RAPPORT MIF02

BOUNEFFA MASSINISSA P1310939

ABEBE FANUEL P1311388

1) FONCTIONNALITÉS

Dans le cadre du projet de MIF02 nous avons été amenés à reprendre un jeu de course de Poney simple afin de l'améliorer. Au départ nous n'avions qu'une simple fenêtre avec des Poneys qui couraient de gauche à droite et il n'y avait aucun design pattern appliqué. Nous avons fait le choix d'utiliser trois Design Pattern MVC, Observer et Factory, des choix que l'on justifiera plus tard. Au final nous sommes arrivés à un jeu fonctionnel où l'utilisateur contrôle deux Poneys (Bleu et Orange) et où le reste est contrôlé par une Intelligence Artificielle (IA). Pour ces derniers l'utilisateur a le choix entre 3 tactiques :

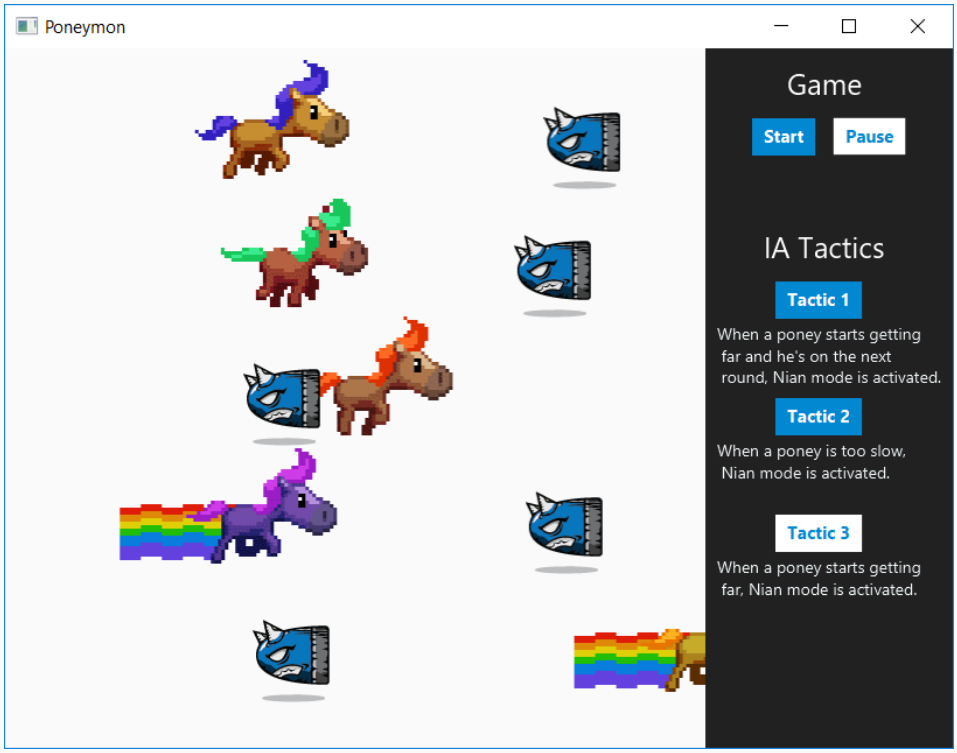
- Quand un Poney voisin au Poney contrôlé par l'IA s'éloigne et qu'il a un tour de plus on double la vitesse du Poney contrôlé par l'IA.
- Quand un Poney est trop lent on double sa vitesse.
- Dès qu'un Poney voisin au Poney contrôlé par l'IA commence à s'éloigner on double la vitesse.

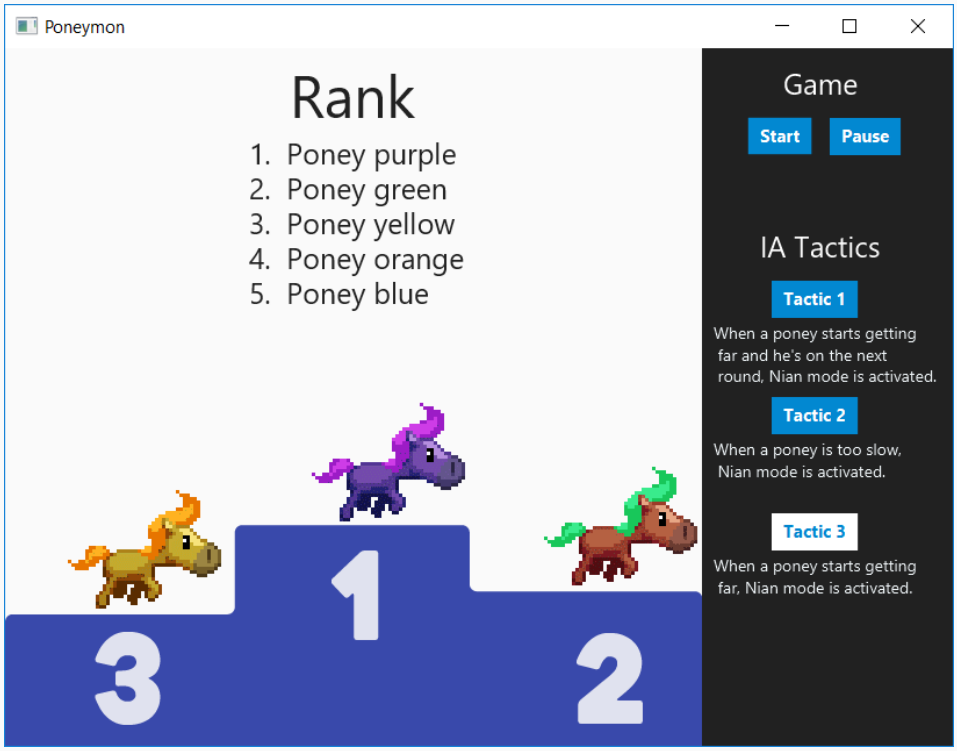
L'utilisateur peut arrêter/lancer le jeu quand il le souhaite en appuyant sur Pause/Start. Il peut aussi changer la tactique des Poneys qu'il ne contrôle pas à tout moment en sélectionnant la tactique qu'il désire à droite de l'écran.

Nous avons aussi ajouté aux jeu des obstacles qui peuvent être soit statiques, dans ce cas leur position est changée après chaque tour, soit être mobiles et dans ce cas ils ont une vitesse et vont dans le sens contraire des Poneys. Pour éviter les Obstacles les Poneys peuvent sauter, si un obstacle est touché il fait ralentir la vitesse du Poney. Pour faire sauter les Poneys que l'utilisateur contrôle il faut appuyer sur 1 (Poney Bleu) ou 3 (Poney Orange). L'utilisateur peut aussi activer le mode NianPoney qui fait augmenter la vitesse du Poney par deux mais que l'on ne peut utiliser que pendant un seul tour (Appuyer sur B pour le Poney Bleu et O pour le Poney Orange).

Le jeu arrive à sa fin quand tous les Poneys ont fait 5 tours. On affiche ensuite le classement ainsi que le Podium final.

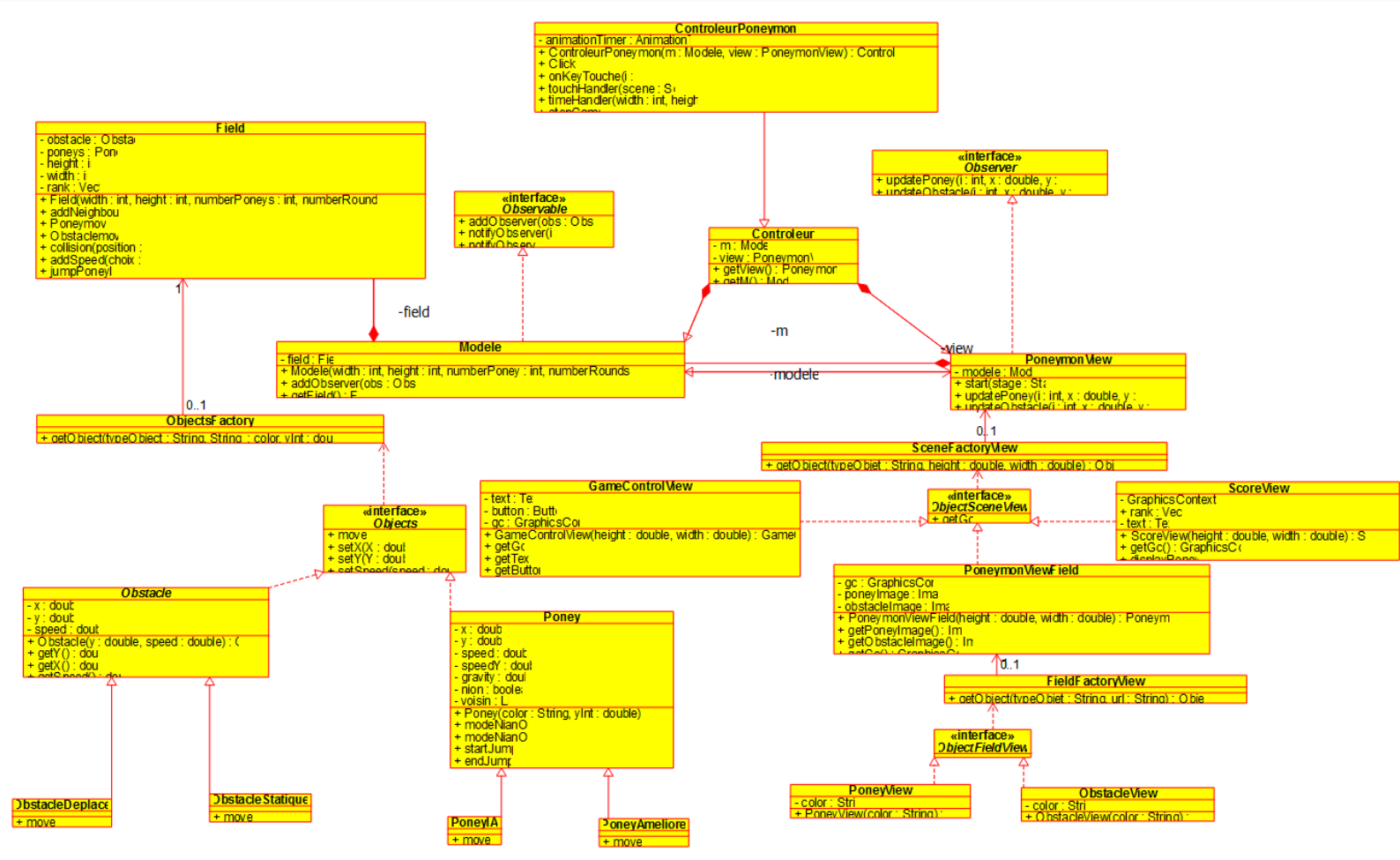
Voici les écrans du jeu final :





2) CHOIX TECHNIQUES

Diagramme de classe de notre application :



Tout d'abord pour pouvoir faire la distinction entre les Poneys gérés par l'utilisateur et ceux gérés pas l'IA, on a créé deux classes de types Poney (PoneyAmeliore et PoneyIA qui héritent de Poney) et deux classes de types Obstacle (ObstacleStatique et ObstacleDeplace qui héritent d'Obstacle). On a donc deux classes abstract Poney et Obstacle.

Dans notre terrain de jeux (classe Field), on peut avoir autant de Poney et d'Obstacles que l'on veut.

Pour que la création des objets dans notre terrain soit plus facile à gérer, nous avons créé une « usine » (ObjectFactory class) qui permet de créer des objets plus facilement. Pour cela nous avons suivi le pattern Factory.

L'architecture de notre projet suit un modèle MVC. Notre vue connait le modèle. Et le contrôleur utilise la vue et le modèle. Le contrôleur récupère tout ce qui se passe dans le modèle pour le transmettre à la vue qui va ensuite afficher les bonnes informations.

Pour pouvoir correctement effectuer ces échanges nous avons des observer et des observable. Notre observer a pour rôle de notifier la vue de tous les changements.

Exemple : Lorsque le Poney fait un mouvement dans le modèle, on notifie la vue pour qu'elle fasse la mise à jour de la position du Poney.

Donc pour cela nous avons eu besoin d'utiliser le pattern Observer.

Un petit récapitulatif :

- Contrôleur :

Notre contrôleur sert d'intermédiaire entre la vue le contrôleur. C'est dans le contrôleur que se trouvent les Events Handlers du jeu.

- Vue :

Notre vue principale (PoneymonView class) est composée de trois vue, le contrôle du jeu (gamecontrolview class), l'affichage du score(ScoreView) et le déroulement du jeux(PonemonViewField). Ici aussi pour pouvoir créer des vues facilement, nous avons créé une « usine » (scenefactory) pour pouvoir créer des vues qui fait références aux patterns Factory.