INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

BREAKOUT PROJECT

par

Gaige Théo, Banks Jeremy,

Minot Valentin, Nagel Margaux

groupe PCC2A – 55

Villeurbanne, le 30 Avril 2022

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Table des matières

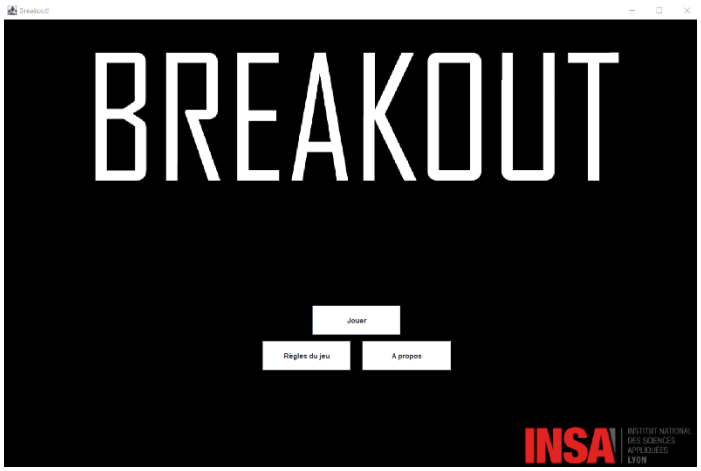
1. Contexte du projet et objectifs

Le projet que nous avons développé s'inspire et reproduit le jeu d'arcade « Breakout » commercialisé par Atari Inc. en 1976. Ce jeu a été développé dans le cadre du projet de fin d'année du cours d'Algorithmique et programmation dispensée en 1ère année et 2ème année de FIMI à l'INSA Lyon. Il utilise les notions d'algorithmique et programmation en Java découvertes tout au long de l'année mais également d'autres notions vues dans un cadre extrascolaire. Celui-ci a été conçu pendant les 2h hebdomadaires de travaux dirigés d'algorithmique mais aussi en dehors de ces créneaux dédiés par les différents membres du groupe. Nous voulions principalement faire ce projet car c'était un jeu qui nous était tous familier. Par ailleurs, l'un de nos objectifs était aussi de rendre l'interface plus moderne, avec plus de ressemblances aux jeux que l'on pourrait croiser aujourd'hui. Nous nous sommes inspirés de l'interface ci-dessous.

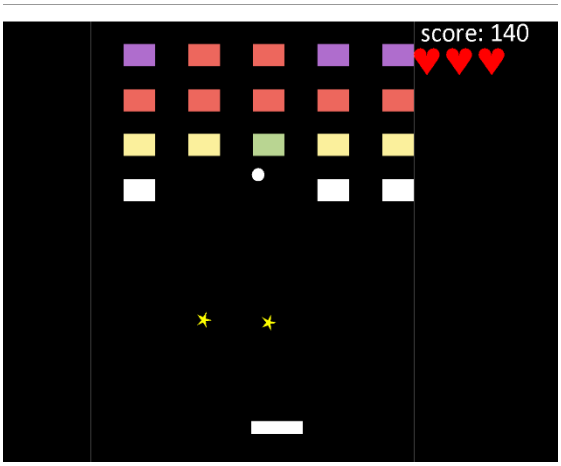
Le but du jeu est simple, il faut casser toutes les briques disposées en haut de l'écran à l'aide d'une balle rebondissant sur une plateforme amovible, dirigée par l'utilisateur. La complexité du jeu pourra être évolutive avec un système de niveau en fonction de l'avancement du joueur dans la partie. Chaque niveau comportera une disposition plus ou moins différente afin d'augmenter la complexité. Les règles du jeu devront être accessibles par l'utilisateur avant le lancement du jeu. Par ailleurs, chaque brique peut avoir un niveau de vie différent compris en 1 et 8. Si la balle touche une fois la brique, sa vie diminue. L'interaction joueur-interface pourra se faire à l'aide de la souris, permettant de diriger la plateforme. Le joueur aura plusieurs vies au sein d'une même partie correspondant au nombre de balles pouvant être régénérées. Par conséquent, si la balle tombe une seconde sera recréée, accompagnée d'effets sonores Un nombre fini de balle sera généré, le joueur n'aura pas un nombre de vie illimité. S'il ne parvient pas à récupérer la dernière balle, alors il a perdu. Un écran de fin doit s'afficher. Dans le cas contraire, si celui-ci parvient à casser toutes les briques de tous les niveaux, alors il a gagné. Dans ce cas-là, un écran de victoire doit s'afficher. Un suivi du nombre de vie et du score de la partie sera aussi réalisé. D'éventuelles fonctionnalités peuvent être ajoutées comme un changement de la vitesse de la balle par exemple

2. Description de l'interface graphique

Une fois la compilation du code exécutée, la 1ère fenêtre que l'utilisateur peut apercevoir est celle située en bas à gauche. Celui-ci à le choix et peut cliquer sur 3 boutons différents : « jouer », « règles du jeu » ou bien à « à propos ».



Capture d'écran du Menu



Capture d'écran du jeu en cours d'exécution

Une fois qu'il clique sur le bouton « jouer », le jeu se lance. Une balle est générée automatiquement au milieu de l'écran et tombe sur la plateforme mobile dirigée par la souris du joueur. Les murs que nous pouvons apercevoir sur l'image de droite délimitent la zone de jeu. Chaque brique peut contenir un nombre de points de vie différents, le maximum étant 8 points de vie. Une brique est donc caractérisée par son nombre de points de vie, grâce à cela une couleur lui est associée. Par conséquent, à chaque fois que la balle touche une brique, celle-ci perd de la vie et prend la même couleur que les briques du même point de vie. Les briques dépourvus de vies n'existent plus. Le score affiché sur l'image de droite augmente de 20 à chaque fois qu'une brique est touchée. Nous avons aussi instauré un système de vie, visible par le jouer à l'aide des 3 cœurs. Chacun de ces cœurs correspond au nombre de balles qu'il peut rejouer si celui-ci ne les récupère pas et les fait tomber.

Dès que toutes les briques du niveau ont été cassées et que le joueur n'a pas perdu la partie, c'est-à-dire, qu'il lui reste au moins une balle, alors un nouvel agencement de briques est généré. Quant au score, il ne cesse d'augmenter. Tout au long du jeu, des bonus sous forme d'étoiles peuvent apparaître aléatoirement lorsque la balle casse une brique. Ceux-ci peuvent, aléatoirement, avoir les effets suivants:

- Agrandir la taille de la plateforme sur laquelle rebondit la balle
- Agrandir la taille d'une balle, ce qui facilite son interception par la plateforme du joueur
- Accorder au joueur une vie supplémentaire
- Générer une autre balle dans le jeu, ce qui permet au joueur de casser plus rapidement les briques (les balles n'ont aucune interaction entre elles).

La balle émet un son lorsqu'elle rebondit sur n'importe quel élément du jeu et lorsqu'elle tombe plus bas que la plateforme. Un son est aussi émis lorsque le joueur récupère un bonus.

Si malheureusement la dernière balle tombe avant que le joueur ne puisse finir tous les niveaux alors la partie est perdue. Dans le cas contraire, si le joueur a réussi à accomplir tous les niveaux alors il a gagné. Voici ci-dessous les 2 images laissant savoir à l'utilisateur s'il a perdu (à gauche) ou gagné (à droite).

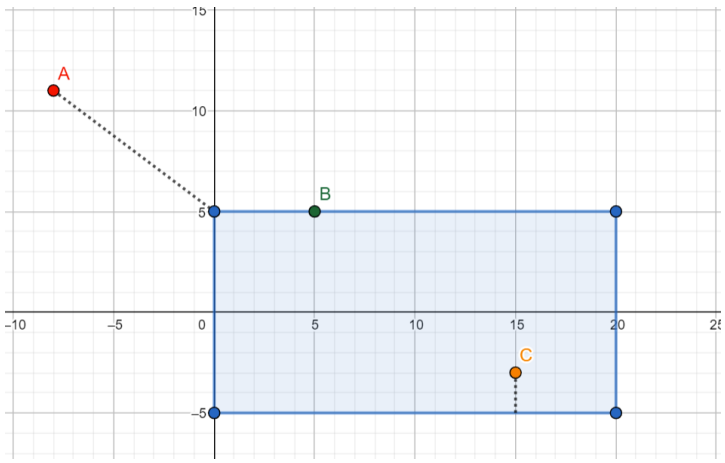
Image de défaite

Image de victoire

3. Description de l'algorithme

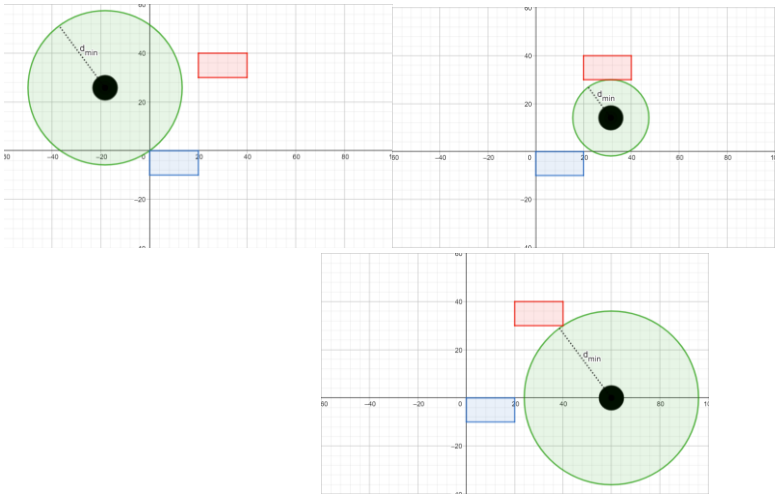
L'un des points importants de notre algorithme est le mouvement de la balle et notamment les collisions avec les différents éléments du jeu. Nous nous sommes inspirés d'une méthode de ray marching appelée *sphere tracing*.

L'algorithme que nous utilisons repose sur les fonctions distance signée. Ces fonctions renvoient la distance signée d'un point à une forme. La distance signée est la longueur du segment le plus court entre le point et la surface de la forme, si le point est à l'extérieur de la forme la distance est positive, si il est a l'intérieur la distance est négative.



Sur ce dessin la distance signée entre A et le rectangle est de 10, pour B elle est de 0 et pour C elle est de -2.

En prenant le minimum des distances signées à tous les obstacles, on obtient une distance de laquelle on peut faire avancer la balle sans risque de collision peu importe la direction.



Notre jeu fonctionne à l'aide d'un timer, ainsi nous faisons les calculs de mouvement puis nous ré-affichons la fenêtre à intervalles réguliers. Nous avons défini une vitesse pour chaque balle et, à l'aide de l'intervalle de temps, on peut donc calculer la distance que doit parcourir la balle entre deux frames. Ensuite tant qu'il reste de la distance à parcourir:

- On cherche l'obstacle le plus proche de la balle à l'aide de fonctions distance signée.
    - Si la distance signée entre l'obstacle et la balle est négative ou nulle, cela signifie qu'il y a collision avec l'objet on va donc modifier la direction de la balle pour simuler un rebond puis on la fait avancer de l'incrément minimal.
    - Si la distance signée entre l'obstacle et la balle est positive, on fait avancer la balle.
        - Si la distance est plus grande que la distance restante à parcourir on fait avancer la balle de la distance restante à parcourir
        - Si elle est trop petite, on la fait avancer de l'incrément minimal
        - Sinon on la fait avancer de la distance entre l'obstacle et la balle
- On retire l'incrément de la distance à parcourir
- On repart au début de la boucle.

L'intérêt de cet algorithme est qu'il est robuste, il permet d'éviter que la balle passe à travers des obstacles quand sa vitesse est trop grande ou quand le temps entre deux frames devient trop long.

4. Diagramme UML

Le diagramme UML ayant une taille trop importante pour tenir sur une page A4, un fichier sous le nom : "Breakout_UML.drawio" est contenu dans le zip du projet. Celui-ci peut-être ouvert depuis Visual Studio Code avec l'extension draw.io ou via internet.

5. Remarques et résultats

A travers notre projet, nous avons accompli la plupart des objectifs que nous nous étions fixés dans le cahier des charges. Le jeu se déroule d'une manière normale. Une fois la partie lancée à l'aide du bouton « jouer » le joueur est capable d'interagir avec la plateforme à l'aide de sa souris. La balle peut rebondir sur celle-ci mais aussi sur les murs, qui délimitent l'espace de jeu. Elle est aussi capable d'enlever un point de vie à une brique lorsqu'elle en touche une. Dès lors qu'une brique n'a plus de points de vie, celle-ci disparaît. Une fois que toutes les briques ont été détruites, le joueur accède à un nouvel agencement de briques, correspondant au niveau suivant. Il ne gagne la partie que s'il réussit à gagner tous les niveaux. De plus, il a 3 balles par partie, correspondant donc à 3 vies. Son score et son nombre de vies sont aussi affichés sur l'écran tout au long que le jeu se déroule.

Certains points importants nous ont demandé beaucoup de travail. C'est le cas de la création du menu, celui-ci devant répondre à l'idée que nous avions. Celle-ci étant de moderniser le design du jeu breakout des années 1970 tout en gardant un aspect jeu d'arcade assez présent.

La gestion des rebonds de la balle sur les obstacles est l'un des points clé de notre projet. Celle-ci doit être capable, comme dit plus haut, de rebondir sur la plateforme, sur les murs, mais aussi sur les différentes briques dont le niveau est constitué. Par ailleurs, la gestion des bonus a, elle aussi, demandé certaines réflexions par les différents membres de l'équipe : identifier quels bonus pouvaient aider le joueur à gagner, sous quelles formes peuvent-ils apparaître…

Malgré notre travail, notre jeu comporte tout de même certains défauts que nous n'avons malheureusement pas eu le temps de résoudre dû à la limitation de temps donnée dans le cadre ce projet. Il ne comporte qu'un nombre fini de niveau. Cela aurait été intéressant d'avoir des niveaux générés aléatoirement, mais aussi évolutif au niveau de la difficulté lors de la partie. Dans notre cas pour gagner la partie le joueur doit finir 2 fois le même niveau.

Ce projet nous a donné une première vision du travail de groupe collaboratif en algorithmie, chose qu'aucun de nous n'avait réalisé auparavant sur un aussi gros projet. Il nous a aussi permis de nous rendre compte de l'importance de la communication que cela soit orale ou écrite lorsque que l'on travaille en équipe. Par exemple, au début du projet, certains membres n'avaient pas pour habitude de commenter leur code. Il était donc parfois compliqué pour les autres de comprendre exactement ce que la méthode en question réalisait.

Si nous avions eu plus de temps sur ce projet, nous aurions voulu mettre un système de pause permettant au joueur de stopper le jeu à l'aide d'un bouton et de reprendre lorsqu'il en avait envie ainsi que passé plus de temps à tenter de comprendre la raison d'une erreur occasionnelle d'affichage (et aurions ainsi pu la résoudre) concernant l'affichage des boutons sur le menu. Un système de niveau évolutif généré aléatoirement peut aussi être envisagé.

6. Chronologie du projet et contribution de chacun des membres

Nous avons tout d'abord réalisé un cahier des charges afin de cibler nos objectifs, quels étaient les points importants de notre algorithme. Une fois cela fait, cela nous a permis de répartir le travail et de réaliser une certaine chronologie de travail pour tout le groupe. Nous avons pu nous rendre compte assez facilement des fonctionnalités à coder qui allaient nous demander plus de travail et de réflexion que les autres.

Afin de travailler tous ensemble, nous avons tous installé Visual Studio Code, Git et Github. Cette dernière application nous a permis de gérer beaucoup de fonctionnalités en ce qui concerne le travail collaboratif et

notamment la gestion de conflits. Par ailleurs, nous avons aussi créé des branches permettant d'avoir une chronologie de notre code. Il nous était donc possible de revenir à la version du code que nous avions écrite les semaines précédentes si nous en avions besoin. De plus, grâce à Visual Studio Code, nous nous sommes aussi servis de la fonction LiveShare, permettant de voir en temps réel l'avancé du code de chacun.

La division des tâches s'est effectuée d'une manière plutôt naturelle. Nous avons chacun décidé de travailler sur une classe en particulier afin de réduire un maximum la gestion de conflits par la suite.

Théo a été chargé de la gestion du mouvement de la balle et des rebonds sur les différents objets du jeu.

Quant à Valentin, il a mis en place la fin du jeu. Il a ainsi défini les méthodes vérifiant si le jeu était achevé et le cas échéant quelle image devait s'afficher au joueur. Il s'est occupé de la création de ces fenêtres et de leur affichage. Il s'est également intéressé à la création des bonus, les a ainsi codés, et a fait en sorte qu'un de ces derniers ne s'active que si le joueur "attrapait" une étoile grâce à la plateforme.

Margaux, elle, s'est principalement chargée de la création de la plateforme et des recherches concernant le déplacement de plateforme fait par l'utilisateur à l'aide de la souris. Elle a aussi constitué le diagramme UML en parallèle de l'avancée du projet. De plus, elle a réalisé à la rédaction du rapport en accord avec tout le reste de l'équipe.

Jeremy, lui, s'est principalement occupé du menu principal ainsi que des deux fenêtres "Règles du jeu" et "À propos". Il s'est ensuite occupé de la charte graphique du projet ainsi que de l'ajout d'effets sonores, du *gameTimer* ainsi que des items bonus.

Nous avons donc décidé d'attribuer les pourcentages d'engagement suivant pour chacun des membres de l'équipe :

Théo : 25%

Valentin : 25%

Margaux : 25%

Jeremy : 25%

Bibliography

https://youtu.be/Cp5WWtMoeKg
https://michaelwalczyk.com/blog-ray-marching.html
http://jamie-wong.com/2016/07/15/ray-marching-signed-distance-functions/