Arnaud Curie | Johann Fouchard | Pascaline Guichard | Juliette Gagnepain | Guillaume Clerc | Noa Barbosa

IUT Dijon | departement informatique

Rapport

Livrable n°1



# Introduction

Le projet consiste en la réalisation d’un clone du jeu « The Binding of Isaac ». C’est un jeu indépendant d’action-aventure en 2D isométrique. Le style du jeu est volontairement gore et effrayant mais est adouci par des designs cartoons. Nous avons décidé pour notre clone de reprendre un style similaire dans l’univers d’Alice au pays des merveilles. L’objectif du projet est de réaliser la version la plus abouti possible du jeu.

Dans ce document, nous nous pencherons sur l’existant, à savoir le jeu que l’on souhaite copier. Puis nous réaliserons une étude de l’art, en recherchant les différentes caractéristiques d’autres jeux similaires, pour ainsi déceler les différents points forts et points faibles du jeu « The Binding of Isaac ». Nous pourrons ainsi ajuster notre copie en conséquence. Bien que pour ce projet le langage de programmation nous soit imposé, nous réaliserons quand même une comparaison des technologies envisageables.

# Etude de l’existant

The Binding of Isaac est un jeu de type « rogue-like ». L’objectif est de progresser dans une succession de salles/niveaux/donjons pour arriver à une fin potentielle. Les personnages incarnés se battent en tirant dans quatre direction (haut/bas/gauche/droite). Le jeu inclut des objets qui peuvent modifier la manière de jouer du joueur.

Nous nous efforcerons dans cette partie de voir les points forts, que nous pourrons réutiliser pour notre jeu, mais aussi les points faibles du jeu que nous pourrions corriger.

## Une grande diversité

The Binding of Isaac offre une grande variété d’objets et de monstres. D’autre part, plusieurs types de salles et même de cœurs existent. La variété d’objets et de monstres font partie de ce qui crée la diversité de gameplay (sa jouabilité) présente dans Isaac. Une mécanique très novatrice est la présence de différents cœurs. Les différents monstres, possédant tous des caractéristiques et des comportements uniques, ainsi que les salles font aussi partie du processus de diversification et permettent d’élargir l’univers présenté. C’est ceci qui permet de différencier Isaac des autres rogue-like avec son univers et son ambiance.

## Rejouabilité : un jeu pour passionné ?

Cependant, cette rejouabilité et cette diversité entraine une progression très faible pour le joueur. Le joueur ne se retrouvera jamais dans la même partie avec les mêmes objets, même ennemis ou salle. Cependant une fois la « connaissance » acquise sur le jeu (les différents objets, combinaisons etc…) le jeu peut vite sembler rébarbatif pour le joueur et il peut vite se lasser en voulant reproduire systématiquement le même schéma de jeu. Toutefois, instaurer un système de progression entrainant des changements sur l’expérience de jeu pour le joueur peut être intéressant.

Néanmoins, la diversité présente avec le nombre d’objets présent et l’apparition de nouveaux personnages peut amener des objectifs à certain joueurs. De plus, le jeu possède une difficulté plutôt élevée ce qui permet de garder les joueurs (les plus courageux) en créant une frustration qui donne envie de recommencer jusqu’à « vaincre le jeu ». Le jeu ne donne pas d’objectifs clair et direct, il laisse le choix au joueur de se fixer ses propres objectifs ce qui peut attirer l’attention des joueurs ou non.

## Une histoire poignante

The Binding of Isaac possède un univers très poussé et varié avec une histoire poignante, triste mais pas explicite. Ce qui amène de nombreux joueur à effectuer des recherches, à recommencer le jeu pour mieux la comprendre. Le jeu n’a pas eu besoin de se créer de pub pour faire de l’engouement, les joueurs l’ont fait d’eux même. En créant une histoire « caché, imprécise », les joueurs se sont mis à enquêter et ont partager leurs différentes trouvailles et c’est ainsi que l’engouement a été créé, permettant à ce jeu indépendant d’avoir de la visibilité. L’histoire se base sur la Bible et le sacrifice d’Isaac, nous pouvons donc nous choisir de se baser sur une autre histoire sans être explicite sur son déroulement pour attiser la curiosité des joueurs.

## Une grande variété de boss

En plus de ce qui a été dit plus tôt, Isaac possède une très grande variété de boss. Ce qui permet de surprendre le joueur pendant bon nombre de partie. Chaque boss possède des attaques et des mécaniques différentes, voire des déplacements différents. Il permet bien entendu de renforcer encore une fois l’univers présenté et chaque boss ajouté apporte des précisions à l’histoire.

## Des mouvements limités

The Binding of Isaac est un jeu facile à maitriser, tout est simple à première vue dans son gameplay. On peut se déplacer et tirer dans quatre directions uniquement. Le jeu est donc très facile à prendre en main mais sera vite dur à maitriser avec la myriade d’objets, des boss et monstres voire même salles, certaines pouvant être cruelles. On peut vite se retrouver limiter dans notre manière de se déplacer.

# Etude de l’art

Voici quelques fonctionnalités qui nous sont venues à l’esprit pour améliorer notre clone d’Isaac en étudiant comment les autres jeux de sa catégorie réussissent à supplanter ses défauts, mais aussi en cherchant des idées originales qu’ils ont développées pour pouvoir les adapter à notre projet.

## 1ère fonctionnalité : changement de salle par téléportation (Risk of Rain)

Lors d’un changement de salle, au lieu de faire passer notre personnage par une porte et avoir une animation vers la gauche, la droite, le haut ou le bas, nous pouvons ajouter un système de téléportation. Ainsi, dans quelques-unes des salles, il y aurait un objet spécial disposé dans un des coins de la salle, et ce dernier ne serait activable que lorsque tous les ennemis présents dans la salle sont morts. Un fondu au noir et l’apparition progressive de la salle suivante remplacerait l’animation de translation initiale. De plus, cette fonctionnalité est adaptée à l’univers que nous souhaitons créer pour notre jeu.

## 2ème fonctionnalité : possibilité d’esquiver (Hadès)

En ce qui concerne le déplacement d’Isaac, il serait intéressant d’ajouter un système d’esquive comme cité plus haut. Une barre de remplissage apparaitrait quelque part sur l’écran et plus le joueur inflige de dégâts aux ennemis, plus cette dernière se remplit. Nous pourrions également ajouter certains mécanismes d’esquive d’autres jeux comme la possibilité de prendre des dégâts lors de l’esquive, comme dans le jeu Dark Soul par exemple, ce qui deviendrait davantage une manière de s’éloigner du combat en cas de regroupement d’ennemis.

## 3ème fonctionnalité : épreuves de discrétion (Aragami)

Cette fonctionnalité pourrait rendre le jeu plus intéressant et moins répétitif sur la durée : des épreuves de discrétion et non de combat.

## 4ème fonctionnalité : Epreuves bonus (Five Nights At Freddy’s 4)

Connaissant le jeu vidéo Five Night At Freddy’s 4 (FNAF 4), nous avons eu l’idée d’adapter un des éléments à notre clone : les épreuves bonus. FNAF 4 est composé de 5 niveaux (5 nuits de 6 heures chacune), et entre chaque niveau, le joueur a la possibilité de faire disparaître 2h de la nuit suivante s’il réussit un mini jeu. S’il réussit, il n’a que 4 heures à jouer la nuit suivante au lieu des 6 heures habituelles. Mais s’il meurt avant d’avoir fini le niveau, alors le bonus ne s’applique et il doit recommencer la nuit avec 6 heures à jouer. Nous pouvons adapter cette mécanique à Isaac en remplaçant les 2 heures en moins par un objet bonus par exemple. Ces petits stage bonus pourrait être l’occasion d’étoffer l’histoire de notre jeu.

## 5ème fonctionnalité : Améliorations post-game (Hadès)

Une amélioration post-game (après avoir fini le jeu) pourrait aider le joueur sans trop l’avantager avec des bonus simples comme commencer la partie avec un certain nombre de pièces ou de bombes, voire commencer avec un objet aléatoire. Le joueur aurait accès à une nouvelle ressource qui lui permettrait de faire les améliorations.

## 6ème fonctionnalité : quête annexe (Faster Than Light)

Venant de la détresse observée sur Faster Than Light, cette idée pourrait diversifier les parties des joueurs en leur proposant des quêtes lors d’une partie, comme tuer un certain nombre d’ennemis ou tuer un monstre particulier avec des conditions ou non comme faire cette quête en un temps imparti ou dans le même étage.

## 7ème fonctionnalité : ajout de personnages (Risk of Rain)

Afin de pallier au manque de diversité dans la manière de combattre des personnages d’Isaac, nous pourrions y ajouter d’autres manières de combattre. Par exemple un personnage plaçant des tourelles pour combattre de la même manière que ses ennemis, un autre qui se base sur renvoyer les projectiles ennemis ou encore un personnage se battant à très courte portée avec une dague pouvant être une bonne idée pour faire référence au personnage choisi dans la maquette.

# Présentation des technologies envisageables

Pour pouvoir réaliser notre projet, le clone du jeu The Binding of Isaac, nous avons le choix à la fois entre de nombreux langages de programmation, de Framework ou même d’IDE.

Traditionnellement pour ce genre de projet les langages de programmation de haut niveau sont utilisés. Parmi tous ceux disponibles nous pouvons citer le Java, le C#, le C++, Python, Ruby et encore bien d’autre.

Pour le développement de l’application, notre client souhaite que le tout soit codé en Java. Si cela correspond à notre premier critère, nous pouvons cependant nous demander s’il cela reste cohérent et adéquat à notre projet.

Notre équipe à maîtrise et a l’habitude de manipuler du Java, du C# et du C++. Nous allons donc comparer ces trois langages entre eux et laisser de côté les autres. En effet, apprendre un nouveau langage, comme Python qui pourrait correspondre à nos besoins, pourrait nous prendre trop de temps et être contre-productif dans la réalisation du projet alors que ceux que nous connaissons pourrait être suffisant. De plus Python ne possède pas de plus-value qu’il pourrait nous apporter qui justifierai de l’apprendre durant le développement.

Une première chose qui les différentie est leur date de création : C++ a été créé dans les 1983, 1999 pour Java et 2001 pour C#. Les trois sont encore mis-à-jour. C++ est donc le plus ancien et pourrait bénéficier d’une documentation plus complète au vu de son âge, mais pourrait souffrir d’une syntaxe vieillissante. Concernant Java et C#, ils sont plus récents mais ils sont cependant plus utilisés et possède une communauté plus présente et active.

Mais ce qui pourrait expliquer cette différence de syntaxe est surtout que C# et Java sont de plus haut niveau que C++, les simplifiant. Ceci les rend un plus agréable à utiliser pour l’équipe de développement.

Une autre différence entre ces trois langages est que les projets en Java et C# sont indépendants de la plateforme sur laquelle ils se trouvent. Cela veut dire que nous pouvons coder une seule application et elle pourra s’exécuter sur n’importe quelle machine qu’importe son système d’exploitation. En revanche, il faut que l’ordinateur possède un interpréteur de ce langage qui va s’occupe de le traduire en langage que la machine pourra directement comprendre.

Prenons l’exemple de Java. Un fois le code écrit, il est compilé pour pouvoir être exécuté. Le compilateur converti le Java en bytecode. Ce byte code peut alors être exécuté sur n’importe quel système exploitation si ce dernier possède un Java Runtime Environment (JRE) compatible. Il est à noter, malgré tout, que Java s’installe plus facilement sur différent support puisqu’il est implémenté de base dans les systèmes sous Android.

Une autre différence importante est sur le temps d’exécution. Pour les projets les plus conséquents, utiliser C++ présente un grand avantage puisqu’il s’exécute plus vite que C# et Java su fait qu’il ne passe pas par une machine virtuelle. Cependant pour la taille de notre jeu, même si nous utilisons Java qui est plus lent, cela ne nous posera pas de soucis, ne nous impactant pas.

Concernant leur utilisation, les trois langages peuvent être utilisé dans le domaine vidéo ludique. C++ est utilisé pour l’Unreal Engine, C# pour Unity et Java possède de nombreux Framwork.

Finalement, nous pouvons voir une tendance se dessiner avec nos choix. Java et C# semble assez semblable, comparé à C++. Cela vient de leur date de création, très proche, mais aussi de la raison de la création de C# qui a été lancé pour concurrencer Java en proposant une version améliorée qui lui-même devait être une amélioration de C++.

Tableau : Récapitulatifs des avantages de trois langages de programmation

|  |  |  |
| --- | --- | --- |
| Java | C# | C++ |
| Multi-plateforme | Multi-plateforme plus compliqué | Multi-plateforme |
| Plus lourd et plus lent | Proche de la machine : Plus rapide | Plus lent que C++ mais moins que Java |
| Langage de plus haut niveau que C++ | Langage de plus bas niveau | Langage de plus haut niveau que C++ |

Si ces trois langages sont adaptés à notre situation, nous nous orientons plus favorablement vers Java ou C#. Java peut être intéressant à choisir puisque, dans une situation réelle, il pourrait être plus facile de recruter des personnes pour travailler sur notre projet. En effet, Java est gratuit et plus facilement accessible que C# qui est détenu par Microsoft.

Maintenant que nous savons que Java est adaptée à ce que nous souhaitons faire, nous pouvons nous demander si nous allons utiliser un framework, une bibliothèque regroupant de nombreuses fonctions pré-codées, très utile pour éviter de recoder des méthodes et pour potentiellement développer plus vite notre projet.

Pour Java, de nombreux framework sont adaptés pour la création d’un jeu. En revanche nous allons séparer les Frameworks dit graphiques comme JavaSwing ou Fx d’autre comme LibGDX ou Slick2d.

Pour le premier type, nous allons utiliser JavaFX pour notre interface. En effet JavaFX est plus récent et a été lancé comme étant le successeur de JavaSwing. Mais plus important encore, FX est plus abouti et possède des fonctionnalités plus poussées contrairement à son ainé. De plus, l’équipe de développement connait ce framework, l’ayant déjà utilisé à plusieurs reprises dans des projets précédents.

Pour la seconde catégorie, notre client ne souhaite pas avoir recours à ces framework, cependant est-ce dérangeant ?

Pas vraiment, puisque si un framework peut faire gagner du temps, apprendre à le maîtriser et à savoir l’utiliser peut couter un temps précieux. Dans notre cas, puisque nous n’avons jamais utilisé ces Framework auparavant nous risquons de ne pas savoir les utiliser correctement et de ne pas être aussi efficace que si nous codions toutes nos fonctions. Nous n’en n’utiliserons donc pas.