 soit terminée.

La tâche est répétée pour chaque item du jeu rajouté lors de cette étape.

Ouvrez la conception (présente sur le serveur VP) avec Visual Paradigm et le projet (présent sur le serveur SVN) avec Netbeans. N'oubliez pas les *update* au besoin.

Implémentez le code de la fonction `collideEffect` de votre item (suivre les diagrammes UML).

Testez et publiez les modifications.

TACHE 5.4 : TESTS UNITAIRES.

Cette tâche nécessite que la tâche 1 soit terminée.

La tâche est répétée pour chaque item du jeu rajouté lors de cette étape.

Ouvrez la conception (présente sur le serveur VP) avec Visual Paradigm et le projet (présent sur le serveur SVN) avec Netbeans. N'oubliez pas les *update* au besoin.

Ajoutez une classe de test pour l'item créé en étape 1, avec un test pour sa fonction `evolve` et un test pour sa fonction `collideEffect`. Aidez-vous des spécifications (conception, documentation) pour bien tester ces fonctions.

TACHE 5.5 : INTEGRATION

Cette tâche nécessite que toutes les tâches précédentes de l'étape soient terminées.

Ouvrez le projet (présent sur le serveur SVN) avec Netbeans. N'oubliez pas les *update* au besoin.

Exécutez les tests unitaires : ils doivent être valides.

Exécutez l'application et vérifiez le fonctionnement du jeu. Corrigez bien évidemment au besoin !

Etapes suivantes

Si vous êtes rapides, ou que vous avez envie de poursuivre le jeu, vous pouvez bien évidemment continuer !

Vous avez compris le principe de chaque étape : on prend une nouvelle fonctionnalité, on la conçoit en UML, on la développe en Java, on la teste avec *JUnit* et on vérifie son intégration. Si tout marche bien on passe à la suite !

Quelle serait cette suite ? Voici quelques pistes (liste non exhaustive) que vous pouvez intégrer suivant votre jeu :

- Gérer des niveaux de difficulté
- Avoir un arrière-plan plus sympa qu'un fond noir
- Animer certains items (mouvements, ...)
- Avoir des items temporaires pour animations (explosions, ...)
- Gérer la sauvegarde d'une partie (et donc son chargement)
- Gérer une pause pendant le jeu
- Avoir plusieurs niveaux dans le jeu (plusieurs vagues de space invaders, plusieurs tableaux de pac man, ...)
- Diversifier les items (plusieurs types d'ennemis, de briques, de bonus/malus, etc.)