GRANDE ÉCOLE D'INGÉNIEURS

UE Informatique
Étape 5 - Réalisation du jeu !

Olivier Beaudoux (olivier.beaudoux@eseo.fr)

Semestre 4

Table des matières

1 Définition du jeu

Le type abstrait *Jeu* met en relation les pioches des joueurs et la pile du jeu ainsi que leurs vues avec les utilisateurs de l'application que sont les joueurs. Cette étape 5.1 consiste à implémenter la structure de données relative au type abstrait *Jeu*.

1.1 Vue d'ensemble des types abstraits

La figure 1.1 fait la synthèse de l'ensemble des type abstraits utilisés pour concevoir l'application « Pytatra ! ». On y retrouve les types *Joueur*, *Pioche*, *Pile*, *Exemplaires*, *Empilement* et *Planchette* des étapes précédentes, ainsi que le type *Jeu* de cette dernière étape.

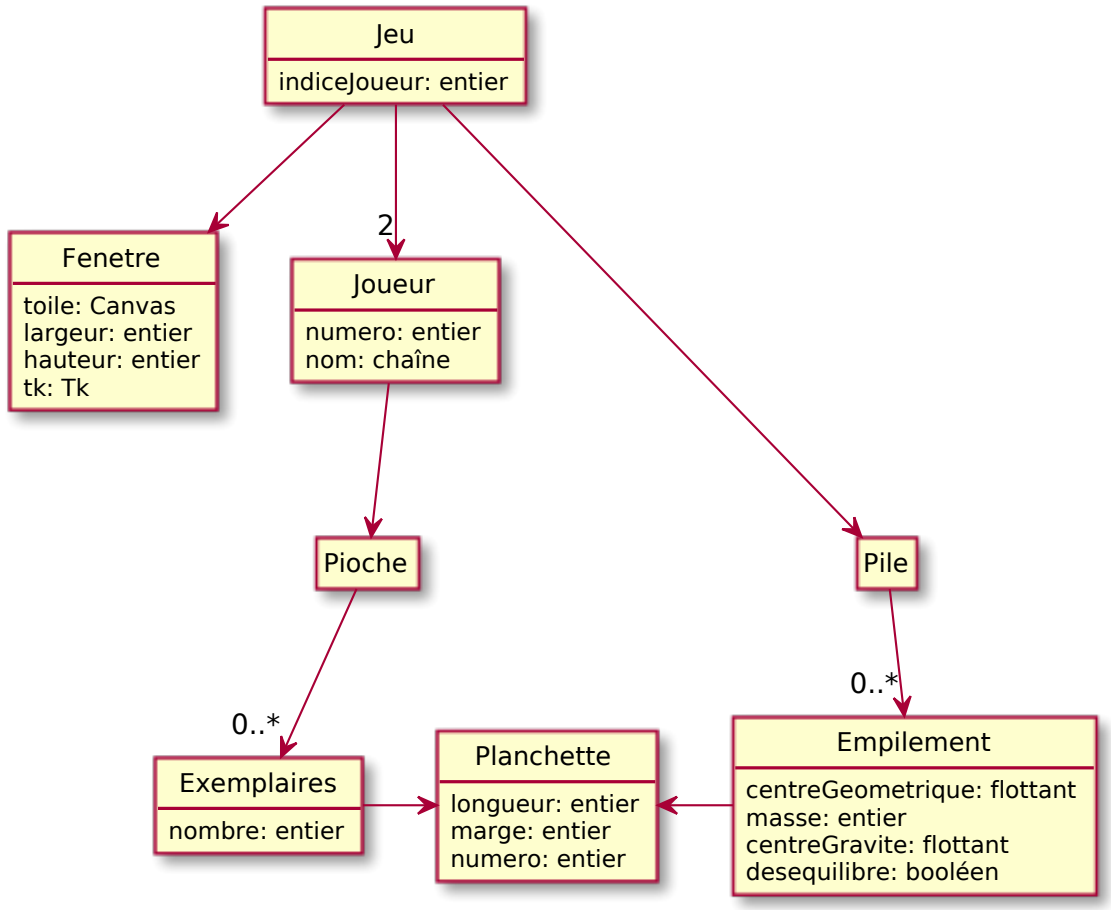

FIGURE 1.1 – Modèle complet du jeu

La figure inclut uniquement les types relatifs aux modèles M au sens MVC. Cette dernière étape traite du jeu et de son IHM. L'IHM fait appel au type abstrait *Fenetre* ainsi qu'aux modules *Dialogue*, *VuePile* et *VuePioche*. Le type abstrait *Jeu* réalise, en tant que contrôleur (C), la liaison entre les modèles (M) et l'IHM (V) ; il implémente ainsi la « logique » du jeu et de ses règles.

1.2 Implémentation de la structure de données du jeu

Spécification partielle

La spécification partielle du type abstrait *Jeu* est la suivante :

***cree()* :** retourne une structure de données représentant le jeu dans son état initial. Cette structure de données doit donc inclure une fenêtre, une pile, deux joueurs et leur pioche. Les vues sont des modules et ne sont donc pas présents dans une telle structure.

***fenetre(jeu)* :** retourne la fenêtre principale de l'application.

***pile(jeu)* :** retour la pile de planchettes du *jeu*.

joueurs(jeu) : retourne la liste des joueurs.

indiceJoueur(jeu) : retourne l'indice du joueur courant (0 ou 1)

***joueurCourant(jeu)* :** retourne le joueur courant du *jeu*.

***passeJoueurSuivant(jeu)* :** passe au joueur suivant.

Travail à faire

Étape 5.1

- Implémenter les fonctions précédentes du type abstrait *Jeu*.
- Tester ce code en exécutant le module *TestJeu*.

2 Le jeu et son affichage

Avant de réaliser à proprement parler le jeu, il est nécessaire de mettre en place l'affichage de ses différents éléments constitutifs : c'est l'objet de cette étape 5.2.

2.1 Fenêtre et vues

Les différentes vues définies des étapes 2, 3 et 4 créent différents éléments graphiques (rectangles, lignes et textes). Ces éléments ne sont cependant pas stockés par les vues elles-mêmes, mais le sont dans le « canvas » géré par Tkinter. Afin de retrouver ces éléments graphiques, il faut utiliser soit leurs identifiants (ou « ID »), soit des marques (ou « tag ») associées à ces éléments ; il est alors possible de retrouver les éléments par leurs identifiants et/ou leurs tags.. Ceci est indispensable pour pouvoir redessiner les objets

Spécification

Le complément de la spécification du type abstrait *Fenetre* est le suivant :

***effaceGraphiques(fenetre)* :** efface tous les graphiques de la toile de la *fenetre*.

Travail à faire

Étape 5.2.1

- Implémenter la fonction *effaceGraphiques* du type abstrait *Fenetre*, en trouvant la ou les fonctions du « canvas » qui permettent de supprimer les éléments graphiques.
- Tester ces modifications en exécutant le module « TestAffichage.py ».

Bien comprendre :

- **la notion de « tag » du « canvas » de Tkinter, indispensable à l'implémentation de la fonction effaceGraphiques ;**
- **les 3 scénarios de test du module TestAffichage.**

2.2 Affichage du jeu

L'affichage du jeu consiste à effacer les graphiques précédemment affichés puis à appeler les fonctions des modules *VuePile* et *VuePioche*.

Spécification partielle

La spécification partielle du type abstrait *Jeu* est la suivante :

joue(jeu) **:** démarre une partie en lançant la boucle principale de la fenêtre.

majVues(jeu) **:** met à jours les vues de l'IHM du *jeu*.

Travail à faire

Étape 5.2.2

- Implémenter les fonctions précédentes du type abstrait *Jeu*.
- Tester ce code en exécutant le fichier « Pytatra.py ».

3 Interaction avec les joueurs

3.1 Diagramme d'activité

Le diagramme d'activité construit lors de l'étape 1 permet d'avoir une vision d'ensemble de déroulement d'une partie. Les activités qui y sont déclarées se retrouve dans l'implémentation de l'application et plus particulièrement dans le type abstrait *Jeu*.

La figure 3.1 fournit le détail du diagramme d'activité correspondant aux interactions utilisateur précises que le type *Jeu* doit mettre en œuvre.

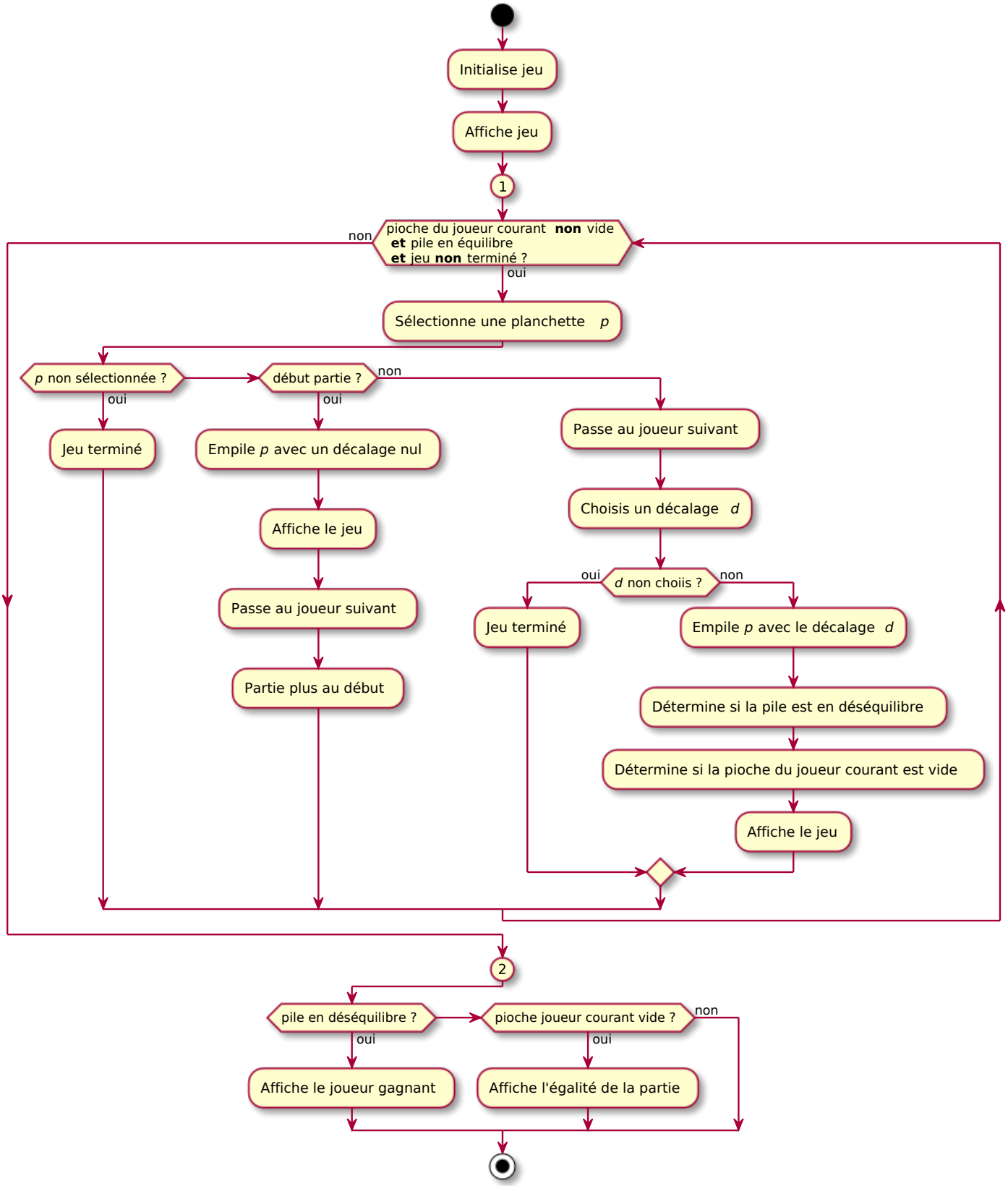

FIGURE 3.1 – Boucle principale de la fenêtre

Le module *Dialogue*, dont le code est entièrement fourni, permet les interactions utilisateur simples de type affichage de message et saisie d'information au clavier. L'utilisation exclusive de boîtes de dialogue, par opposition à des interactions qui auraient directement lieu sur les pioches des joueurs et la pile du jeu, rend l'implémentation du type abstrait *Jeu* assez simple. En particulier, la fonction *Jeu.activite* implémentera directement le diagramme d'activité de la figure 3.1.

3.2 Interaction

Spécification du type Fenêtre (suite et fin)

***quandOuverte(fenetre, fonction, argument)* :** invoque, une fois la *fenetre* ouverte, la *fonction* en lui passant un *argument*. Permet de lancer le jeu automatique dès lors que la fenêtre est ouverte suite à l'appel de *bouclePrincipale*.

quitte(fenetre) : ferme la fenêtre et quitte donc l'application.

Le code des fonctions *quandOuverte* et *quitte* est fourni.

Spécification du type Dialogue

***afficheMessage(message)* :** affiche une boîte de dialogue affichant le *message*. L'appui sur le bouton « Ok » ferme la boîte de dialogue et termine l'exécution de la fonction.

***saisisEntier(message)* :** affiche une boîte de dialogue affichant le *message* invitant l'utilisateur à saisir un nombre entier. L'appui sur le bouton « Ok » ferme la boîte de dialogue et la fonction retourne l'entier saisi. Si l'utilisateur n'a pas entré un entier, la fonction l'invite à en saisir un à nouveau. Si l'utilisateur appuie sur le bouton « Cancel », la valeur *None* est retournée.

***saisisFlottant(message)* :** affiche une boîte de dialogue affichant le *message* invitant l'utilisateur à saisir un nombre réel (à virgule flottante). L'appui sur le bouton « Ok » ferme la boîte de dialogue et la fonction retourne le nombre réel saisi. Si l'utilisateur n'a pas entré un nombre réel, la fonction l'invite à une nouvelle saisie. Si l'utilisateur appuie sur le bouton « Cancel », la valeur *None* est retournée.

***saisisNombre(message, entier)* :** appelée par *saisisEntier* et *saisisFlottant*.

Le code du module *Dialogue* est intégralement fourni.

Spécification du type Jeu (suite et fin)

***activite(jeu)* :** démarre l'activité liée à une partie, conformément à ce qui est décrit dans le diagramme d'activité.

***selectionnePlanchette(jeu)* :** demande au joueur courant de saisir au clavier le numéro de la planchette et retourne la planchette ainsi sélectionnée dans sa *pioche*. Si le joueur entre un numéro de planchette non valide (numéro non valable ou correspondant à une planchette non présente *dans* la pioche), une nouvelle demande est effectuée. Si le joueur annule l'interaction en activant le bouton « Cancel » dans la boîte de dialogue, la valeur *None* est retournée.

***choisisDecalage(jeu, planchetteAPoser)* :** demande au joueur courant de saisir au clavier le décalage souhaité pour la *planchetteAPoser* en haut de la *pile*. Le décalage doit avoir une valeur absolue minimale, le minimum dépendant de la dimension des planchettes (celle à poser et celle au sommet de la pile) et de la zone de marge de la planchette au sommet de la pile. Si le joueur entre un

décalage en deçà de la valeur minimum, une nouvelle demande est effectuée. Si le joueur annule l'interaction en activant le bouton « Cancel » dans la boîte de dialogue, la valeur *None* est retournée.

Les fonctions *selectionnePlanchette* et *choisisDecalage* sont appelés par la fonction *activite*, comme le suggère de le diagramme d'activité.

Travail à faire

Étape 5.3

- Récupérer la nouvelle version du fichier « Fenetre.py » et incorporer le code des fonctions *quandOuverte* et *quitte* dans le code du type *Fenetre*.
- Compléter le code de *Jeu.joue* de manière à ce que la fonction *Jeu.activite* soit invoquée quand la fenêtre est ouverte.
- Compléter progressivement le code de la fonction Jeu.activite de manière à implémenter pas à pas le diagramme d'activité.
  1. Commencer par construire la boucle de telle sorte qu'il soit possible de quitter l'application simplement (sans avoir a relancer le terminal par exemple).
  2. Compléter la partie relative à la sélection de la planchette (fonction *Jeu.selectionnePlanchette*).
  3. Compléter enfin la partie relative à la pose de la planchette (fonction *Jeu.choisisDecalage*).
  4. Tester régulièrement le code en exécutant le module *Pytatra*.

Bravo ! Le développement de l'application « Pytatra ! » est terminée. Terminée... ? Il reste probablement des « bugs » : il est donc nécessaire de tester plus longuement l'application en jouant au jeu. Si toutefois aucun « bug » n'est découvert, il reste peut-être du temps pour envisager quelques extensions :-) .

4 Et plus encore : quelques extensions possibles !

La liste suivante donne quelques extensions envisageables pour l'application :

1. Sauvegarde et restitution d'une partie
2. Amélioration de l'affichage

Il est possible de proposer vos propres extensions. Dans ce cas, faire valider l'extension proposé par l'enseignant. La page suivante donne quelques précisions pour chacune de ces extensions.

Dans tous les cas, ces extensions sont à gérer en mode « projet » : une forte autonomie est attendue et l'encadrant aidera principalement sur des questions techniques.

Ext. 1

Sauvegarde et restitution d'une partie :

- La partie en cours est sauvegardée lorsque la fenêtre est fermée alors que la partie n'est pas terminée.
- Lorsque l'application est relancée et qu'il existe une partie sauvegardée, l'application demande si la partie sauvegardée ou si une nouvelle partie doit être lancée.
- Le format du fichier est libre, mais il devra s'agir d'un fichier texte.
- Un unique fichier est à considérer (pas de gestion de multiples parties).

Ext. 2

Amélioration de l'affichage :

- Afficher les pioches et les piles avec des vues 3D en perspective
- Le jeu reste sur une IHM 2D.