INSA Lyon 26 avril 2022

Ponts

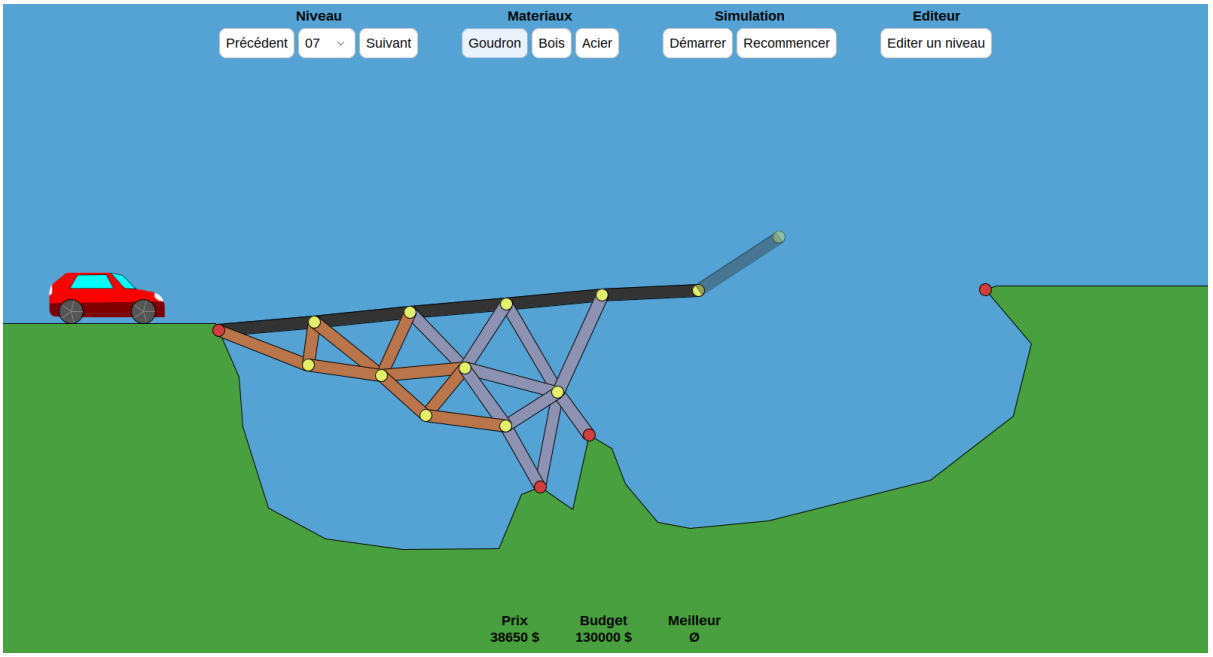
Par
Thomas BOYER
Rémy VAUDEY
Romain BENOIT
Groupe 45

Sommaire

I. Cahier des charges

L'objectif de notre projet est de créer un jeu de construction de ponts, et de tester leur robustesse en faisant passer un véhicule dessus. Le concept est tiré du jeu Poly Bridge, sur lequel nous nous sommes basés.

Ramené en 2D, un pont est constitué de barres reliées les unes aux autres, symbolisées par des segments comme visible sur la capture d'écran ci-dessous. La simulation se veut relativement réaliste, nous avons donc utilisé un moteur physique pour gérer les forces, les mouvements et les collisions entre les objets. Nous avons retenu pour cela la bibliothèque jbox2d. Cette gestion de la physique est l'objet de la partie scientifique du projet.



Fenêtre principale du jeu, avec un pont en cours de construction

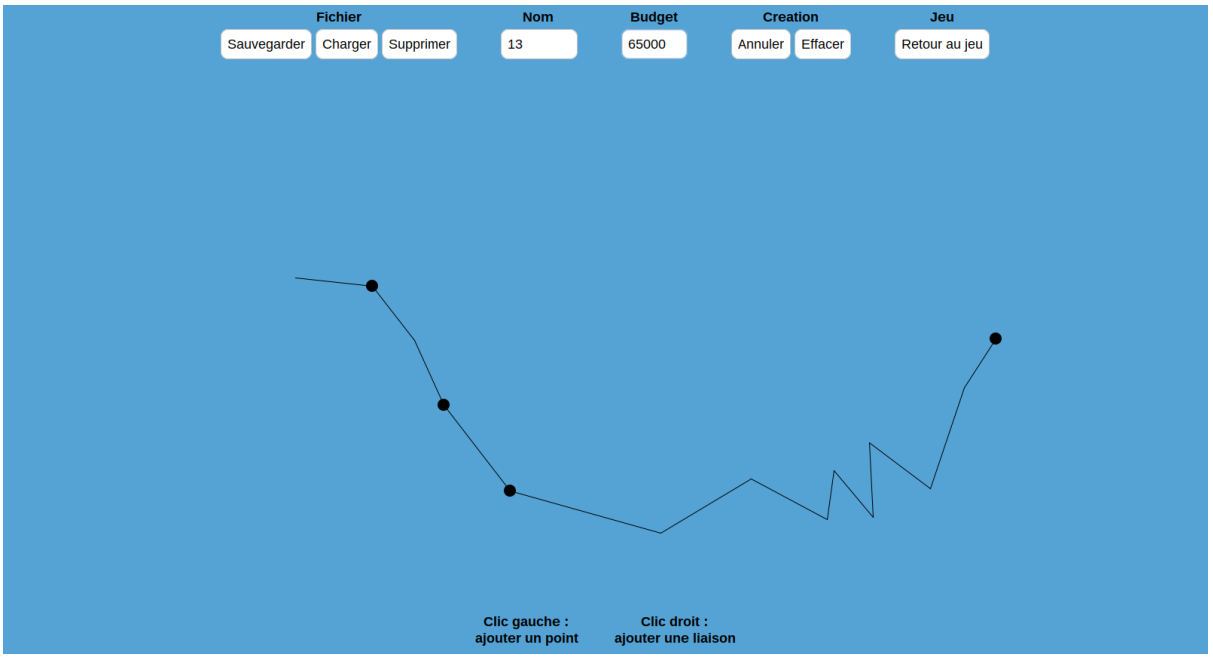
Quant à l'IHM, l'affichage est réalisé avec awt pour le dessin des éléments de jeu. L'interface comprend aussi des composants Swing tels que des boutons, des labels et un menu déroulant. Ils permettent de choisir le niveau, le matériau du pont ainsi que de contrôler la simulation.

En jeu, l'utilisateur peut créer une barre en cliquant sur une liaison, c'est-à-dire un point de fixation au sol (ronds rouges sur le schéma) ou à l'extrémité d'une barre déjà placée (ronds jaunes), puis en cliquant en un point quelconque pour définir l'autre extrémité de la barre, dans la limite d'une longueur maximale. Il peut également connecter deux liaisons déjà existantes pour former des structures complexes. Il doit choisir le matériau en fonction des situations : la voiture peut rouler seulement sur du goudron, l'acier est plus solide mais plus cher que le bois.

Une fois son pont construit, il peut lancer la simulation et voir si le pont tient debout. La voiture s'élance alors pour le mettre à l'épreuve. Le niveau est réussi si elle arrive de l'autre

côté du pont, et si le prix du pont est inférieur au budget. Le plus bas prix réalisé sur chaque niveau est gardé en mémoire et affiché.

Le jeu dispose aussi d'un éditeur de niveau qui permet de créer simplement l'environnement en plaçant des points et des liaisons. L'IHM permet de spécifier le nom et le budget, chaque niveau étant finalement enregistré dans un fichier.



Éditeur de niveau du jeu

II. Structuration du projet

Le projet est constitué de 23 classes représentées sur le diagramme UML page suivante. À noter que le logiciel que nous avons utilisé n'utilise pas les mêmes symbolismes de flèches que le cours. Les relations d'association sont représentées par des losanges pleins et les relations d'héritage et d'implémentations par des flèches vides. Nous n'avons pas représenté les relations de dépendance par souci de lisibilité.

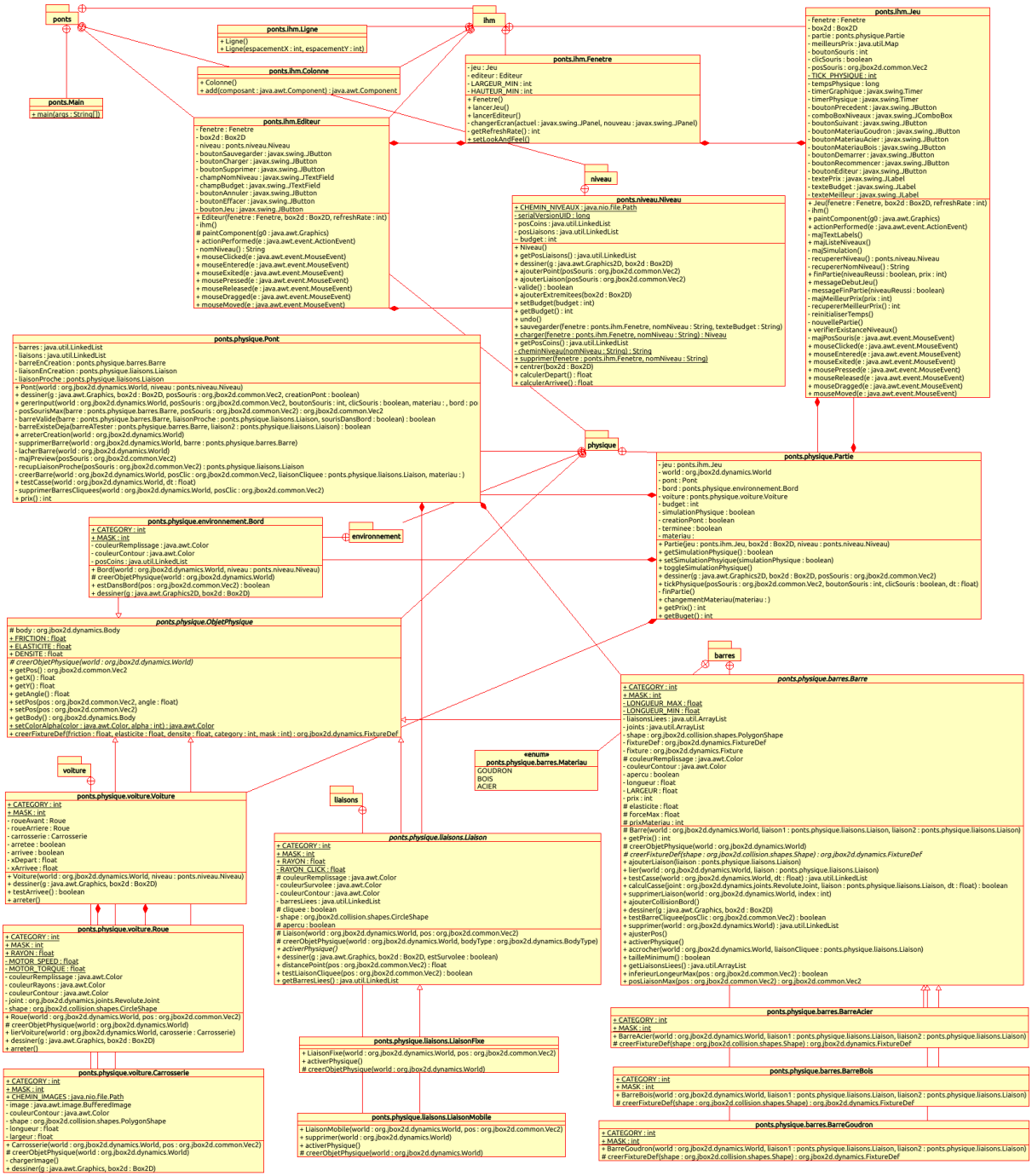
Le nombre important de classes nous a poussé à les ranger en sous-dossiers, correspondant à des packages Java. Ils sont symbolisés par des boîtes sur l'UML, reliées aux classes par des ronds barrés.

Le jeu est lancé dans la classe Main, qui crée une Fenêtre (JFrame) à laquelle on ajoute le Jeu (JPanel). Cet objet gère uniquement l'IHM ce qui représente déjà 500 lignes de code. La partie fonctionnelle du jeu est déléguée à l'objet Partie. À l'initialisation, la partie charge un objet Niveau stocké dans un fichier, qu'elle utilise pour initialiser toutes les classes associées à des éléments de jeu. L'objet Bord en particulier utilise la liste de positions des points pour créer un polygone qui représente l'environnement du niveau (en vert).

La partie crée aussi l'objet Pont, objet central dans le projet. Il s'apparente à un graphe non orienté, dont les arêtes sont des objets Barre et les nœuds des objets Liaisons. Le pont

stocke simplement des listes de ces deux objets, ce sont les objets eux-mêmes qui stockent dans des listes les autres objets auxquels ils sont connectés dans le graphe. La classe Barre est une classe abstraite dont hérite les barres des différents matériaux. Idem pour la classe Liaison qui donne lieu aux LiaisonFixe (en rouge) et LiaisonMobile (en jaune).

Le véhicule est associé à la classe Voiture, qui instancie deux objets de la classe Roue et un de la classe Carrosserie, qui sont eux dotés de physique. Plus généralement, tous les objets répondant à la physique de la bibliothèque jbox2d héritent de la classe ObjetPhysique. On y trouve entre autres la méthode abstraite creerObjetPhysique, qui initialise étape par étape les objets nécessaires à la librairie dans chacune des classes filles.



L'Editeur (JPanel) est une classe différente de Jeu qui vient remplacer ce dernier dans la Fenêtre. Il crée un objet Niveau qu'il met à jour en fonction des interactions de l'utilisateur, avant de le stocker dans un fichier grâce à l'implémentation de l'interface Serializable.

Enfin, il faut noter l'existence de la classe Box2d, un singleton initialisé au début du jeu gérant toutes les conversions de coordonnées. En effet, l'IHM utilise des coordonnées en pixels (int) tandis que la physique de jbox2d travaille avec des mètres (float), l'axe Y étant en plus inversé. Il faut donc pouvoir passer de l'un à l'autre en fonction de la taille de la fenêtre, qui dépend de l'écran et peut être redimensionnée à tout moment.

III. Problèmes et algorithmes

La partie algorithmique la plus élaborée de notre programme est l'ajout de barres par le joueur, qui présente de nombreux cas différents. Si lors d'un clic, si aucune barre n'est en cours de création, il faut vérifier si l'utilisateur a cliqué sur une liaison pour en créer une nouvelle. Si à l'inverse une barre est déjà en création, il faut s'assurer qu'elle est de la bonne taille ou sinon la réduire, qu'elle n'est pas dans la berge, qu'il n'existe pas déjà une barre reliant les deux liaisons, etc. Cette partie fait donc intervenir beaucoup de conditions.

Nous avons d'abord codé cet algorithme sans animation, c'est-à-dire qu'il faut attendre le deuxième clic pour que la barre s'affiche. Nous avons ensuite voulu rajouter la prévisualisation de la barre en construction, importante pour l'expérience utilisateur, mais qui nous a posé beaucoup de difficultés. En effet, on ne manipule plus du tout les objets de la même façon, puisque la barre est créée dès le premier clic pour pouvoir l'afficher. Notre première tentative de modification n'a mené qu'à des bugs incompréhensibles, liés au fait que les objets jbox2d sont censés être indéformables, or ils doivent changer de taille lors de la création. La solution pour laquelle nous avons finalement opté est d'initialiser la physique seulement à la fin de la création, avec la méthode 'activerPhysique'.

Après avoir compris le fonctionnement de la librairie de physique, il nous a fallu rajouter un comportement clé pour notre jeu qui n'est pas prévu : la possibilité de casser une liaison entre deux objets. La seule information intéressante à laquelle nous avons accès est la force de réaction d'un objet par rapport à un autre, via la méthode 'joint.getReactionForce'. Nous l'utilisons en comparant la projection de cette force dans la direction orthogonale à la barre par rapport à une valeur maximale dépendant du matériau, algorithme qui donne les résultats les plus convaincants.

La voiture s'est aussi révélée compliquée à implémenter. Nous l'avons modélisé par trois classes mentionnées dans la partie précédente (Voiture, Carrosserie, Roue). Pour qu'elle ne soit pas qu'un simple rectangle, nous avons cherché à utiliser une image que nous avons dessinée. Mais la taille n'est pas forcément bonne, et surtout la voiture change d'orientation, on ne peut donc pas simplement dessiner l'image telle quelle. Nous avons donc utilisé dans notre algorithme une 'BufferedImage' sur laquelle nous faisons des opérations de mise à l'échelle et de rotation. Bien qu'elle ne soit pas dessinée comme les autres éléments de jeu, la voiture doit respecter la même physique, nous avons donc essayé de faire correspondre sa 'hitbox' rectangulaire au mieux pour que les collisions ne paraissent pas étranges. Enfin,

pour qu'elle puisse rouler, nous avons utilisé une fonction de moteur des liaisons entre les roues et la carrosserie, ce qui leur applique un couple dont on peut régler la norme.

Cela nous amène au problème des collisions : tous les éléments de jeux n'interagissent pas entre eux. La voiture peut traverser le bois et l'acier, pour permettre au joueur de construire des structures porteuses au-dessus du pont. Moins évident, les barres n'interagissent pas entre elles, car on observe sinon des comportements très étranges liés à la superposition des coins. Nous avons utilisé pour cela des 'bitmasks' : un bit correspond à une couche de collision. On définit ainsi pour chaque objet physique dans quelles couches il interagit, via les constantes CATEGORY et MASK qui sont utilisées pour définir un filtre.

IV. Bibliographie

Le plus gros défi du projet a été d'apprendre à maîtriser la librairie gérant la physique, jbox2d. Elle n'est plus activement développée et n'a pas de documentation officielle. Nous l'avons d'abord trouvé sur le dépôt GitHub puis sous forme de jar sur un autre site qui comprend la javadoc de la librairie, ce qui nous a été très utile.

JBox2d est en fait un port Java de la libraire Box2D en C++, nous nous sommes donc appuyés sur la documentation de cette dernière en traduisant les fonctions. En première introduction aux concepts de Box2D, la série de vidéos de 'The Coding Train' sur le sujet nous a été d'une aide précieuse.

Tout au long du projet, nous avons fait un usage important de Stack Overflow pour trouver des réponses à nos problèmes. Pour des fonctions d'IHM avancées en particulier, nous avons trouvé des questions sur la rotation et la mise à l'échelle d'image (pour la voiture).

Pour en apprendre davantage sur les fonctions de Swing et awt, telles que les Layout Manager, permettant le placement automatique des boutons sans avoir à spécifier des coordonnées, nous avons eu recours à la documentation officielle de Java par Oracle. Enfin, pour rendre l'IHM plus moderne, nous avons utilisé la librairie FlatLaf, très bien documentée.

V. Améliorations possibles

Tout d'abord, nous pourrions chercher à améliorer la physique de notre jeu car après de nombreux essais, on remarque qu'elle n'est pas tout à fait réaliste. En effet, il est possible que des ponts censés être résistants cassent mais qu'à l'inverse certains ponts censés se briser tiennent le choc. La librairie a cependant ses limites, car elle nous donne accès qu'à peu d'informations sur son fonctionnement interne.

Une simulation réaliste pourrait aussi être obtenue en ajustant tous les paramètres des objets physiques en fonction des matériaux (masse volumique, élasticité, frottements) en faisant de nombreux essais ou en cherchant des données réalistes.

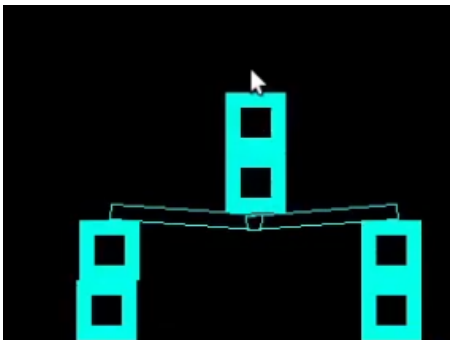
Une fois la physique parfaitement réglée, nous pourrions essayer d'informer le joueur sur la résistivité en temps réel de chacune des liaisons durant la simulation. Les liaisons seraient

initialement vertes, mais elles seraient proches de la rupture, plus leur couleur se rapprocherait du rouge.

De plus, nous pourrions ajouter de nouveaux matériaux de construction ou de nouveaux objets utilisés dans le génie civil afin de rendre plus réaliste notre simulation. La librairie permet par exemple de faire des cordes, que l'on pourrait utiliser comme câbles. On pourrait aussi offrir au joueur la possibilité de choisir différent type de véhicule avec chacun des poids ou vitesses différentes et ainsi observer pour quels véhicules le pont est fiable.

Enfin, des améliorations visant à améliorer l'expérience de jeu pourraient être implémentées. On pourrait ajouter des raccourcis claviers pour faire certaines actions, permettre le déplacement de liaisons déjà placées, ou encore greffer un système de copier/coller, permettant de gagner du temps lors de la construction d'un pont symétrique par exemple.

VI. Échéancier et répartition du travail



Voici un résumé des étapes du projet, réalisées au fur et à mesures des séances :

- Compréhension de la librairie avec création de boîtes et de poutres pour évaluer la viabilité du sujet (ci-contre).
- Création des liaisons et des barres que l'on utilisera dans le but de faire le pont.
- Création des "liaisons fixes", points d'attache à partir desquels on pourra créer le pont.
- Implémentation des boutons pour lancer/arrêter la physique, changer de matériau.
- Ajout laborieux de la prévisualisation des barres lors de leur création.
- Ajout de la possibilité de supprimer des barres par un clic droit.
- Implémentation du décors et des bords, avec leurs collisions.
- Création de la voiture initialement représentée par un rectangle et deux disques.
- Création de l'objet niveau enregistrable dans un fichier.
- Implémentation de l'éditeur permettant de créer des niveaux.
- Ajout de l'image de la voiture.
- Implémentation de la notion de prix et calcul du prix d'un pont.
- Création du point de départ et d'arrivée.
- Adaptation de la taille de la fenêtre à la taille de l'écran.
- Finalisation des interfaces et affichage de messages pendant le jeu.

Romain s'est occupé de la première phase de test, puis rejoint par Thomas pour la création du pont. Les tâches se sont vite réparties entre les membres. Rémy et Thomas ont travaillé sur la voiture et la gestion de l'image tandis que Romain à fait les niveaux avec l'éditeur.

Nous avons travaillé avec git pour synchroniser les fichiers, via le dépôt GitHub du projet. Pour ce qui est des pourcentages, nous souhaiterions répartir équitablement entre les trois membres de l'équipe.