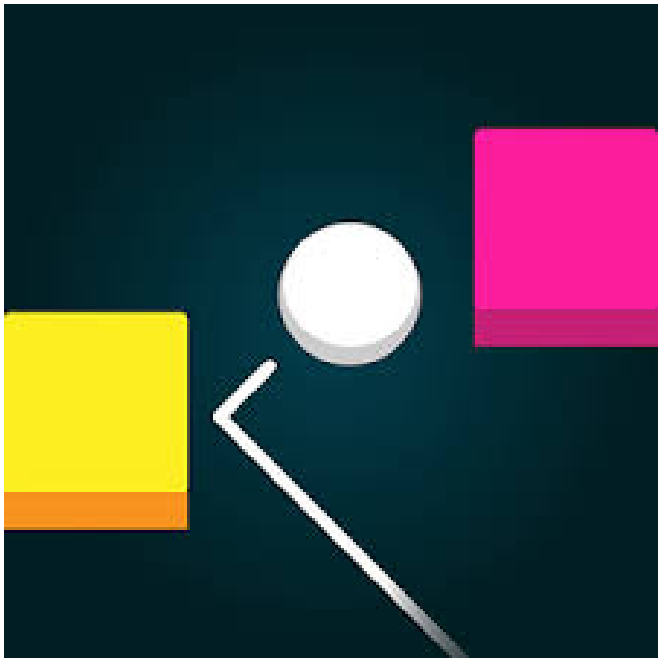
Rapport de projet
bac + 4
Analyste Développeur

Pong Breaker

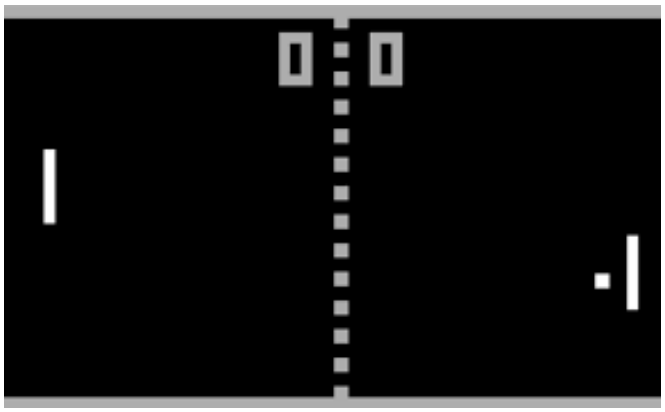


Axel Bouvier
François-Xavier Arthaut
Theo Nardin
Axel Pericart

Sommaire

Introduction

Le nom de notre projet de création d'application ludique intitulé Pong Breaker s’inspire de deux anciens jeux vidéo présents sur le marché depuis plusieurs décennies.



Pong

Il s’agit d’un jeu vidéo d’arcade qui fut sorti en 1972 et s’illustrant par la même occasion comme le premier jeu de ce genre existant dans la catégorie sport. Celui-ci s’inspire du tennis de table comme décrit ci-dessous.

Pong est définit comme un programme fonctionnant dans les conditions suivantes :

- deux joueurs obligatoires occupant chacun une moitié de terrain (la partie droite et la partie gauche)

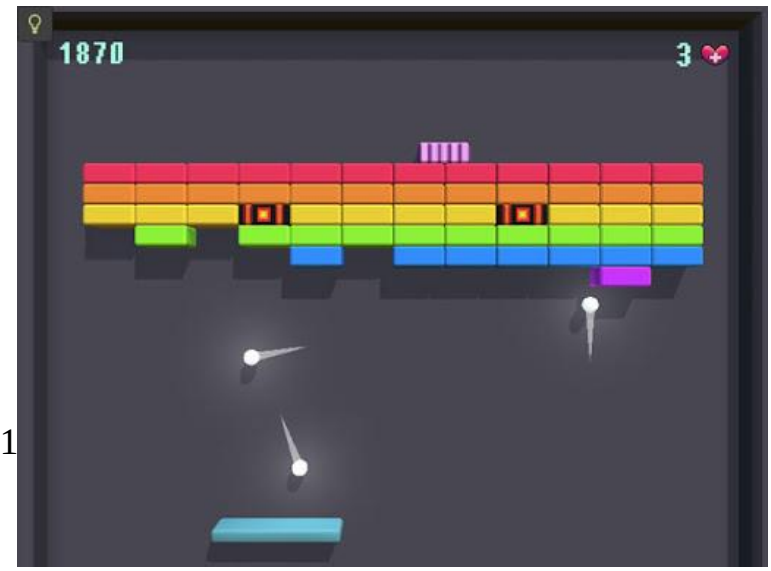
- deux raquettes déplaçables de haut en bas pour renvoyer la balle dans le camp adverse

- une balle dont la vitesse accélère au fil du jeu et qui doit être réceptionnée et redirigée par chaque joueur vers le camp adverse (sinon le point est remporté par le joueur de ce camp adverse)

Ce premier jeu servira d’exemple pour celui présenté ci-après.

Breakout

Breakout est également un jeu vidéo arcade sorti en 1976 qui fut le tout



premier à être commercialisé comme étant un jeu casse-brique.

Il est jouable sous les conditions suivantes :

- un seul joueur suffit pour tester ce jeu
- une seule palette présente sur la partie basse de la plateforme qui a pour but de renvoyer la balle en détruisant chacune des briques disposées horizontalement

I) Besoins fonctionnels et techniques

a- Cahier des charges

Type de programme

Un jeu borne d'arcade composé de deux joueurs pouvant se déplacer où ils le souhaitent sur chaque moitié de terrain.

Les deux participants du jeu se distinguent par leur couleur respective (bleu et rouge) à travers leur palette et leurs briques. Une balle sert à marquer des points et augmenter son score pour chaque tour de jeu.

Objectif du programme

Se frayer un chemin dans la moitié de terrain adverse avec la balle tout en s'aidant de la palette dans le but de gagner un maximum de points.

Des briques sont érigées pour assurer une défense solide de son camp en se protégeant efficacement des attaques au niveau de la zone de but lorsque le joueur adverse contrôle la balle.

b- Solution trouvée pour la conception du logiciel

Notre programme est fonctionnel grâce à l'introduction des éléments ci-après :

- Deux joueurs défendant chacun leur moitié du terrain sur lequel la scène prend place de la même façon que dans Pong

- Deux raquettes mobiles utilisables par les joueurs durant toute la partie utilisées pour contrôler la direction dans laquelle la balle sera repoussée vers le terrain de l'adversaire tout comme dans pong

- Présence de briques (comme dans Breakout) qui seront cette fois-ci agencées de manière verticale et protégeront chaque camp

- 3 types de balles :

  - **Classique** : c'est la balle qui sera utilisée pour progresser dans le camp opposé jusqu'à la zone de but pour marquer des points.

    Ce type de balle était présent à la fois au sein du jeu Pong mais aussi Breakout.

  - **Destructrice** : un autre type de balle qui aura pour seul but de détruire les briques et la raquette du camp opposée, ce qui a pour conséquence d'affaiblir la défense de ce dernier.

    Cette balle a la même capacité de destruction de briques que la seule balle utilisée dans Breakout.

  - **Créatrice** : le dernier des trois types de balles qui à l'inverse de la précédente aura la capacité de créer des briques pour défendre le camp du joueur qui s'en sert

II) Gestion de l'avancement du projet

a- Les contraintes de développement

Pour mener le développement de l'application à bien, nous avons dû composer avec les contraintes imposées par le client :

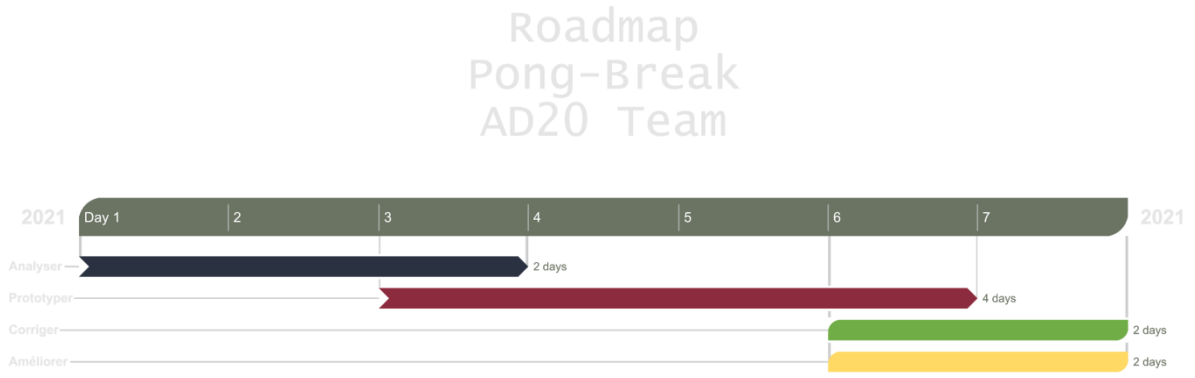
- Une semaine de développement pour avoir un prototype fonctionnel
- L'utilisation d'un système de gestion des collisions
- Une interface graphique en deux dimensions
- Un diagramme de classes modélisant les objets et interactions

Pour mener à bien cette mission, nous avions quatre développeurs disponibles en temps partiel, correspondant à 4 jours/ homme.

Une fois le sujet choisi, nous avons estimé l'efficacité de différentes méthodes de travail applicables.

b - Le plan de travail:

Pour répondre aux demandes du cahier des charges nous avons choisi d'établir un calendrier à respecter, à l'issu duquel notre projet serait présentable. Dans un premier temps notre objectif était de réfléchir aux aspects techniques de notre



projet: Quelles technologies ? Quels outils ? Quelles méthodes ? Comment répartir le travail ?

De là nous avons pu commencer à penser plus en profondeur le projet : quel découpage dans notre architecture logicielle ? Quels classes et objets utiliser ?

Pour répondre à ces questions nous avons pris le temps d'analyser, durant les deux premiers jours, toutes les options disponibles. Nous avons fourni le diagramme des classes à l'issu de ces deux premiers jours.

Nous en avons profité pour mettre à jour les environnements de développement et pris les décisions aux questions laissées en suspens.

Du troisième (jour) au cinquième jour, nous avons mis à exécution la construction du programme, en accord avec le diagramme de classe établi.

Enfin, durant le week-end nous avons terminé le développement et corrigé les derniers problèmes.

c - La méthode de travail :

Pour atteindre une efficacité maximum et minimiser les pertes de temps nous avons choisi de suivre la méthode dite d'extreme programming. De cette manière nous nous sommes retrouvé agiles avec une grande réactivité.

Théo Nardin a joué le rôle de Lead-Dev et de "scrum master" durant la période développement.

Nous avons privilégié la communication et n'avons pas hésité à user du pair programming pour maximiser notre rendement.

Dans un premier temps, nous avons choisi de découper le projet en features afin de faciliter la distribution du travail, puis nous avons engagé les implémentations.

Nos cycles de développement se déroulaient en suivant les étapes suivantes :

- estimation/planification -> implémentation -> tests

Si le test ne s'avérait pas concluant, alors nous recommencions l'implémentation.

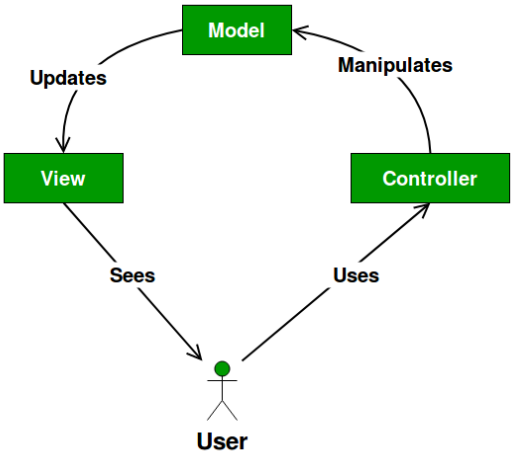
Nous avons choisi d'utiliser Slack comme plateforme de communication et de partage de ressources, GitHub comme hébergeur de code et IntelliJ IDEA comme logiciel de développement. Nous avons choisi d'utiliser Java car c'est un langage que tous les développeurs de l'équipe ont étudié et maîtrisent.

Associé à Java nous avons choisi d'utiliser JavaFx, une bibliothèque graphique de Java simple d'utilisation et répondant parfaitement aux besoins du projet.

III) L'architecture logicielle:

a - Le design pattern MVC

Nous avons choisi de développer notre programme autour de l'architecture MVC : Modèle, Vue et Contrôleur sur le squelette d'un autre programme utilisé lors d'un cas pratique d'étude, le moteur Alpha de Bin-Minh Bui-Xuan.



Dans notre cas, l'utilisateur est l'élément qui fournit les entrées au programme.

Le contrôleur reçoit et interprète les changements qui sont répercutés sur les données et objets stockés dans le modèle.

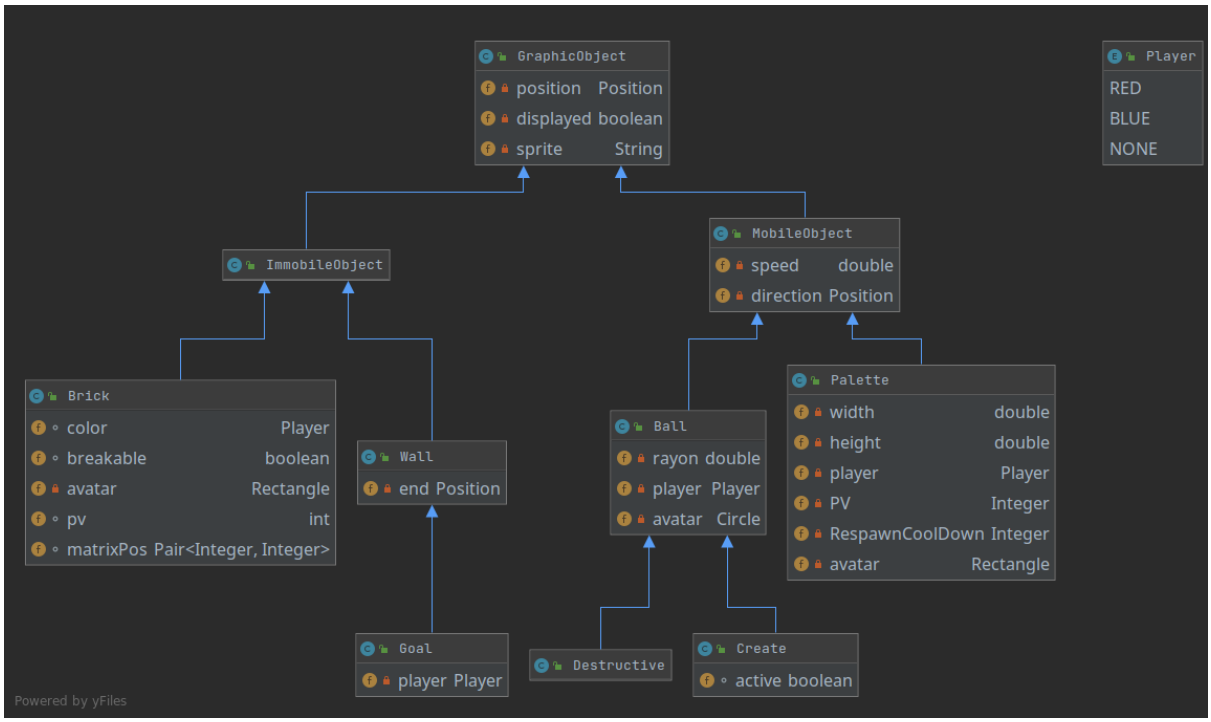
La vue va lire les données du modèle et fournir l'affichage correspondant.

La vue est une classe qui centralise les appels à la librairie JavaFx et s'occupe de dessiner à l'écran les éléments nécessaires au bon fonctionnement du jeu.

b - Modèle

Le modèle de notre application contient les classes de tous les objets nécessaires au bon déroulement du jeu. On y retrouve les palettes, les balles, les buts et les murs.

Chaque objet se caractérise par sa position et sa représentation graphique. De cette manière, à l'appel de chaque objet, toutes ses caractéristiques sont aisément accessibles.



Les attributs tels que la vitesse et la position des objets sont stockés dans le modèle.

L'update des positions: Au début de chaque partie le contrôleur donne une impulsion initiale à la balle, correspondant à une direction, représentée par deux nombres x et y, ainsi qu'une vitesse. Ces paramètres sont donc set par le contrôleur dans le modèle grâce aux méthodes que ce dernier met à la disposition du second.

En fonction de sa position courante associée au vecteur de direction, le modèle de la balle calcule la nouvelle position de l'objet et la stocke.

Pour calculer la position, on additionne les valeurs correspondant à la direction qu'on aura au préalable multipliées par la vitesse à la position courante, ce qui en "téléportant" l'objet de proche en proche va simuler un effet de mouvement et de vitesse.

Ce calcul est appliqué à tous les objets mobiles présents dans le programme

Elle sera lue plus tard par la vue, puis mise à jour à nouveau.

L'update de la direction : il est effectué à chaque collision avec un objet, il est donc déclenché par le contrôleur mais les calculs sont effectués dans le modèle.

Quand il s'agit d'une palette, cette dernière va alors transmettre son vecteur de direction à la balle pour simuler un rebond et transmettre à la balle la direction souhaitée. Lorsqu'il s'agit d'une brique le vecteur de direction de la balle est inversé de façon a compliquer la tâche du tireur n'ayant pas pris soin au préalable de détruire les briques adverses. Lorsqu'il s'agit d'une collision avec un des murs seul la direction y est inversée pour que la balle continue sur sa trajectoire horizontale mais en inversant sa trajectoire verticale. Il est à noter que ces considérations ne sont à porter que sur la balle principale. En effet les balles de créations ou de destruction n'ont pas de réels effets de rebond. Elles se déplacent de manière horizontale et disparaissent en atteignant leur cible.

Le modèle fourni les entrées de la vue et est actualisé par les entrées de l'utilisateur capté par le contrôleur.

c - Vue

La vue est représentée par une classe View dont le travail est d'interagir avec la bibliothèque JavaFx, et d'afficher les objets dans la fenêtre.

Pour créer les objets nous avons choisi de faire appel à une factory, une classe dont le but est d'instancier les images utilisées dans le jeu. Cela nous permet de

donner des objets à la factory et d'avoir en retour les images correspondantes. Nous avons mis en place un système permettant de reconnaître le type des instances fournies en entrée et qui renvoie l'image correspondante en sortie, la factory gère alors entièrement la création des objets.

Pour afficher le mouvement, les objets sont redessinés à chaque tick en fonction de leurs nouvelles positions.

d - Contrôleur

Le contrôleur est le module de l'architecture MVC qui se charge de prendre les entrées utilisateurs et de modifier les informations contenues dans le modèle et la vue.

Notre contrôleur est présent dans le package *engine* de notre projet, on y trouve le code source de la classe *engine* et de la classe *collisions.*

La classe *engine* en elle-même va gérer directement les entrées utilisateur récupérées dans le main. Ces entrées ne concernent qu'une poignée des objets présents dans notre jeu : La palette, les balles créatrices et destructrices.

Les joueurs peuvent donc diriger la palette pour faire rebondir la balle et tenter de marquer dans le camp adverse. Le joueur peut aussi choisir de lancer les balles créatrices et destructrices et choisir quand la première s'active de façon à construire une brique à l'emplacement choisi.

La captation des contrôles se fait en trois temps :

- Le signal d'enclenchement d'une touche passe par le main.
- Le main va ensuite appeler dans le contrôleur une méthode permettant d'activer un booléen.
- A chaque itération du moteur les booléens sont vérifiés
- En fonction des booléens actifs, le contrôleur va effectuer les actions correspondantes.

Et quand une touche est relâchée le même processus s'effectue mais va chercher à désactiver le booléen de façon à stopper l'action.

A chaque itération du moteur les collisions sont aussi vérifiées grâce aux méthodes mises à disposition via la classe *Collisions.* Ces méthodes vont modifier directement les données présentes dans le modèle.

Pour déterminer si une collision entre les différentes balles et les palettes ou les briques a lieu nous utilisons les méthodes présentes dans le package *tools.PBUtils*:

- *intersectCercle :* Cette méthode permet de savoir si une droite (AB) coupe le diamètre d'un cercle. Pour ce faire une droite perpendiculaire passant par (AB) et par le centre du cercle est tracée et le point d'intersection entre cette droite et (AB), C est récupéré. Ensuite on

compare la distance entre C et le centre du cercle et si c’est inférieur au rayon du cercle et que C est sur le segment [AB] alors une collision est détectée.

- *pointInRectangle* : À l'aide du produit scalaire, détermine si un point est à l’intérieur d’un rectangle

Parfois il est adjoint à ces tests un contrôle permettant de vérifier que le propriétaire de ces objets est différent. Par exemple, il n’y a pas de collision entre les balles tirées depuis la palette bleue et les briques bleues.

Dans le contrôleur, il est aussi possible de trouver la méthode qui va se charger de créer les briques en évitant les chevauchements. Pour cela l’espace où la création est autorisée, entre x = 320 et x = 960, a été divisé en plusieurs cases dans une matrice associée de 8*8. Si une brique existe dans une case alors la valeur de cette case vaut 1, sinon 0 et on ne peut créer de brique que si la case est nulle.

Conclusion

Malgré le court laps de temps qui nous a été donné pour concrétiser ce projet, l'ensemble de l'équipe s’est harmonisée au mieux pour décider collectivement du rôle de chacun dans la réalisation des tâches.

La distribution des tâches a été possible grâce à l’utilisation d’outils de communication et de partage de données indispensables tel que slack puis github. L’application de notre méthode de travail groupé, connue sous le nom d’extrem programming a aussi grandement contribué à une juste répartition des petits objectifs à court terme du projet.

L’emploi de cette méthode a entre autres impliqué des pratiques de code optimales tel que le Pair Programming afin d'optimiser en terme de temps l’acheminement des tâches. En effet, les personnes ayant les meilleures connaissances de gestion du code apportent leurs conseils à celles qui ont moins d’aisance pour cela. De même en ce qui concerne l’emploi de git.

Extreme Programming nous a aussi poussés à changer régulièrement la structure du projet en supprimant certaines méthodes ou classes devenant inutiles après quelques feedback entre membres de l'équipe et tests fonctionnels.

Avec plus de temps nous pourrions ajouter des bonus, un menu ou la possibilité d’avoir plusieurs terrains.