**Projet intégrateur**

**ANALYSE**

**Par**

**Gabriel Ouellet**

**Michael Duchesne**

**Pascale Morin**

**Travail présenté à**

**M. Michel Payette**

**Dans le cadre du cours Projet intégrateur**

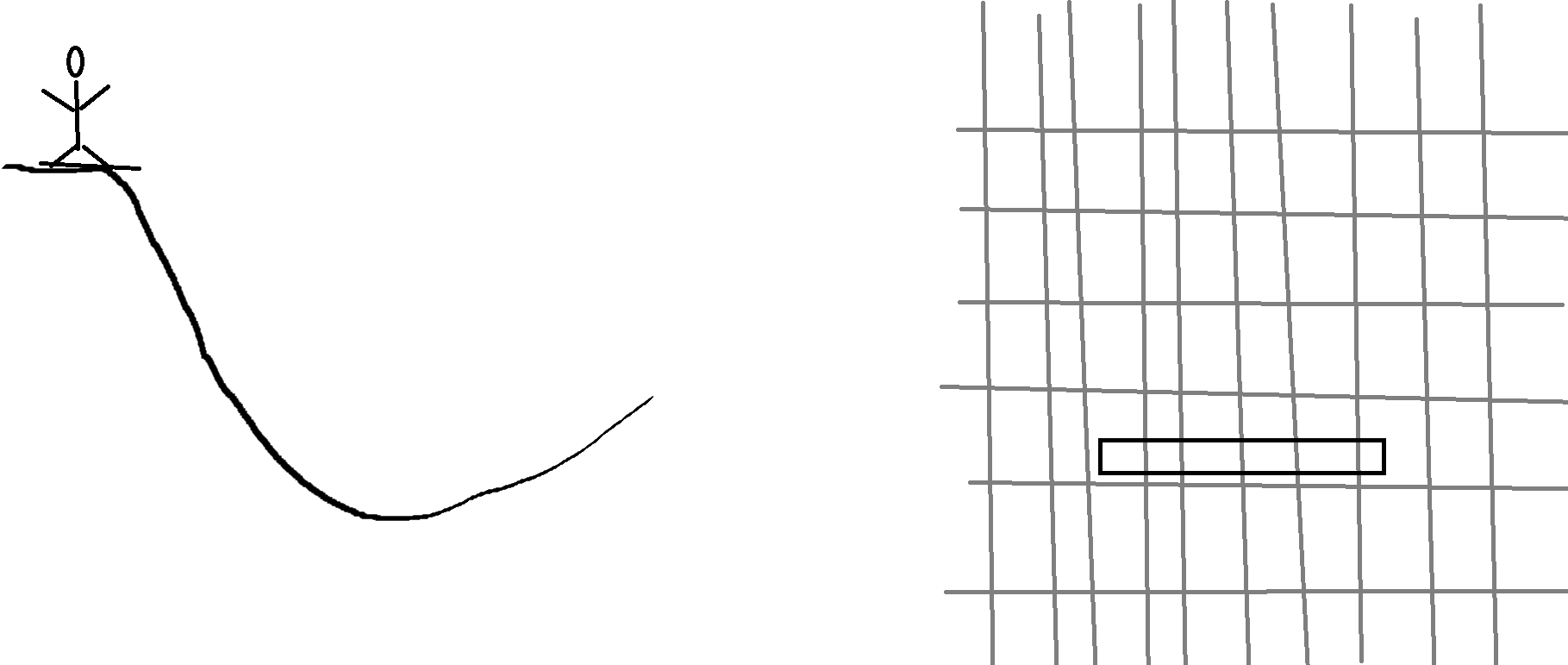
**420-KMA-JQ**

**Session Hiver 2018**

****

Interfaces :

Pour débuter, nous aurons une interface de menu qui permettra à l’utilisateur de choisir son niveau de jeu (si nous avons le temps d’en créer plusieurs). Ensuite, nous aurons une interface de jeu où un skieur dévalera une pente pour atterrir sur une plateforme. Un autre mode de jeux peut se retrouver dans le programme si le temps nous le permet. Au lieu de choisir le lieu d’atterrissage, le joueur devra faire la courbe que le skieur descendra. La position d’arrivé sera prédéterminée. Dans ce mode, le code serait relativement différent, puisque nous devrions faire passer une courbe par l’endroit que le joueur décide. La courbe se dessinera de la même manière, sauf que ce sera l’utilisateur qui décidera l’allure de celle-ci.



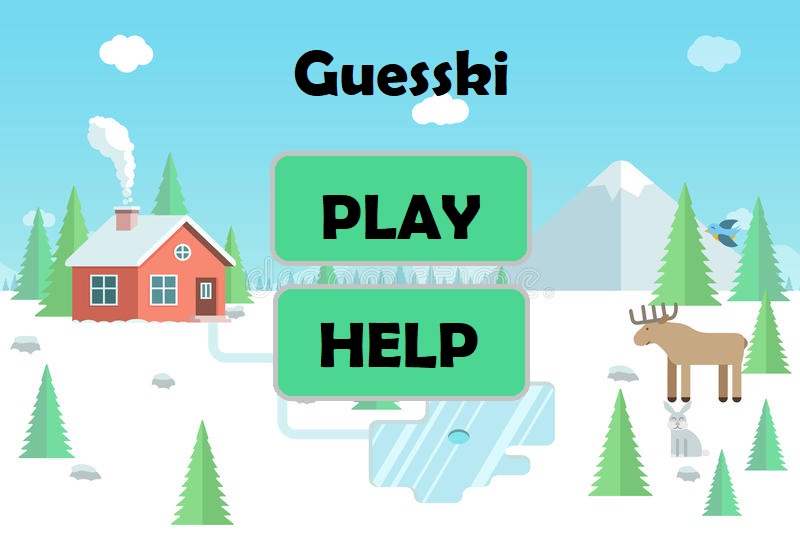
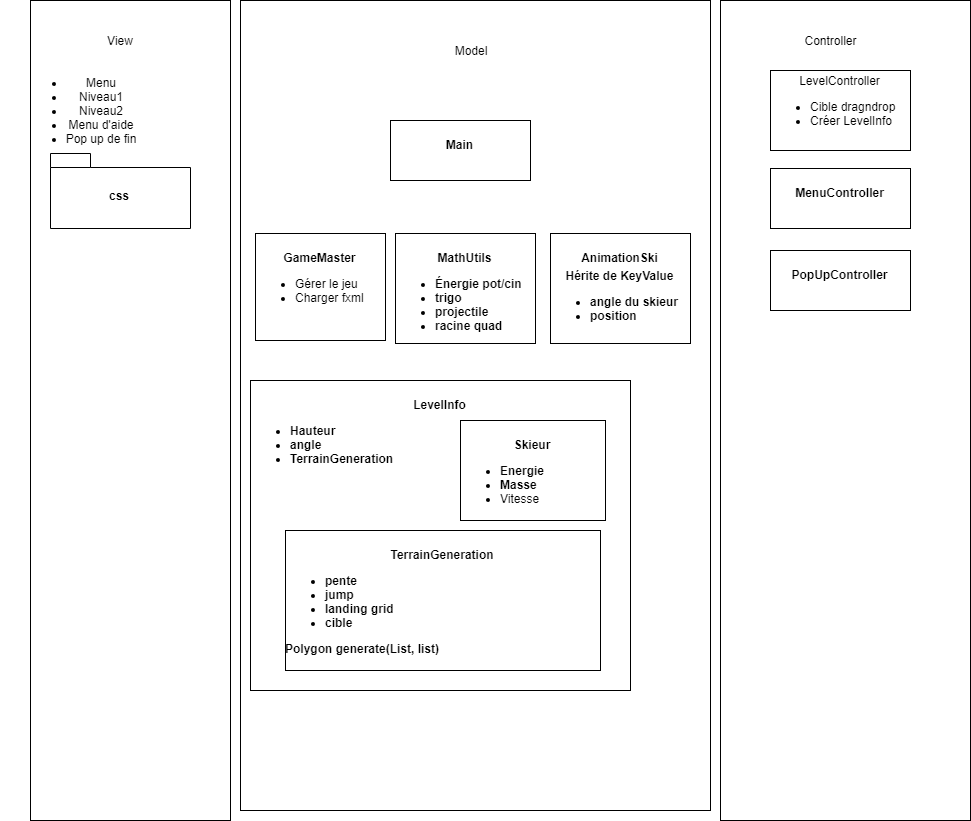


Diagramme de classes :

Nous utiliserons le modèle MVC pour structurer notre code.



Fonctionnement du code :

Notre jeu sera gérer par la classe GameMaster à laquelle on aura passé le stage lors de son instanciation dans le main. Le root de notre Stage sera un AnchorPane à lequel nous ajouterons en tant qu’enfant les fichiers fxml chargés.

GameMaster :

Le GameMaster s’occupera de charger les fichiers fxml dans la fenêtre de jeu. Lors de sa construction, il affichera le menu dans la fenêtre de jeu.

MenuController :

Cette classe permettra de faire apparaitre un menu d’aide et d’entrer dans un mode de jeu en appelant les méthodes de la classe GameMaster.

LevelController :

La classe LevelController sera le controller pour nos niveaux. Cette dernière s’occupera notamment des drags and drop. Cette classe contiendra une instance de LevelInfo. Elle s’occupera de l’affichage du terrain, du score ainsi qu’un bouton pour retourner au menu. Cette classe contiendra aussi un bouton play, qui démarrera l’animation et affichera un pop up à la fin de l’animation pour dire au joueur s’il a gagné ou pas.

LevelInfo :

La classe LevelInfo concerne tout ce qui est important de savoir à propos du niveau. Lors de la création d’une nouvelle instance de LevelInfo, un nouveau terrain sera générer selon une fonction mathématique. LevelInfo contiendra aussi une instance de Skieur dans lequel il on pourra retrouver la masse, l’énergie potentielle et cinétique ainsi que la vitesse du skieur.

TerrainGeneration :

La classe TerrainGeneration génèrera le terrain ainsi que les listes utiles pour l’animation du Skieur. Il sera possible de récupérer le terrain sous forme de Group afin de facilement l’ajouter à une Node.

Animation :

La classe animation servira à créer l’animation du skieur en déplaçant le skieur selon une liste de coordonnées de points appartenant à la pente ainsi qu’une liste contenant la distance entre chaque point. Ceci nous permettra de calculer à quel point déplacer le personnage selon sa vitesse ainsi que de déterminer l’angle d’inclinaison de la pente afin d’incliner le skieur selon la pente.

MathUtils :

Dans cette classe se trouvera plusieurs méthodes statiques que nous pourrons utiliser à travers notre code.

PopUpController :

Cette classe nous servira à contrôler les pops up qui s’afficheront notamment lors qu’un joueur réussit ou échoue à placer la cible correctement.

Échéancier :

|  |  |
| --- | --- |
| Semaine | Tâches |
| 4 (8 février) | Configurer git  Commencer GameMaster  Créer les interfaces |
| 5 (15 février) | LevelInfo/Skieur  MathUtils  Génération du terrain |
| 6 (22 février) | LevelController  Pop-up Controller  MenuController  Génération du terrain |
| 7 (1er mars) | Animations |
| 8 ((15 mars) | Animations  Controller nouveau niveau  Création de la pente |
| 9 (22 mars) | Génération de la plateforme  Détecter l’atterrissage  Gestion des pentes bizarres |
| 10 (29 mars) | Finaliser le 2e niveau |
| 11 (5 avril) | Finaliser détails  Affichage |
| 12 (12 avril) | Correction de bugs |
| 13 (19 avril) | Préparation de l’exposé |

Légende distribution des tâches :

Gabriel

Pascale

Michael

Nous avons prévu 3 semaines pour la finalisation et la correction de bugs dans notre programme, car, par expérience, nous savons que cela peut-être plus long que ce que nous envisageons. Cela nous laisse aussi un peu de temps pour des tâches qui pourraient prendre plus de temps que prévu.