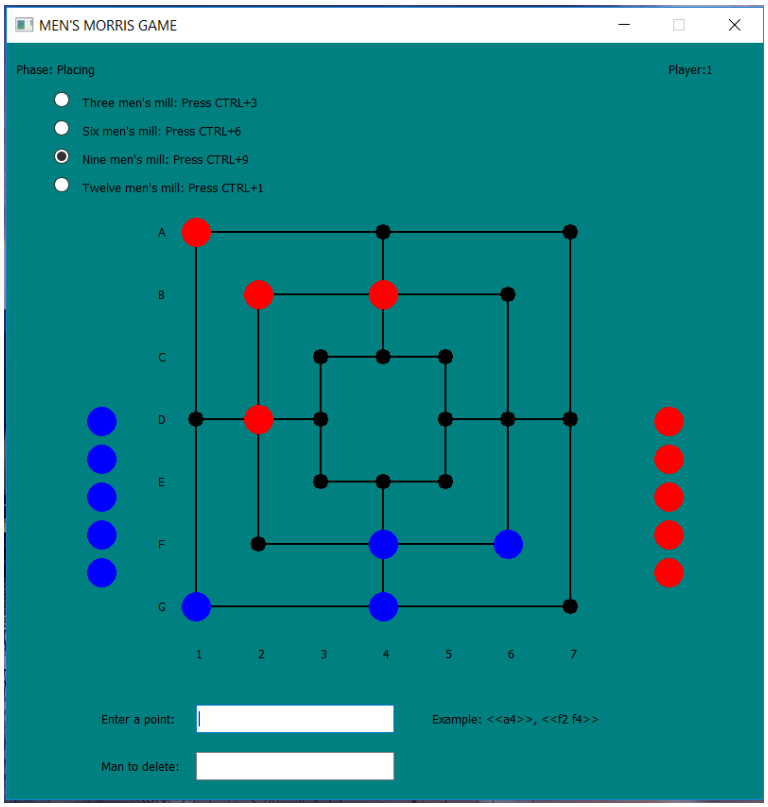
MEN'S MORRIS

PROJET INFROMATIQUE I4



15 JUIN 2019

ÉTUDIANT: BEHROOZ BOZORGMEHR

Hes·so VALAIS WALLIS

Table des matières

INTRODUCTION :

Dans le cadre de cours d'informatique, de quatrième semestre, pour mettre en pratique et approfondir notre connaissance appris pendant ce cours nous réalisons un projet.

Jeu du moulin, Men's Morris en anglais, est un jeu de société à travers le monde. L'auteur de ce rapport a choisi ce jeu pour son projet pratique. Pour vous familiariser avec ce jeu, tout d'abord dans le chapitre suivant nous parlerons des règles de ce jeu et en suite les autres parties de ce texte vous montrera les différentes étapes pour accomplir ce projet.

Règles de jeu[1] :

Le jeu se déroule en trois phases :

- La pose
- Le mouvement
- Le saut

À tout moment du jeu, celui qui réalise un moulin, c'est-à-dire l'alignement de trois de ses pions, peut capturer un pion adverse quelconque parmi ceux n'appartenant pas à un moulin.

Phase 1 - La pose

Tant qu'il en possède encore, chaque joueur place à tour de rôle un pion sur une intersection libre. La phase 2 débute après que les joueurs ont placé tous leurs pions.

Phase 2 - Le mouvement

Lorsqu'il n'a plus de pion à poser, chaque joueur fait glisser l'un de ses pions vers une intersection voisine libre en suivant un chemin prévu. La phase 3 débute dès que l'un des joueurs est réduit à 3 pions.

Phase 3 - Le saut

Celui qui ne possède plus que trois pions, peut alors se déplacer en sautant où il veut.

Le jeu s'achève quand un joueur n'a plus que deux pions ou ne peut plus jouer, il est alors le perdant.

SPECIFICATION DE JEU :

Avec 3 pions :

Trois lignes horizontales, trois lignes verticales, trois pions pour chaque joueur et six possibilités de faire un moulin.

Avec 6 pions :

Six lignes horizontales, Six lignes verticales, six pions pour chaque joueur et huit possibilités de faire un moulin.

Avec 9 pions :

Huit lignes horizontales, huit lignes verticales, neuf pions pour chaque joueur et seize possibilités de faire un moulin.

Avec 12 pions :

Huit lignes horizontales, huit lignes verticales, 4 lignes diagonales, douze pions pour chaque joueur et vingt possibilités de faire un moulin.

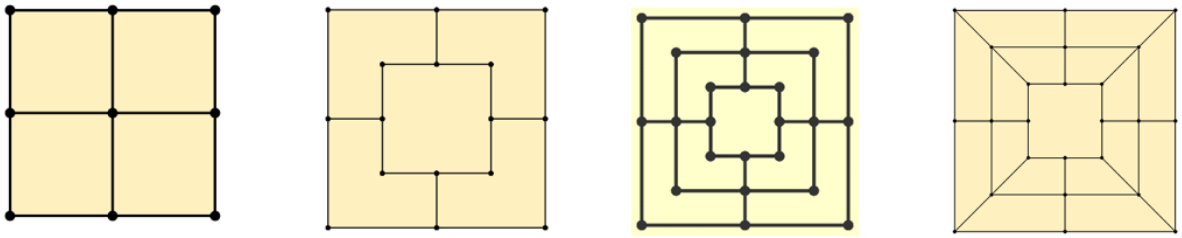


Figure 1 : trois, six, neuf et douze pions – moulin

ANALYSE :

Dans un première temps ce chapitre et les chapitres suivants s'occupent de réaliser le jeu pour neuf pions et ensuite dans un chapitre ultérieur nous voyons les modifications nécessaires pour les restes formes de jeu.

Nous avons besoins d'un milieu de communication avec les joueurs, disons un élément physique qui reçoit les commandes d'utilisateur « le clavier », un élément physique qui traite les mots donnés par l'utilisateur « le processeur », et un élément physique qui transmet à utilisateur le résultat de ses ordres « l'écran »

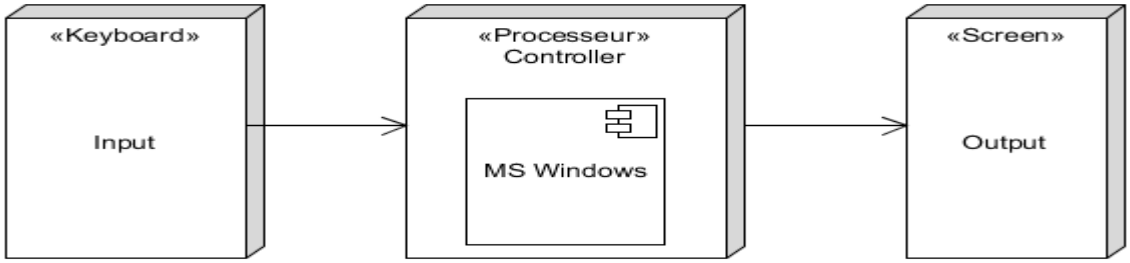


Figure 2 : Diagramme physique de jeu

Ce jeu a un terrain de jeu qui est constamment fixe et le joueur souhaite de le voir à tout moment de jeu sur l'écran. Donc nous avons un objet « Field » qui peut représenter se terrain de jeu. Chaque joueur doit avoir à disposition neuf objets «2x9 Man » qui représentent les pions de jeu. Un objet qui peut recevoir les commandes d'utilisateur via le clavier, un « InputPanel ». Les commandes des utilisateurs doivent respectent les règles de jeu donc un objet doit les contrôler. C'est le rôle d'un objet « Controller ». Finalement un objet qui contenir les Field, Men, et InputPanel et accepte les commandes passées par le Contoller, un « Window ». Il facilite la communication avec les joueurs.

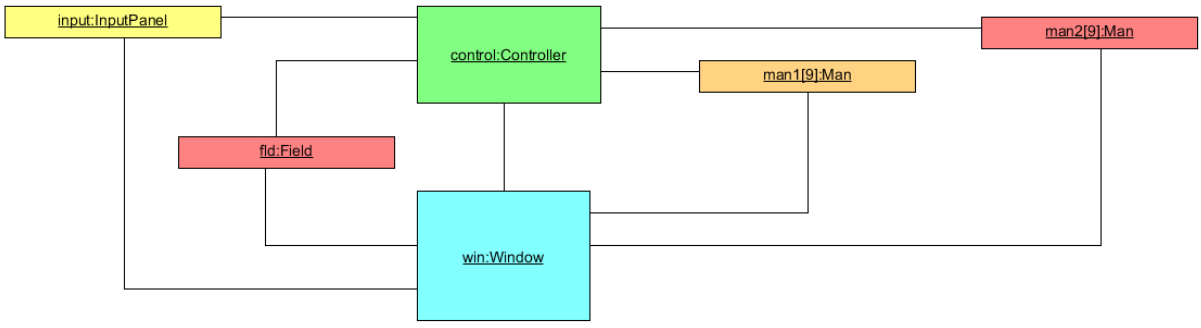


Figure 3 : Diagramme des objets de programme de jeu

CONCEPTION :

Pour réaliser le projet dans ce chapitre nous allons définir plus en détail les classes montrées dans le diagramme d'objets, figure 3.

- Window : une fenêtre principale, un texte box en bas de celle pour permettre à joueur d'écrire sa commande. Deux indicateurs pour mentionner le tour de rôle de chaque joueur et la phase de jeu. Chaque indicateur est placé dans un coin en haut de cette fenêtre. Au milieu on pose le tapis de jeu en forme carrée. Le tapis doit changer selon le nombre de pions définis au debout. Donc quatre buttons pour définir le type de jeu. On les pose en haut à gauche de la fenêtre. Tous les indicateurs sont réalisés par la classe QLabel, les buttons par la classe QRadioButton et texte box par la classe InputPanel.

Il faut noter qu'on parle d'une position (précise ou non précise) dans la fenêtre, c'est simplement une indication banale. Le but principal de ce chapitre est de définir le lien entre les objets.

- Field : le tapis de jeu est composé des lignes horizontales et verticales. On met un code en forme alphabétique pour les lignes horizontales et numérique pour les lignes verticales. De cette façons chaque intersection a un code d'alphanumérique qui permettre au joueur de mentionner le point où il souhaite poser son pion. On pourrait utiliser le même codage pour mentionner les extrémités des lignes. Les linges sont réalisés par la classe QLine, le codage par QLabel, et les extrémités par la classe QPoint.

- Man : un pion peut être un disque qui a une couleur différente de celle de pion l'adversaire. Mais pour le connaitre parmi les autres pions de son groupe il faut qu'il ait un ID.

- InputPanel : une sous fenêtre qui inclut le texte box qui reçoit les mots de l'utilisateur et les transmet à l'organe de control. Ce texte box est réalisé par la classe QLineEdit.

- Controller : une classe qui n'est pas visible par un objet dans la fenêtre de jeu, mais qui une influence très importante sur les éléments de jeu présentés dans cette dernière. Car cette classe s'occupe de tout logique de jeu il doit avoir un accès non négligeable aux composants de jeu. À ce but il doit avoir accès aux objets définis par les classes précédentes.

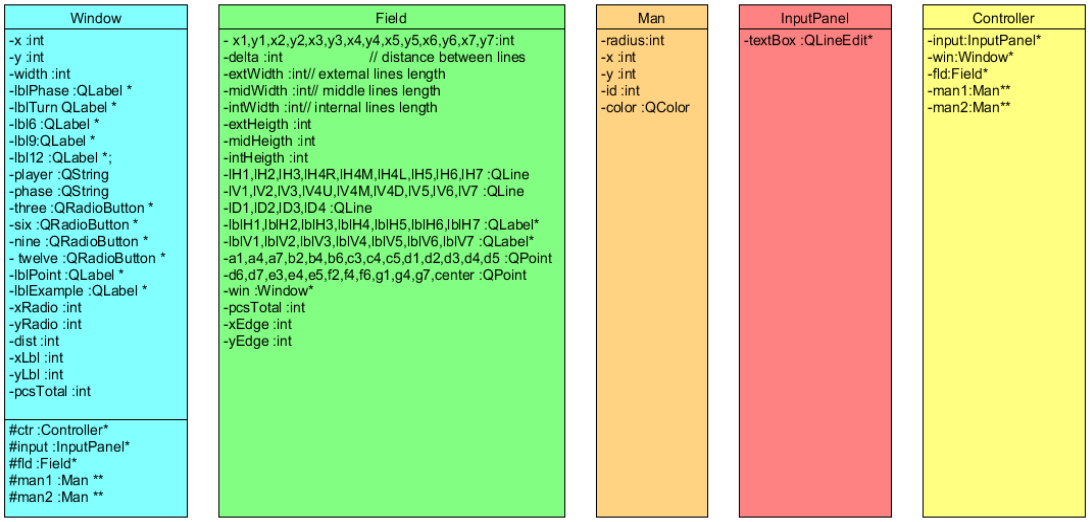


Figure 4 : Les classes en premier étape de conception montrent leurs composants

La figure 4 montre simplement les attributs des classes. Le lien entre les classes est brièvement discuté par le texte-dessus. Le diagramme des classes suivant montre ce lien dans une langage technique. Il montre aussi les méthodes dont nous avons besoins pour faire les tâches précises dans cadre de ce jeu. Il faut préciser que le diagramme dessus inclut aussi une classe Factory qui s'occupe de construire les éléments principaux, les classes mentionnées-dessus, de jeu et mettre les tout première relations entre aux. Cette dernière classe détruira tous éléments une fois que l'utilisateur ferme la fenêtre de jeu.

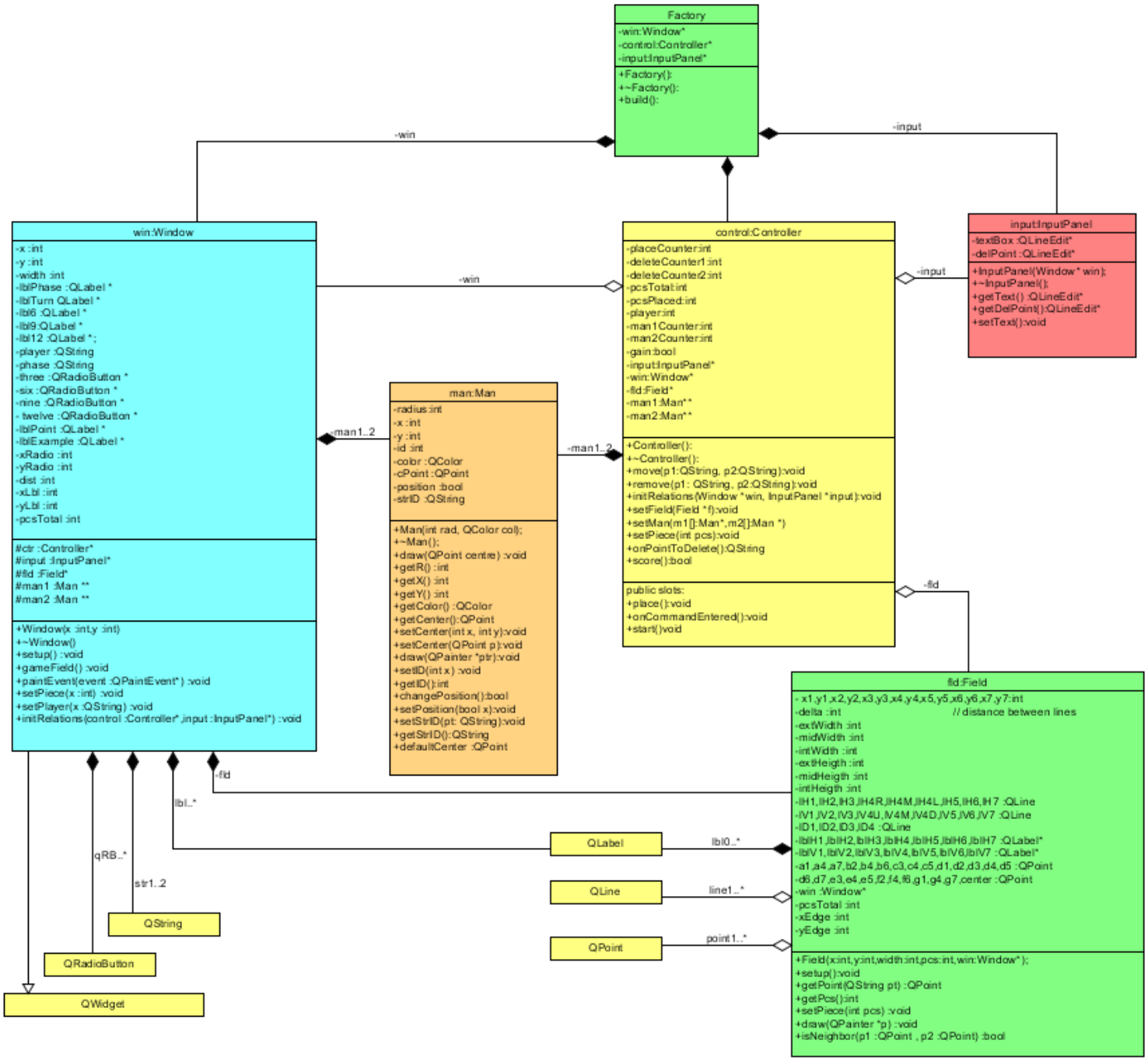


Figure 5 : Diagramme des classes, montre les liens entre différents objets de ce jeu

IMPLEMENTATION :

Précédemment nous avons vu le modèle de notre programme, les liens définis entre les classes et nous sommes informés de règles de jeu. Maintenant avec ce data nous pouvons définir les algorithmes pour les méthodes de chaque classe à but de faire une tâche souhaitée. Mais d'abord par une machine d'état nous pouvons montrer comment nous voulons que le programme fonctionne.

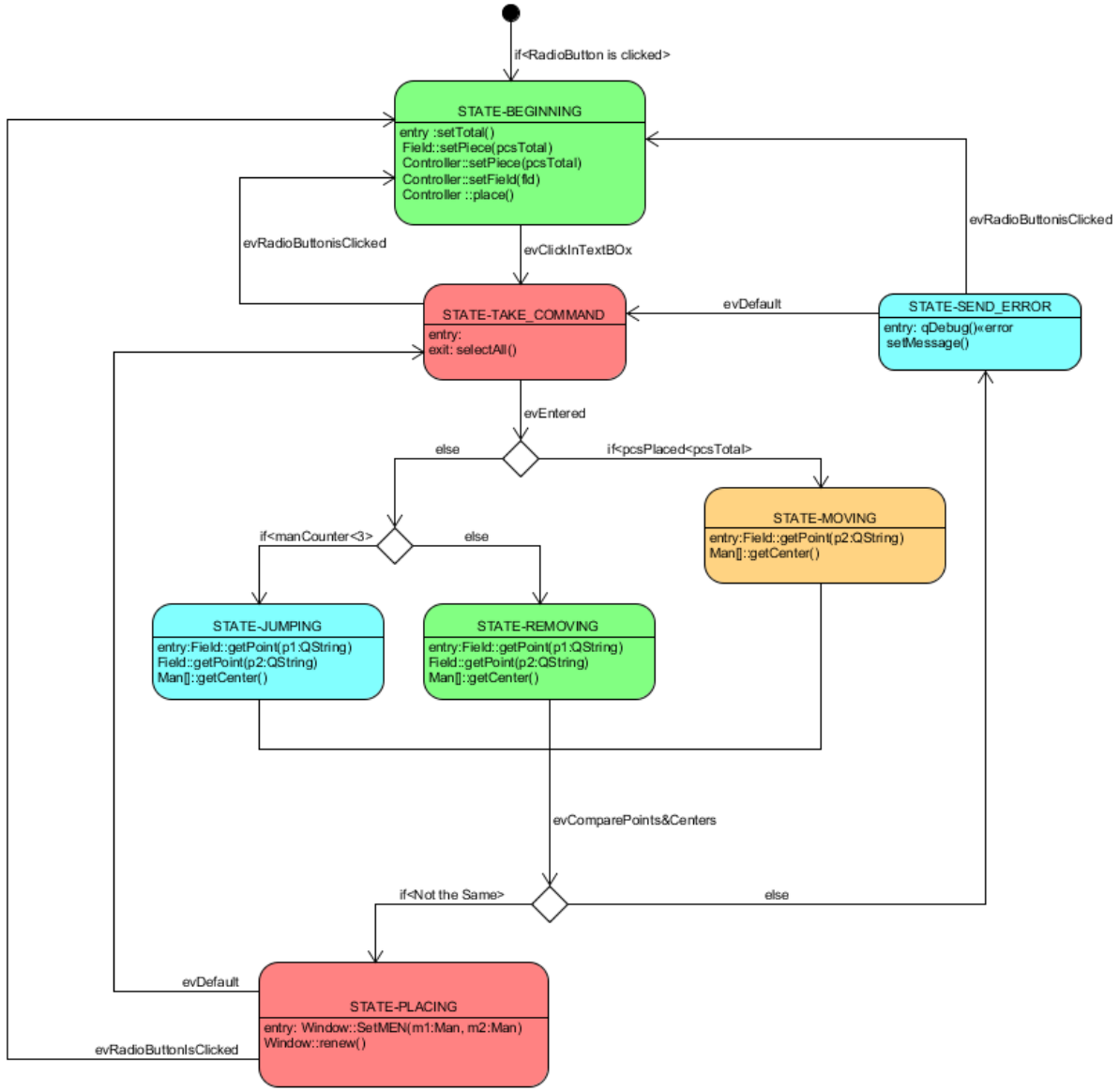


Figure 6 : Machine d'états transitions de jeu

Comme la figure 6 montre, une fois que l'utilisateur choisit une version de jeu par un radio-button le programme entre dans l'état de commencement « STATE-BEGINNING ». Il déroule le tapis de jeu et poser le nombre de pions pour la version voulue. Après que l'utilisateur entre dans le texte box, il est près de prendre la commande, « STATE-TAKE_COMMAND », et en touchant « Enter » traite l'ordre reçu. Pour un traitement adéquat selon les règles de jeu, le programme se pose certaines questions. Dans un premier temps à savoir dans quelle phase on est, « STATE-MOVING », « STATE-REMOVING » ou « STATE-JUMPING ». La différence entre ces états est le pas de mouvement. Ensuite si le/les point/s donné/s est/sont correct/s on demande au programme de la placer au point souhaité « STATE-

PLACING », sinon envoie une erreur à avertir le joueur « STATE-SEND_ERROR ». Puis attend pour une nouvelle commande corrigée « STATE-TAKE_COMMAND ».

À tout moment on peut recommencer le jeu en cliquant sur un radio-button.
La machine état de la figure 6 s'occupe de la globalité de jeu. Evidement pour un bon fonctionnement de programme, la partie logique de jeu doit répondre aux autres questions à l'intérieur un état donné qui ne sont pas en cadré par la figure 6, à titre d'exemple le gain de chaque partie. Pour une vue plus détaillée de ce que passe, dans la partie logique, derrière la fenêtre principale de jeu nous regardons les diagrammes de séquences suivants.

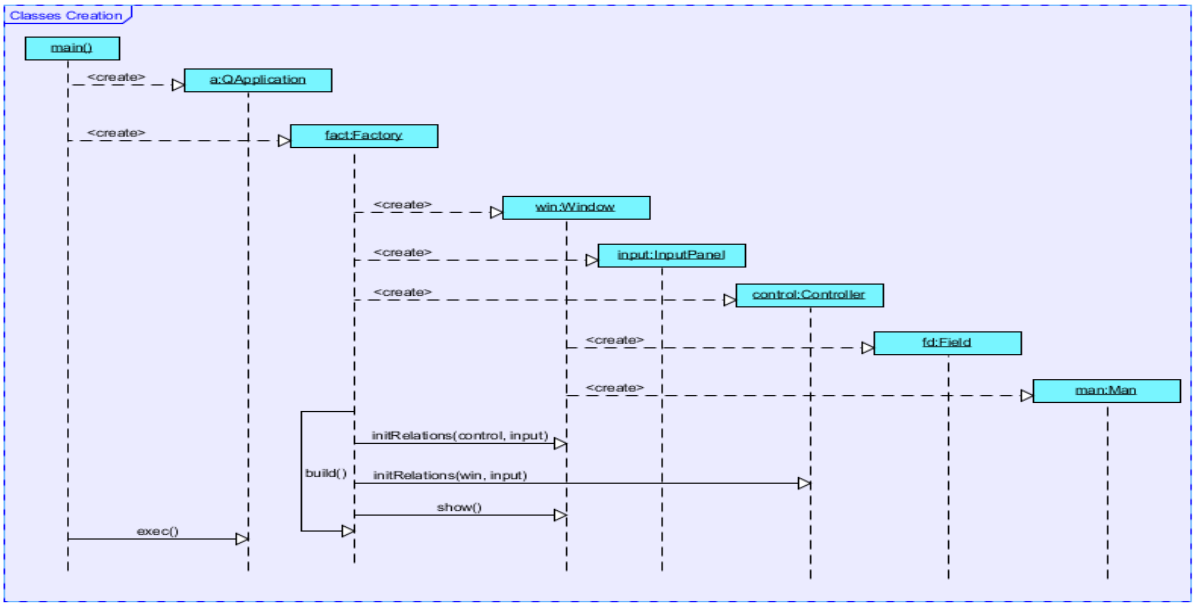


Figure 7 : Diagramme de séquence de création des objets. À lancement de jeu la classe Factory crée les éléments principaux « input, control, win (output) ». « Win » crée le terrain de jeu « fld » et les pions « man».

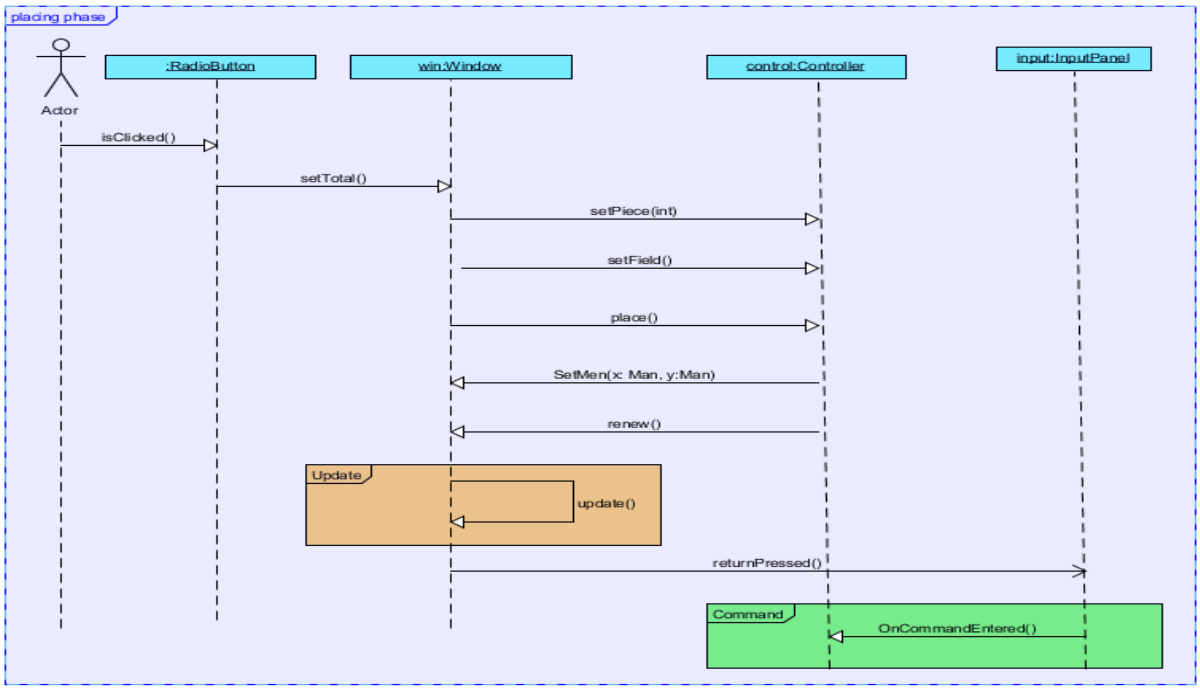


Figure 8 : Séquence d'arranger le tapis de jeu et les pions avant commencement de jeu. En cliquant un « RadioButton » le terrain de jeu et nombre des pions change.

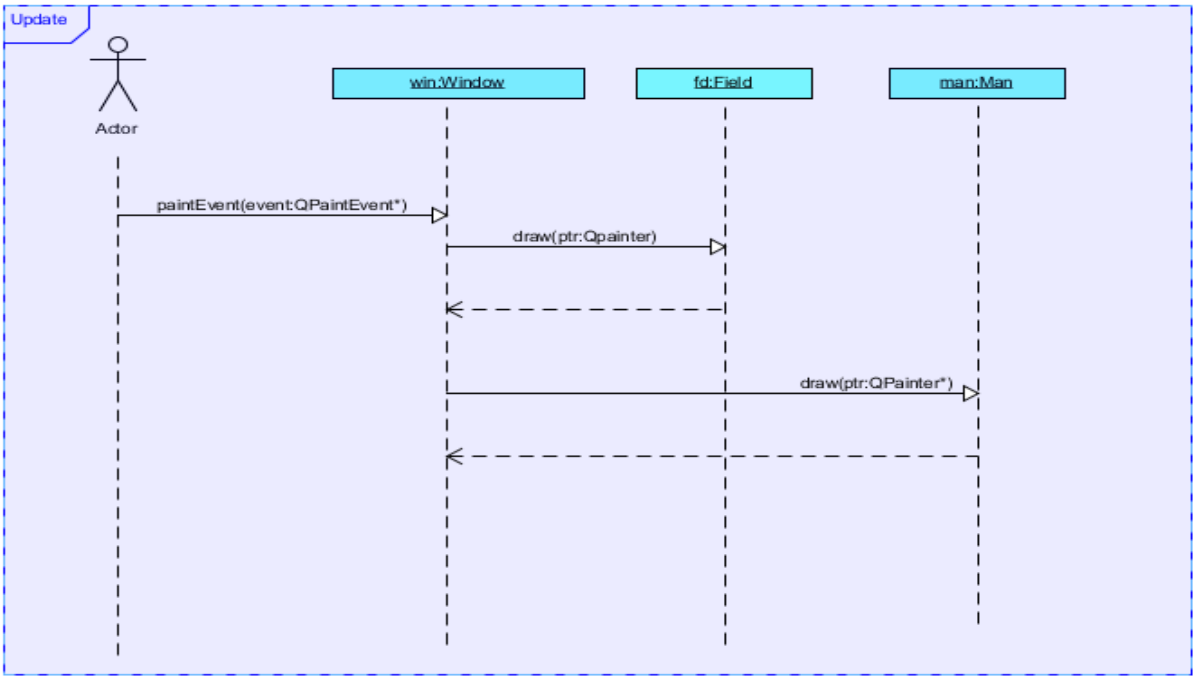


Figure 9 : Séquence de mise à jour de la fenêtre de jeu après définir la version de jeu, ainsi qu'après chaque mouvement.

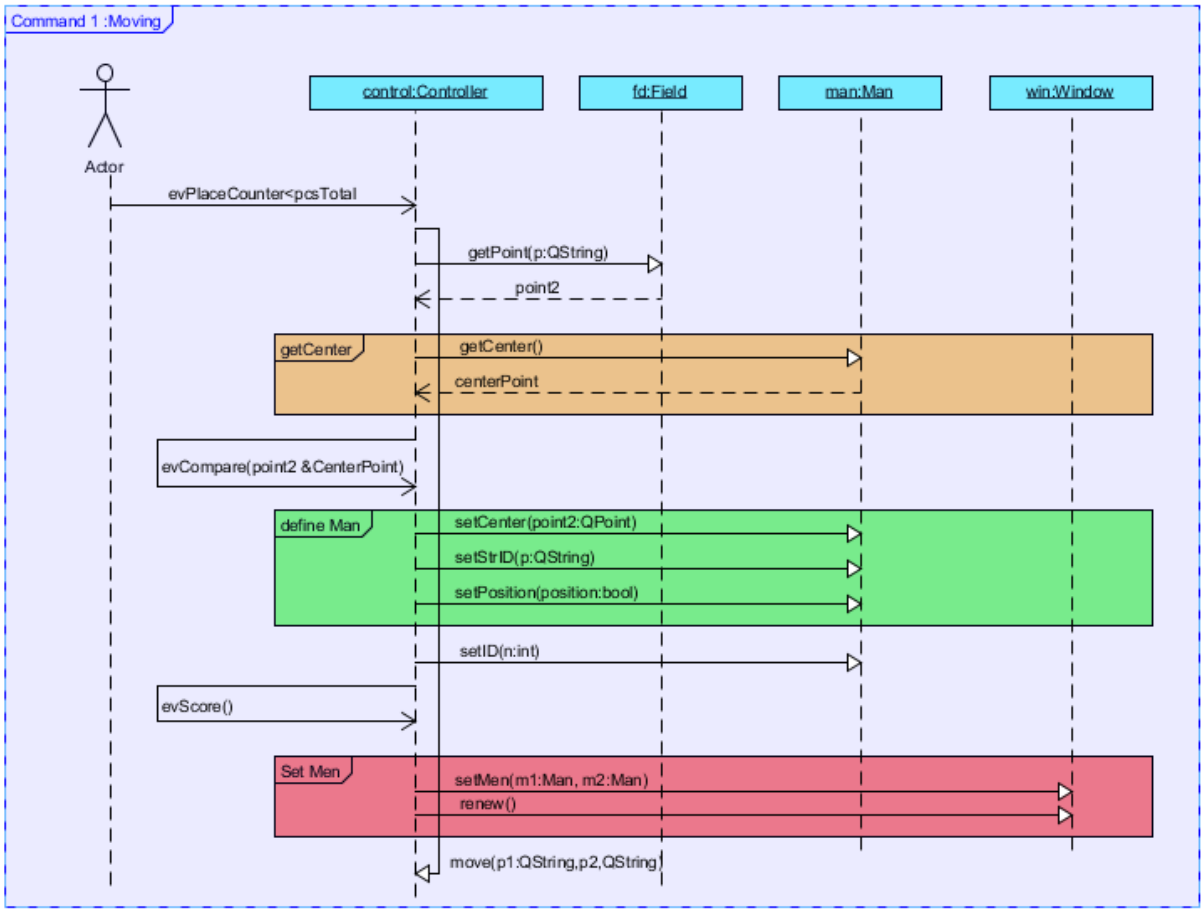


Figure 10 : Diagramme de séquence en phase poser un pion. Dans cette phase le joueur peut poser son pion sur un point qui n'est pas occupé par un autre pion. À ce but on contrôle si le code donné par joueur est un point libre et il existe sur le cadre de jeu. Dans cette hypothèse que c'est le cas le pion est posé à point souhaité. Sinon un message d'erreur est envoyé au joueur.

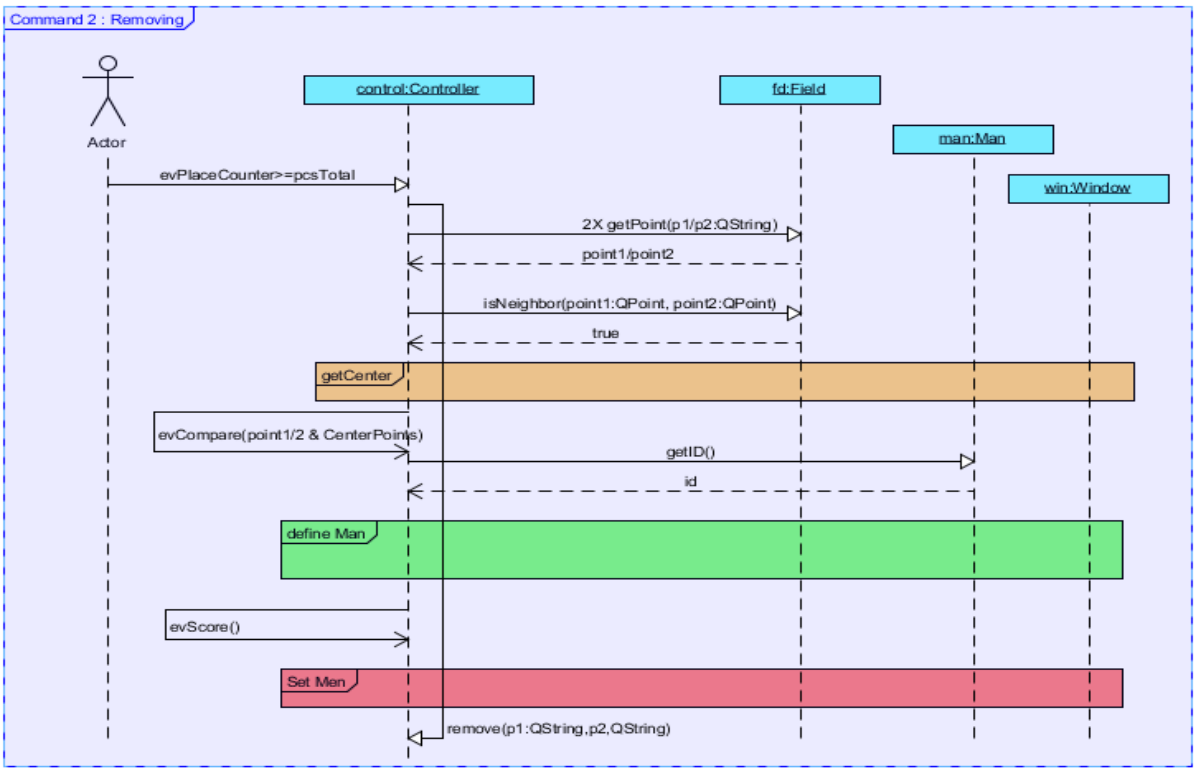


Figure 11 : Séquence de mouvement une fois que tous les pions sont posés sur le tapis. Dans cette phase le joueur ne peut bouger qu'un pas dans un point voisin libre. Dans cette phase d'abord le joueur définit quel pion doit bouger et ensuit à quel point voisin doit se poser. Pour savoir quel pion choisi par le joueur on demande l'ID de pion posé à point de départ. Ensuit on le donne simplement la coordonnée de point voisin souhaité et on fait l'appel au méthode renew() de « win » pour les mises à jour.

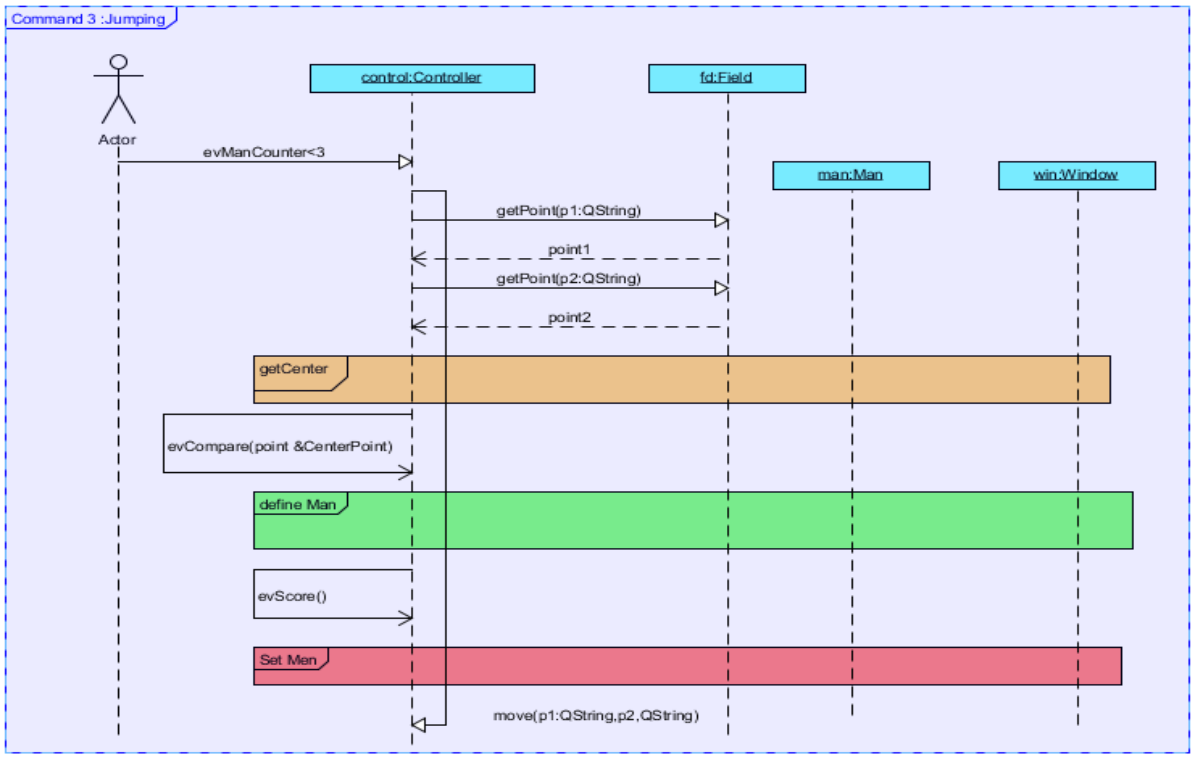


Figure 12 : Séquence de saut une fois qu'un joueur n'a que 3 pions. Dans ce cas le point de déplacement peut être n'importe où sur le tapis à condition qu'il est libre pour poser le pion

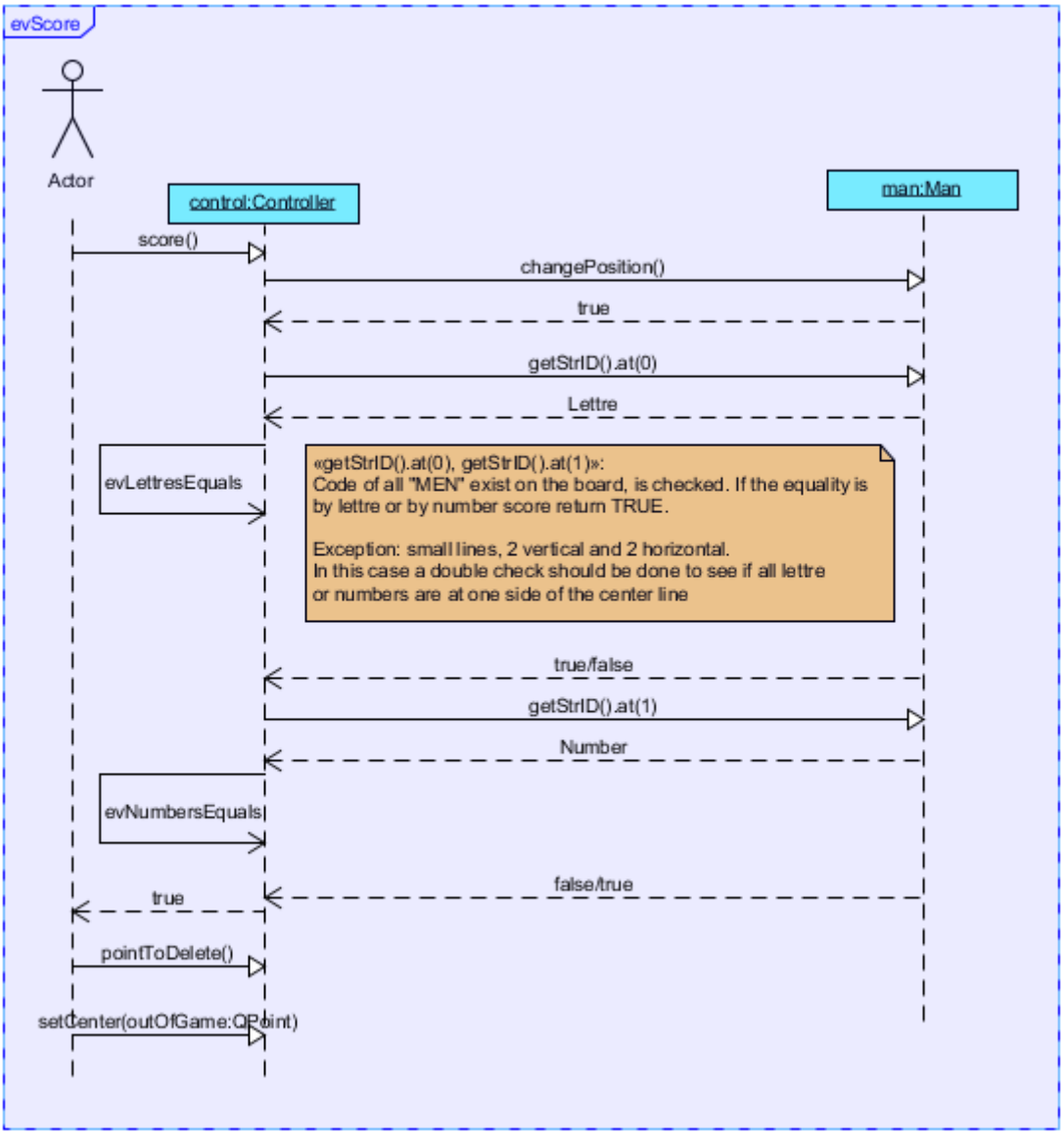


Figure 13 : Séquence d'avoir un gain par une partie. Le résultat, « TRUE », de méthode score() permet au joueur d'éliminer un pion de la partie adversaire. « Actor » ici est une de deux méthodes, move() ou remove(), de la classe Controller.

VERIFICATION :

Pour un bon fonctionnement de programme de jeu on lance des tests pour vérifier la fonctionnalité de programme. On utilise différente stratégie de jeu et différent scenarii pour voir si une erreur d'exécution se montre. Il y a un bug dans l'étape d'élimination d'un pion adversaire en 2ème phase de jeu.

AMELIORATION :

Tout d'abord on doit enlever le bug trouvé. La conception de base de la fenêtre principale montre différente version de jeu. Pour que toutes les versions fonctionnent il faut apporter des modifications nécessaires comme redéfinir les points de codage pour tout version selon les codes de celle-là. Pour diminuer la complexité du code une solution est d'ajouter le propre code à chaque version. L'algorithme de jeu en mode ordinateur-joueur qui a une complexité plus élevée pourrait écrire pour une version plus simple de jeu comme la version à trois pions. Ensuite le développer pour une version plus difficile.