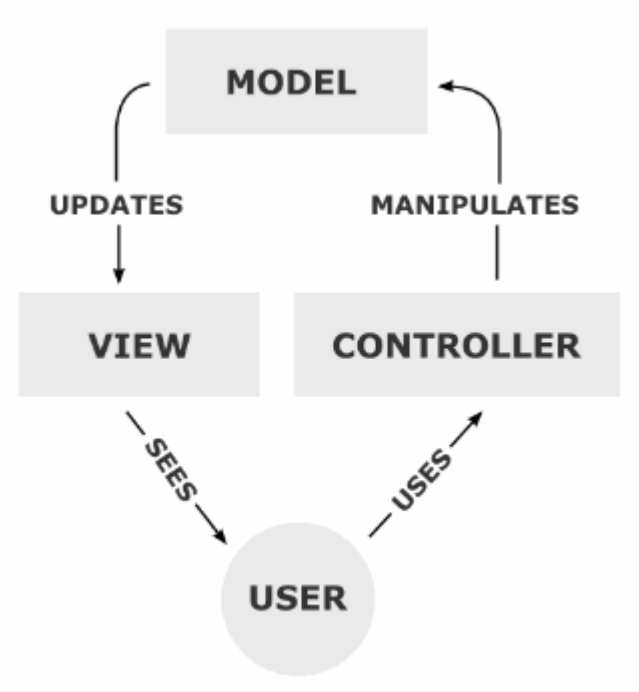
Rapport MVC

Question 1 :

Le code du modèle de jeu ne suit pas l'architecture MVC ?
Donnez les parties du code qui ne respectent pas ce pattern. Argumentez.
En quoi l'architecture mise en place limite l'évolution du code ?

Tout d'abord il est important de bien définir ce qu'est le MVC : « model vue contrôleur » c'est à dire que le contrôleur va aller modifier le model après une action de l'utilisateur puis le model met alors à jour la vue.



Dans le code Devint on peut effectivement voir qu'il ne suit pas l'architecture MVC sachant qu'il est architecturé en game loop. C'est à dire que toutes les x secondes le moteur relance les méthodes suivantes : update, render, reset.
Au temps 0 il lance la méthode init qui permet d'initialiser le jeu.

Pourtant dans le code actuel on retrouve tout de même des aspects contrôleurs dans le constructeur de Menu.

```
// la gestion des touches directionnelles haut et bas
addControl("DOWN", new DownAction(this));
addControl("UP", new UpAction(this));
```

On retrouve aussi des aspects de vues dans des classes contenant du contrôleur (Jeu) qui est la base des jeux Devint.
Donc on ne peut pas dire que le code fourni est totalement game loop ni totalement MVC. C'est un code hybride.

L'architecture mise en place limite l'évolution du code car comme on a du game loop on ne pas séparer proprement les différentes parties. A chaque fois on doit réimplémenter les méthodes héritant de Jeu.

Question 2 :

Comment pouvez-vous modifier le code du modèle de jeu pour qu'il respecte MVC ?
Donnez les parties de code à modifier et leurs modifications.
Justifiez.

Il faut séparer dans des classes différentes la base de données, la vue et la gestion des évènements (clics et appuis sur clavier). Il faut que ces classes "s'écoutent". Le contrôleur capte un évènement sur la vue et émet un changement au modèle pour changer les données. Ensuite le modèle change les vues !

Pour ce faire il faudrait créer plusieurs classes contrôleurs en fonction de si l'on est dans les menu ou dans une partie afin de gérer les différents inputs utilisateurs. Il faudrait ensuite créer une classe qui stocke et gère toutes les éventuelles données que l'on peut avoir (difficulté, mode de jeu, etc..) et qui gère donc aussi les vues. Les classes contrôleurs seront liées à la classe modèle pour pouvoir émettre les changements à faire sur les vues à chaque event/input utilisateurs.

Deux exemples pour illustrer notre propos :

```
public void loop() {
    long lastLoopTime,timeLoop;

    // initialisation des valeurs
    reset();

    while (this.isDisplayable()) {
        long now = System.nanoTime();
        lastLoopTime = now;

        // mise  jour des informations
        update();

        // r-affichage  chaque tour
        // optimisation possible : afficher seulement quand certaines informations ont chang
        // ncessite de lisser l'affichage d'un temps t  t+delta
        render();

        try {
            timeLoop = (lastLoopTime - System.nanoTime() + 1000000000L/60) / 1000000;
            if(timeLoop>0) {
                Thread.sleep(timeLoop);
            }
        } catch (InterruptedException e) {
            throw new IllegalArgumentException("");
        }
    }
}
```

On voit ici l'architecture en game loop qu'il faudrait donc changer à savoir qu'il faudrait faire en sorte que la vue change à chaque fois qu'une touche est appuyée ou qu'un clic est effectué

```
     */
    public Menu() {
        listeBoutton = new ArrayList<JButton>();
//          this.getSIVOX().playWav(ACCUEIL_SON);

        GridBagLayout layoutMenu = new GridBagLayout();
        c.fill = GridBagConstraints.BOTH;
        c.insets = new Insets(MARGE_TOP_BOT, MARGE_LEFT_RIGHT, MARGE_TOP_BOT,
                MARGE_LEFT_RIGHT);
        c.fill = GridBagConstraints.BOTH;
        c.ipady = 100;
        c.gridwidth = 3;
        menuPrincipal.setLayout(layoutMenu);

        addLabel(TITLE_GAME);

        // les options possibles
        addMenu("Labyrinth Mystère",new Action(this,1));
        addMenu("Score", new Action(this,4));
        addMenu("Quitter", new Action(this, 5));



        // la gestion des touches directionnelles haut et bas
        addControl("DOWN", new DownAction(this));
        addControl("UP", new UpAction(this));

        this.add(menuPrincipal);
        this.setVisible(true);
    }
```

```
    /**
     * Permet d'ajouter du texte dans le menu
     * @param title Le titre du jeu
     */
    public void addLabel(String title) {
        this.titleJeu = new JLabel(title.toUpperCase(), JLabel.CENTER);
        c.weightx = c.weighty = 1.0;
        c.gridy = countMenu++;
        menuPrincipal.add(this.titleJeu, c);
    }
```

On voit ici que dans la classe Menu on a des aspects contrôleurs (addControl) et des aspects view (addLabel) il faudrait donc regrouper un seul aspect dans cette classe et transférer l'autre dans une autre classe

Question 3 :

Quelles sont les spécificités des jeux DEVINT ?
Est-ce que la solution MVC respecte les besoins et si oui comment ?
Comparez les 2 architectures et leurs avantages par rapport aux besoins de DEVINT.

Les jeux DEVINT s'adressent à un public déficient visuel, ils doivent donc avoir une interface assez concise et adaptée ainsi qu'un suivi et retour sonore tout au long du jeu. De plus les jeux DEVINT sont de natures très variées, on peut retrouver des jeux de plateaux à des jeux dynamiques.

La solution MVC respecte les besoins des jeux Devint puisqu'elle permet de séparer la partie graphique des autres parties du code et donc de changer complètement la vue en fonction des différentes déficiences des joueurs sans changer le reste du code et donc le reste du projet. De plus elle permet une interaction entre un utilisateur et le model du jeu.

L'avantage de la structure MVC est donc de pouvoir parfaitement contrôler ce qui se passe. L'inconvénient est que pour un jeu où on aura besoin de rafraichir en continue la vue sans forcément que l'utilisateur appuie sur un bouton on sera coincé. Or pour pouvoir répondre à ce problème on a besoin d'une architecture en game loop. Ce qui fait que l'architecture actuelle permet effectivement de supporter les jeux de types plateau tel un échiquier, ou un jeu de shoot où il faudrait tirer sur des cibles en mouvement.
Et donc que tous les élèves puissent développer le jeu qu'ils veulent. Sans à avoir à modifier le code fourni.