



UNIVERSITÉ DE MONTPELLIER
PROJET DE L2 INFO

ARKANOID

Auteurs :

BRESSAND Jérémie
MASSAVIOL Mathieu
BARRY Amadou Bailo
JULIEN Marin

Encadrant :

Mickael MONTASSIER

17 mai 2015

Table des matières

1 Problématique

Le sujet de notre Projet informatique de L2 est la production d'un jeu de type casse brique, à l'image d'[Arkanoid](#). Son but pédagogique est de fournir un programme à étudier à des étudiants en parcours Physique. La plupart des logiciels utilisés par les physiciens sont codés en Python. Une demande du client est d'utiliser ce langage pour notre projet. Aussi il nous a été demandé d'utiliser la bibliothèque tkinter, permettant une gestion simple d'éléments graphiques. Dans un souci de simplicité, on a choisi de manipuler des rectangles. Le type de jeu de notre projet nous permet d'aborder de façon pédagogique un ensemble de concept de programmation, tels que les structures de contrôles, les structures de données, la manipulation des types de base python (liste, tuple, dictionnaire) et des fichiers ainsi que la gestion des exceptions. Conformément à la demande du client, nous rendrons notre projet sous deux formes différentes, impérative et orientée objet.

Arkanoid est un jeu d'arcade conçu par Akira Fujita, développé et édité par Taito. Ce jeu sorti en 1986 sur borne d'arcade est un casse brique. C'est un jeu de dextérité où l'on contrôle un vaisseau spatial en forme de barre. Ce jeu possède une histoire simple : après la destruction par une attaque extraterrestre du vaisseau Arkanoid, un vaisseau rescapé nommé Vaus est envoyé dans une autre dimension pour se venger et détruire Doh l'alien responsable de l'attaque. Le jeu se compose d'une balle, d'une barre et de briques. Le but est de casser les briques grâce à la balle que l'on fait rebondir. Le joueur perd lorsqu'il perd la balle.

2 Cahier des charges

2.1 Fonctionnalités attendues

Le jeu dispose d'un écran d'accueil et de fin et est doté d'un système de score persistant.

Les éléments de jeu sont les briques, la barre, les murs, les balles et les bonus. Une balle rebondit sur les murs, la barre et les briques. On peut disposer de plusieurs balles. Lors de la collision balle/barre, la balle est dirigée du côté touché de la barre. La barre est contrôlée par le joueur via souris ou clavier. Le jeu sera doté des modes :

Campagne sa suite de niveaux est écrite à l'avance ;

Arcade sa suite de niveaux est générée procéduralement à partir d'une graine ;

Coop deux joueurs jouent avec deux barres en coopération.

Lorsque le joueur ne rattrape pas la dernière balle avec la barre, il perd une vie. Lorsqu'il a perdu toutes ses vies, la partie est finie. Alors, si le score du joueur entre dans le top10, il y est inscrit. Le top10 est alors réorganisé.

2.1.1 Brique

Les briques peuvent encaisser un nombre déterminé de coups qui leur est propre. Passé ce nombre, elles sont détruites. Elles peuvent avoir un type special, leur attribuant un comportement propre. On aura notamment les briques explosives : lors de leurs destructions, elles endommagent les briques environnantes. Lorsque que toutes les briques destructibles sont détruites, le niveau est fini. On passe alors à un autre niveau. Lorsqu'une brique est détruite, un nombre de points est ajouté au score du joueur. Un bonus peut alors aussi être lâché.

2.1.2 Bonus

Un bonus descend progressivement. S'ils entrent en collision avec la barre, ils deviennent effectifs. Ils déclenchent les effets suivant :

- Ajout d'une balle ;
- Balle de feu ;
- Agrandissement temporaire de la barre ;
- Retrecissement temporaire de la barre.

2.1.3 Balle

Une balle peut temporairement avoir un type special, lui attribuant un comportement propre. On aura notamment la balle de feu qui transperce les briques destructibles sur son passage. Une balle va de plus en plus vite au cours d'une même vie.

2.2 Fonctionnalités supplémentaires

1. Le joueur peut créer ses propres niveaux grâce à un éditeur de niveau.
2. Le programme gère le plein écran.
3. Gestion de la redimension de la fenêtre.
4. La barre peut être contrôlée par une manette.
5. Si le joueur termine le niveau rapidement, un niveau bonus apparaît lui permettant un gain de points.
6. Le jeu est doté d'un mode Battle : Deux joueurs partagent une même série de niveaux. Chacun a son aire de jeu, et le premier qui finit la suite de niveau remporte le match.
7. Certaines briques peuvent tirer. Lorsque la barre est touchée, elle est temporairement immobilisée. On aura les bonus supplémentaires :
 - (a) Barre gluante : La barre peut temporairement retenir une balle lors d'une collision ;
 - (b) Module laser : La barre peut temporairement tirer des lasers endommageant les briques ;
 - (c) Division des balles en trois petites de tailles inférieures et de vitesses supérieures.
8. Le jeu gère des couples de portails, permettant la téléportation de balles.
9. Lors de collision barre/mur, les éléments de jeu tremblent à la façon d'un tilt de flipper, permettant à la balle de toucher des briques qu'elle ne toucherait pas sinon.

3 Gestion du projet

Nous avons décidé d'utiliser Python3 pour notre projet. Cela nous a occasionné des problèmes de compatibilité, de nombreux modules étant prévus pour Python2. Nous verrons ici la liste des outils abordés, notre organisation et les problèmes d'ordre humain rencontrés dans le déroulement du projet.

3.1 Listes des outils abordés

- Tkinter** Bibliothèque graphique du langage python, permettant la création d'interfaces graphiques et de différentes sortes de widgets.
- Pygame** Bibliothèque graphique de python plus riche que tkinter. N'apportant rien de crucial et s'étant déjà familiarisé avec tkinter, nous ne l'avons pas utilisé.
- Pymunk** Trop complexe : Outil de gestion de contraintes et collisions pour python. Sa complexité et nos délais impartis nous ont empêché de pouvoir l'utiliser.
- PIL** Bibliothèque de chargement d'image. Obsolète.
- Pillow** Fork de PIL non obsolète. Nous n'avons pas réussi à l'utiliser.
- Pickle** Bibliothèque python permettant de transformer un objet python en code binaire et vice versa. Elle est utilisée dans la gestion des scores et des niveaux.
- Git** Outil de gestion de version de fichiers. Nous avons notamment eu recours aux :
- Branches** Nous avons utilisés une branche de développement, et une branche contenant les build stables de notre projet. Une capture d'écran de notre working tree est jointe en annexe.
 - Wiki** Nous avons utilisé le wiki interne au git pour partager l'ensemble des informations relatives au projet : dates des réunions, comptes rendus, choses à faire dans la semaine, idées, ...
 - Issues** Nous les avons utilisés pour rendre compte des bugs à corriger, des features et éventuelles améliorations à ajouter. Une capture d'écran de nos issues est jointe en annexe.
- Jabber, Skype** Jabber est un protocole de communication. Suite à des difficultés relatives à son utilisation pour faire des conférences, nous avons utilisé skype.
- Geogebra** Outil de modélisation géométrique. Il nous a permis de modéliser aisément les fonctions associés au vecteur vitesse de la balle après rebond sur la barre.
- Framapad** Service d'édition de texte collaborative en ligne.
- Ganttproject** Logiciel d'édition de diagramme de Gantt.
- LaTeX**

3.2 Organisation du travail

Le travail a été divisé en trois axes majeurs : analyse, conception de notre programme et implémentation.

La première phase s’est notamment composée de réunions pour aligner notre vision du projet. Nous avons d’abord fait de nombreuses suggestions, nous avons débattu à leurs sujets et à terme nous avons posé le cahier des charges avec notre encadrant. Au fur et à mesure de l’apprentissage des outils liés au projet, nous avons allégé notre cahier des charges dans un soucis de réalisme. La phase de conception a également nécessité plusieurs réunions pour se mettre d’accord sur la structure du projet. L’élaboration de nos algorithmes s’est fait parallèlement à leur implémentation. La troisième phase s’est composée du développement de la version objet suivie de l’impérative.

Un diagramme de Gantt et la liste des *work packages* associés sont joints en annexe.

3.3 Gestion des ressources humaines

Ce projet ayant été une première expérience de travail collaboratif, des tensions sont apparues. Elles ont été liées à des niveaux d’implication différents ou à des divergences de point de vues. Aussi les tâches ont été distribuées au sein du groupe sur la base du volontariat.

Les niveaux d’implication différents des membres du groupe ont été la source d’un gros retard de la version impérative sur la version objet.

On a observé un autre désaccord relatif au mode campagne, deux membres du groupe ne s’étant pas compris sur sa définition. D’autres désaccords de moindre ampleur ont également ponctué notre travail.

4 Développement technique

Nous avons développé notre code de manière modulaire dans l'idée de maximiser notre productivité, et de permettre une manipulation des fichiers et une gestion des bugs plus simple. L'ensemble des variables globales sont contenues dans le fichier *config.py*. On notera l'absence de constante en python.

4.1 Animations

Pour concevoir nos animations, l'idée retenue a été de décomposer une animation en deux parties : la classe *Animation* et la classe *ElemAnim*.

Un objet *Animation* contient plusieurs instances d'*ElemAnim* et ne manipule pas de surfaces dans le canvas. Son rôle est de créer, déplacer et enlever des objets de type *ElemAnim*. Elle gère ainsi la structure d'une animation, non son apparence.

Un objet *ElemAnim* contient des surfaces. Son rôle est d'ajouter, de déplacer, d'enlever et de modifier les propriétés des surfaces du Canvas. *ElemAnim* gère ainsi l'apparence d'une animation.

Cette décomposition entre structure et apparence permet de simplifier le code et de le rendre plus lisible.

Lors de la conception des animations, de nombreuses classes filles de la classe *Animation* ont été créées ainsi que pour la classe *ElemAnim*. Il était alors possible de combiner telle classe héritant de *Animation* avec telle classe héritant de *ElemAnim* ce qui donnait de la généricité dans notre code. Peu satisfaits de l'apparence de ces animations, nous n'avons gardé que peu de ces classes pour le programme final.

4.2 Collision

Notre algorithme de collision teste deux entités (balle, brique, ...) et renvoie une liste de dictionnaires que l'on appelle **flag**, qui correspond aux modifications à faire ne pouvant être faites en interne du fait de la limite de la portée des variables. Un exemple est la création d'une animation de destruction de brique, le *flag* correspondant contient alors le nom et la position de l'événement.

La stratégie adoptée pour l'algorithme est la suivante :

- En fonction de leurs directions, on teste si les deux entités testées se chevauchent à la prochaine frame – ie s'il va y avoir collision ;
- Si oui, on détermine le type de collision : sur les bord verticaux ou horizontaux ;
- On gère toutes les données de collision que la portée des variables nous permet de modifier (direction de la balle après rebond, points de vie des briques, destruction du bonus lors de son ramassage, ...) ;
- On retourne les *flags* correspondant aux événements n'ayant pu être gérés en interne.

4.3 Génération procédurale

La génération procédurale consiste en la génération de contenu de manière algorithmique, par opposition à la lecture de contenu pré-écrit. Nous utilisons ce concept dans nos modes Arcade et Coop pour générer les niveaux à partir d'une graine saisie par l'utilisateur. Une donnée supplémentaire vient compléter la génération : la difficulté. À chaque fois qu'un joueur avance de niveau, elle augmente, assignant aux malus et types de briques désavantageux des probabilités d'apparition supérieures.

Notre stratégie de génération de niveau est la suivante :

- Génération de bruit blanc plafonné entre 1 et 3, correspondant aux points de vie des briques ;
- Suppression de briques, donnant à la disposition des briques une forme autre qu'un grand rectangle ;
- La distribution des types spéciaux de briques, limités aux types explosif et indestructible dans la version rendue ;
- La distribution des bonus et malus dans les briques ;
- Nous retournons finalement notre liste de briques pour permettre son exploitation.

La fonction principale de notre génération procédurale est disponible en annexe.

4.4 Éditeur de niveaux

Parmi les fonctionnalités supplémentaires de notre cahier des charges nous avons implémenté un éditeur de niveaux qui nous a permis d'écrire facilement les niveaux de notre mode campagne.

L'éditeur se lance à partir d'un fichier annexe contenant la classe ***EditeurLvl***. Cette classe comprend comme attributs des variables correspondant aux différents paramètres du constructeur d'une brique ainsi qu'une liste de briques ainsi qu'un compteur incrémenté à chaque ajout de brique. En effet nous avons paramétré plusieurs touches pour pouvoir changer les attributs correspondant à la brique. Une dernière touche (la touche *Entrée*) permet de rajouter à la liste de briques une brique avec comme paramètre l'état actuel des attributs de la classe. Lorsque le niveau est rempli (quand le compteur de briques correspond à la taille d'un niveau définie dans ***config.py***), on appelle la methode de sauvegarde du niveau. Elle crée un nouveau fichier binaire et dans lequel on écrit grâce à la fonction *dump* de la bibliothèque ***pickle*** qui prend la liste de briques et la transforme en code binaire.

Nous n'avons pas eu le temps de créer un mode de jeu pour jouer au niveaux édités. L'utilisateur peut quand même réécrire la campagne à son gout en créant des niveaux ayant pour nom 0, 1, ... , 9 qui remplaceront les fichiers actuels.

5 Tests unitaires

On introduira ici seulement les tests unitaires nécessitant une présentation de leur fonctionnement. On notera aussi que seule la version objet, plus aboutie, en est dotée. On conseillera de les lancer à partir d'un terminal, diverses informations étant censées y être affichés.

5.1 Collision

Le test unitaire du fichier **collision.py** permet de tester les différents types de collisions. Les contrôles sont les suivants :

- clic gauche et mouvement pour déplacer la balle (lui donner une vitesse et direction)
- flèches directionnelles pour déplacer la barre

On trouve en sortie dans le terminal les éléments en collision, le type de collision détecté (vertical, horizontal) et les éventuels *flags* non traités à l'intérieur de la fonction de collision. Ils peuvent par exemple correspondre à la création d'un bonus ou d'une animation.

5.2 Génération procédurale

Le fichier **procgen.py** s'exécute avec un argument correspondant à la graine. À défaut, la graine est définie à 1337. Les petits rectangles verts correspondent à la présence d'un bonus dans une brique, les rouges à des malus. La version fournie étant réglée par défaut à une difficulté de 0%, aucun malus n'est censé apparaître.

6 Conclusion

6.1 Autocritique

Ce projet ayant été le premier de cette envergure pour nous, nous avons fait diverses erreurs même si nous restons globalement satisfait du produit final.

Tout d’abord, notre mauvaise gestion du temps et du planning a été à la source de différents problèmes. Une de nos erreurs a été de commencer à coder avant d’avoir finaliser tous nos algorithmes. Cela a mener à la production de programme contenant de nombreux bugs – on pense aux animations et plus encore aux collisions – et à terme à un code peu intuitif une fois ces bugs résolus. La version impérative a souffert d’un retard du même fait de la gestion du temps. Nous avions initialement prévu de les développer en même temps, mais étant donné l’évidente similarité des algorithmes nous avons préféré d’abord produire la version objet et l’adapter en impératif. Aussi des tensions internes ont gêné à la production de celle-ci.

Même si globalement satisfaits de notre projet, force est de constater que des améliorations importantes peuvent être faites. Par exemple, les bonus sont présentement gérés type par type. Il faudrait disposer d’un système permettant d’assigner dynamiquement tel effet sur un élément de jeu à tel bonus ; cela permettrait d’ajouter une grande quantité de bonus de manière simple. Une autre amélioration possible aurait été d’assigner de manière automatique un type d’animation à la destruction d’une brique. On aurait par ces deux améliorations largement réduit notre usage des flags lié à la collision, qui sont assez peu pratiques.

Enfin notre organisation a également souffert de divers problèmes. La répartition des tâches s’étant faite sur la base du volontariat, la charge de travail par membre a été déséquilibré. Une meilleure solution aurait été de se répartir les tâches équitablement. Nous avons programmé de manière modulaire, chose qui facilite la compréhension et la réutilisation du code. Aussi nous avons eu des problèmes d’interfaçage, bien que mineurs. Cependant, nous nous sommes trop centrés sur nos tâches respectives. Nous aurions gagné en efficacité en travaillant parallèlement dans la même salle, du fait de la stimulation et de l’échange d’idée qui en serait sorti.

6.2 Perspective

Notre programme connaît différentes perspectives d’évolution, principalement grâce à la programmation modulaire et la présence des variables globales dans le fichier `config.py`.

Certains fichiers, du fait des ajouts successifs auxquels ils ont été sujet, ont des perspectives d’évolution limité en l’état. C’est notamment le cas pour le module de collision et de gestion des différents modes de jeux.

Malgré tout, on peut adapter la dimension du jeu par une simple modification des variables globales, comme l’on peut modifier la vitesse des balles, des bonus et barres, la durée des bonus, l’intervalle de point de vie des briques, ...

L'utilisation des modules pré-existants permettent l'ajout de nouvelles fonctionnalités. Par exemple, l'ajout de l'éditeur de niveau a été postérieur aux autres modules et n'a pas demandé d'adaptation.

7 Annexe

7.1 Bibliographie

- "Apprendre à programmer avec Python 3" - Gérard Swinnen
- <http://effbot.org/>

7.2 Liste des Work Packages

Arkanoid_Gantt1

22 avr. 2015

Tâches

2

Nom	Date de début	Date de fin
Basics with python	06/02/15	09/02/15
Reflexion, viabilité	09/02/15	01/03/15
brainstorming fonctionnalités	09/02/15	15/02/15
<i>En réunion, on note toutes nos idées et nos envies sur le tramapad.</i>		
ébauche fonctionnalités	21/02/15	24/02/15
<i>On se met d'accord sur un ensemble de fonctionnalités.</i>		
Prototypes	09/02/15	23/02/15
<i>Chacun réalise un prototype de jeu arkanoid afin d'apprendre à utiliser tkinter.</i>		
Ébauche pb et cdg	25/02/15	25/02/15
<i>On rédige ensemble une ébauche de la problématique et une ébauche du cahier des charges.</i>		
Gestion sprite	23/02/15	01/03/15
<i>On réalise des animations avec l'utilisation de sprites.</i>		
Étude pygame	28/02/15	28/02/15
<i>Prise en main plus difficile que tkinter.</i>		
Étude pymunk	01/03/15	01/03/15
<i>Prise en main plus difficile que tkinter.</i>		
Étude pillow	28/02/15	01/03/15
<i>Difficultés d'installation, le module pillow est une fork en cours de développement.</i>		
Ébauche gantt	01/03/15	01/03/15
<i>On détermine ensemble la liste des workpackages et on construit le diagramme.</i>		
Algorithme	02/03/15	22/03/15
Diagrammes d'utilisation	02/03/15	03/03/15
<i>Débat puis élaboration du diagramme.</i>		
Diagramme de classes (autres).	04/03/15	04/03/15
<i>Diagramme représentant les éléments globaux du jeu : Le menu, le jeu, les scores.</i>		
Diagramme de classes (éléments jeu)	04/03/15	08/03/15
<i>Diagramme représentant les éléments basiques du jeu : Classes rectangle, balle, brique, mur, animation</i>		
Génération procédurale	09/03/15	15/03/15
<i>Fonctionnalité prenant une seed en entrée et renvoyant une liste de briques.</i>		

Arkanoid_Gantt1

22 avr. 2015

Tâches

3

Nom	Date de début	Date de fin
Animation d'explosion de briques <i>Affichage des animations.</i>	09/03/15	15/03/15
Collisions <i>Algorithme traitant l'ensemble des collisions entre les éléments du jeu.</i>	09/03/15	15/03/15
Elaboration prototype menu <i>Nous avons sauté l'étape de l'élaboration de l'algorithme du menu pour passer directement à la réalisation d'un prototype. L'arrière plan du menu est un Canvas dans lequel on voit la barre se déplacer automatiquement et la balle rebondir sur la raquette et détruire les briques.</i>	16/03/15	22/03/15
Code	21/03/15	17/04/15
OBJ : menu <i>Elaboration du menu. Il permet au joueur de démarrer une partie parmi plusieurs mode de jeu, de rentrer son pseudo et une seed. Dans l'arrière plan du menu on voit une partie se jouer de façon automatique.</i>	21/03/15	22/03/15
OBJ : Mode arcade <i>Implémentation du mode arcade en version objet.</i>	22/03/15	22/03/15
OBJ : fenêtre saisie pseudo/seed <i>Permet de récupérer le pseudo et la seed entrée par le joueur dans une fenêtre prévue à cet effet.</i>	24/03/15	24/03/15
OBJ : gestion des bonus <i>Gestion des bonus.</i>	28/03/15	28/03/15
OBJ : mode coop <i>Implémentation du mode coopératif. Deux raquettes bougent en parallèle l'une en dessous de l'autre. Chaque raquette possède ses propres touches de contrôle.</i>	04/04/15	04/04/15
OBJ : animations <i>Implémentations des animations de type feu d'artifice et explosion.</i>	07/04/15	09/04/15
OBJ : mode campagne <i>Implémentation du mode campagne permettant d'utiliser une suite de niveaux écrits dans des fichiers.</i>	08/04/15	17/04/15
OBJ : gestion des scores <i>Implémentation de la gestion des scores.</i>	14/04/15	14/04/15
OBJ : éditeur de niveaux <i>Implémentation de la classe éditeurM permettant de créer et de visualiser un niveau dynamiquement.</i>	15/04/15	17/04/15

Arkanoid_Gantt1

22 avr. 2015

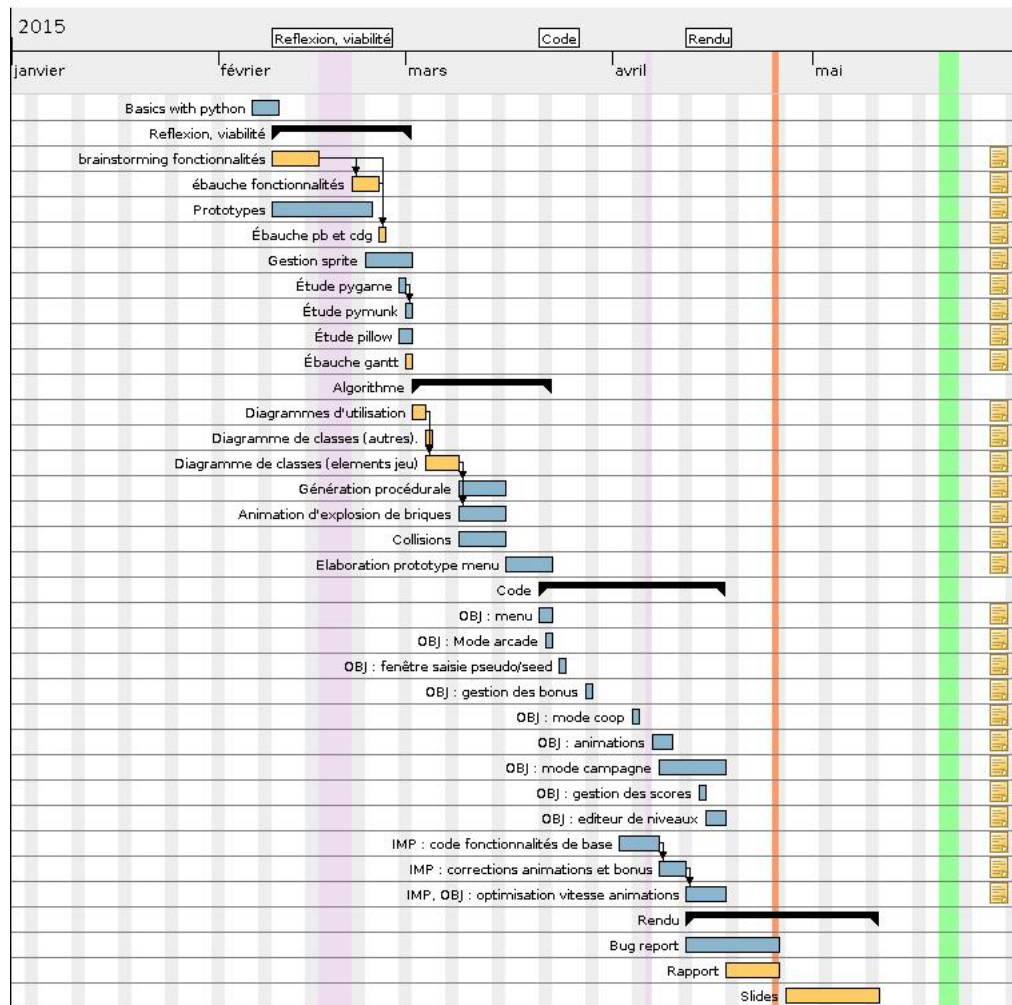
Tâches

4

Nom	Date de début	Date de fin
IMP : code fonctionnalités de base <i>Elaboration du code correspondant aux différents éléments du jeu : balle, barre, brique, bonus, animations d'artifice, animations d'explosion. Création d'une version basique du jeu.</i>	02/04/15	07/04/15
IMP : corrections animations et bonus <i>Les bonus et les animations fonctionnent. Cependant un problème de vitesse apparaît au niveau de l'affichage des animations de type feu d'artifice et explosion.</i>	08/04/15	11/04/15
IMP, OBJ : optimisation vitesse animations <i>Recherche d'optimisation des algorithmes de gestion des animations.</i>	12/04/15	17/04/15
Rendu	12/04/15	10/05/15
Bug report	12/04/15	25/04/15
Rapport	18/04/15	25/04/15
Slides	27/04/15	10/05/15

7.3 Diagramme de Gantt

FIGURE 1 – Diagramme de Gantt



7.4 collision.py : fonction procgen

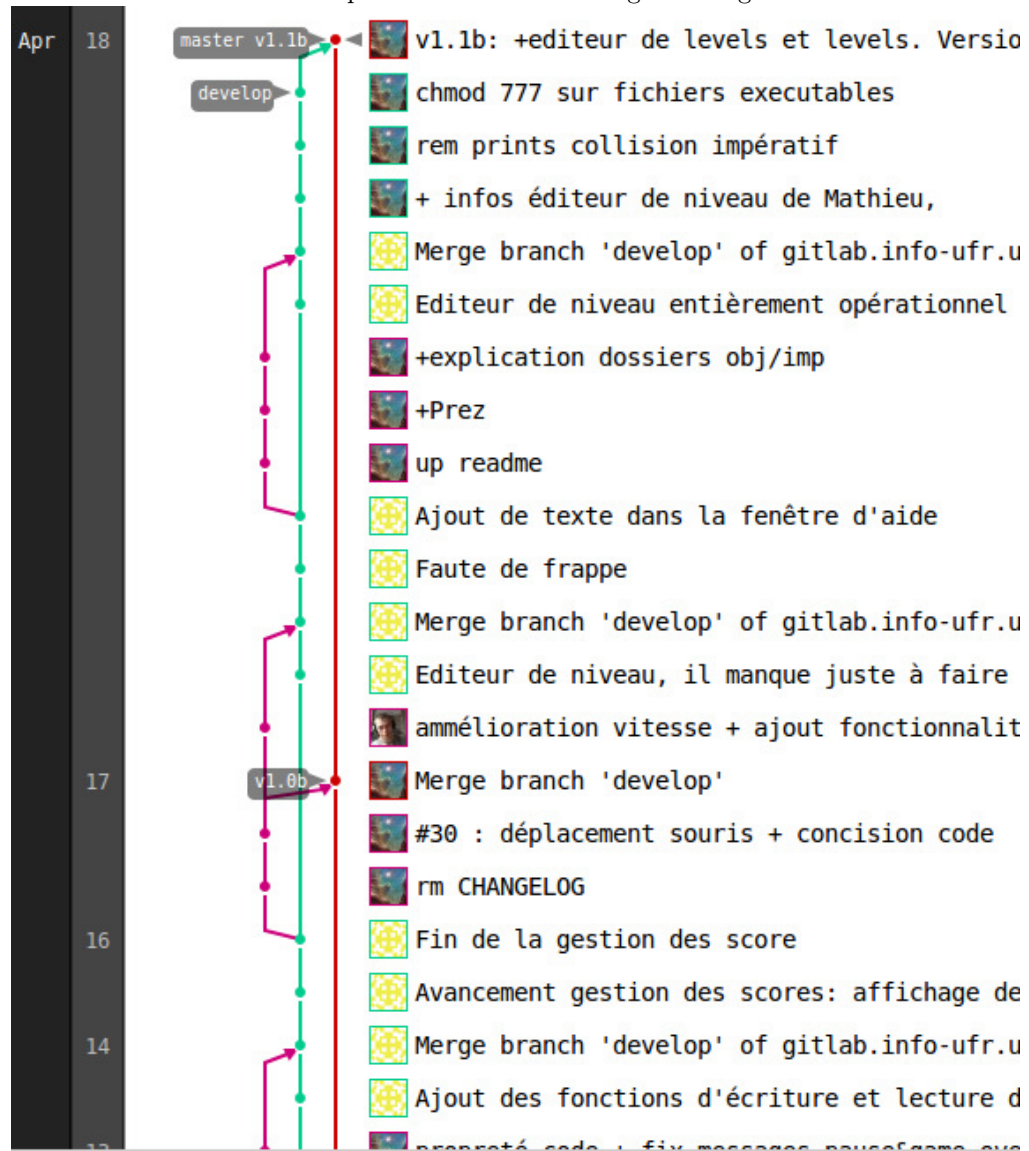
```
1  #!/usr/bin/python3
2  from random import *
3  from tkinter import *
4  import config
5
6  # utile ici que pour test unitaire
7  TESTING = False
8
9  X = config.BRIQUE_NB_X # Nb de colonne de briques
10 Y = config.BRIQUE_NB_Y # Nb de rangee de briques
11
12 # Recoit la graine et la difficulte , renvoie liste de brique
13 def procgen(graine, difficulty):
14     def enlever(tab, prent): # prent = pourcent a enlever
15         if (TESTING):
16             print(" ' Generation des trous...")
17         for x in range(X):
18             for y in range(Y):
19                 if (prent > randrange(0, 100)):
20                     tab[x][y] = 0
21         if (TESTING):
22             print(" ' 'ok")
23     #####
24     seed(graine)
25
26     # Generation bruit blanc SANS ABSENCE DE BRIQUE
27     noise = White_noise(config.BRIQUE_MIN_PV, config.BRIQUE_MAX_PV)
28
29     # Generation de trous (%)
30     enlever(noise, 30)
31
32     # Parsage du bruit en dictionnaire
33     briques = parse_noise_imp(noise)
34
35     # distribution des types de brique (%)
36     distrib_types(briques, difficulty, 20)
37
38     # distribution des bonus de brique (%)
39     distrib_bonus(briques, difficulty, 20)
40
41     # a ce stade , brique sous forme de liste de dico.
42     if (TESTING):
43         return briques
44     else:
45         return parse_imp_to_obj(briques)
```

Listing 1 – fonction procgen

Voir la section developpement technique pour de plus amples explications sur la stratégie adoptée. la fonction *seed* (l 24), du module *random*, permet de paramétrer la génération de nombre pseudo-aléatoire sur une graine. *noise* correspond à une matrice de chiffre, correspondant aux pv des futurs briques. La fonction *enlever* vient redéfinir certain de ces chiffres à 0. La variable *TESTING* n'est vrai que si et seulement si le programme est en test unitaire.

7.5 Working tree






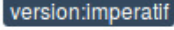



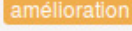


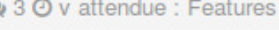



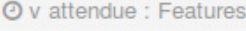













FIGURE 2 – Capture d'écran du working tree de git



On pourra au passage noter notre gestion du système de tag

7.6 Issues

FIGURE 3 – Capture d'écran des issues de git

Mode coop : gestion barres	CLOSED
#27 assigned to  Julien Marin  	updated 11 days ago
problème vitesse version impérative	
#26 assigned to  Bressand Jeremy   	updated 14 days ago
Gestion game over	CLOSED
#25   	updated 11 days ago
Mode campagne	CLOSED
#24 assigned to  Massaviol Mathieu   	updated 14 days ago
Mode coop	CLOSED
#23 assigned to  Julien Marin   	updated 16 days ago
Gestion de pause	CLOSED
#22 assigned to  Julien Marin  	updated 9 days ago
Explosion: crash, tentative de suppression d'élément inexistant	CLOSED
#21 assigned to  Bressand Jeremy  	updated 17 days ago
Ne pas afficher le bandeau en mode "auto"	CLOSED
#20 assigned to  Julien Marin  	updated 16 days ago
Gérer les animations dans thread	
#19 assigned to  Bressand Jeremy  	updated 16 days ago

7.7 Bug report

- × Lorsque la balle est coincée entre la barre et le mur et que la barre fait pression contre la balle, la balle est éjectée de l'aire de jeu.
- ✓ Bug variation vitesse balle lors de la collision avec la barre : Lors de la collision entre une balle et une barre, la vitesse de la balle varie de manière inattendue.
 - Le problème a été résolu par un passage par angle, permettant d'obtenir une norme constante – dans notre cas, une vitesse.
- × Lorsque la barre reçoit le bonus d'agrandissement et qu'elle se trouve près du mur, elle se retrouve coincée dans celui ci.
- ✓ Des briques supposées avoir un nombre de point de vie différents sont détruites au bout du même nombre de coups.
 - Le problème était dû à une comparaison "inférieure" au lieu de "inférieure ou égale".
- × Lors de collision balle/barre, il arrive dans de rares cas que la balle ne rebondisse pas de manière logique : elle inverse sa direction horizontale au lieu de sa direction verticale, ou inversement.
- ✓ Crash du jeu lorsqu'une brique explosive en détruit plusieurs autres.
 - Le programme essayait de supprimer des briques déjà supprimées. Nous avons donc effectué la suppression avec la gestion de l'explosion.
- ✓ En mode coopératif, la position des barres des deux joueurs s'inversent, et une barre n'est pas supprimée - laissant une troisième barre "fantôme".
 - Rien de conceptuel, nous avons simplement codé le mode coop trop rapidement.
- ✓ Vitesse anormale de la balle lorsqu'elle heurte un bord vertical d'une barre en mouvement.
 - On ajoutait la vitesse de la barre à la balle, afin d'éviter qu'elles se retrouvent l'une dans l'autre. On a résolu le problème en plafonnant la vitesse horizontale de la balle à celle de la barre dans ce cas.
- × Lorsque l'on commence une partie après en avoir finie une (ie être retourné au menu), le jeu commence en pause
- × En version imperative, le bonus d'ajout de balle crée une balle bien plus lente que les autres jusqu'à collision avec la barre.